A short guide to choosing initial parameter values for the estimation in moveHMM

Théo Michelot & Roland Langrock

May 8, 2023

1 Introduction

In the package moveHMM, hidden Markov models are fitted by numerical maximisation of the likelihood. The function `fitHMM` uses the optimiser `nlm` for this purpose. The optimiser uses a numerical (Newton-type) procedure to explore the parameter space, with the aim of identifying the model parameters that maximise the likelihood function. Roughly speaking, the optimiser uses the derivatives of the likelihood function to iteratively update the parameter values such that the likelihood value increases over the iterations. The situation can be compared to that of a hiker who tries to find the highest peak in a mountain range by always walking uphill until reaching a peak. To do this, `nlm` requires a starting point in parameter space, from where to start the search. In moveHMM, that starting point is passed to `fitHMM` with the arguments 'stepPar0' (starting values for the parameters of the step length distributions) and 'anglePar0' (starting values for the parameters of the turning angle distributions). In the following, we refer to 'initial' parameter values and 'starting' parameter values interchangeably.

The choice of the starting values is not part of the model specification, and the starting values themselves are not meaningful for the interpretation of the model. However, it is important to choose them well, to avoid convergence issues in the optimisation of the likelihood. An example of optimisation failure is illustrated in Figure 1. The example shows how, for a simple likelihood function of one parameter, different starting values can lead to different estimates. In the analogy of the hiker searching for the highest peak in a mountain range, the peak reached by just walking uphill may be the highest peak close to where they started, but there may be an even higher peak somewhere else in the landscape. The likelihood of a hidden Markov model is typically a high-dimensional function of many parameters, but the same problem can arise.

In this document, we provide suggestions to select starting parameter values. We focus on a 2-state model, although the ideas described here can be applied to models with more states. We consider the default distributions for the movement variables: the gamma distribution for the step lengths, and the von Mises distribution for the turning angles. For more details about the model specification and model fitting in moveHMM, please refer to the main vignette of the package.

2 Data preparation

We consider the haggis data set described by Michelot et al. (2016), and automatically loaded with moveHMM.

```
# Load package
library(moveHMM)
```

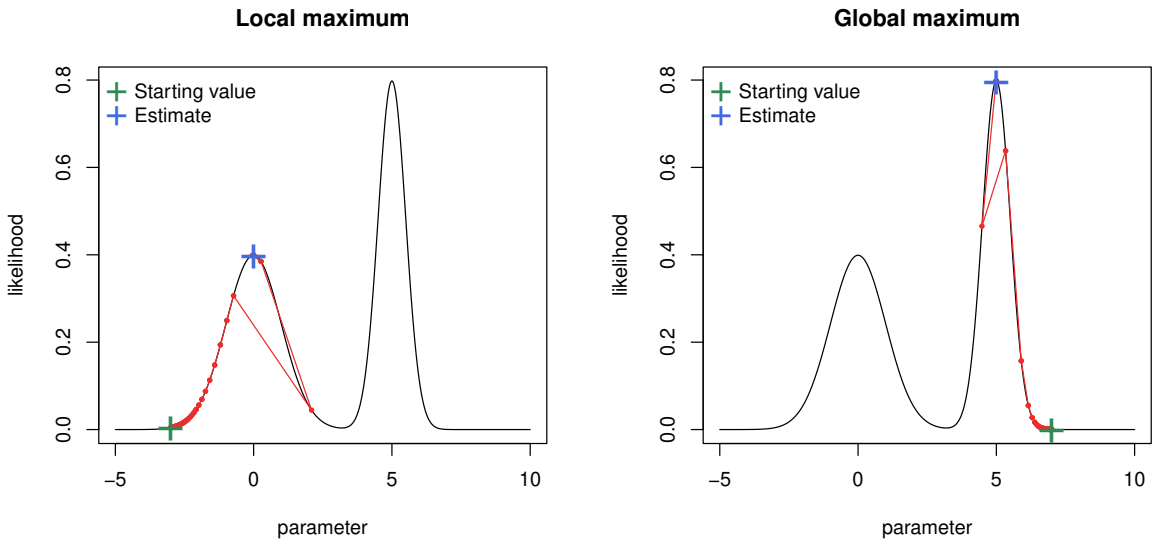


Figure 1: *Example of convergence issue. The optimisation was started from two different points, and the red line shows the steps of the optimiser in both cases. On the left, the starting value was too far from the maximum likelihood estimate, and the optimiser got stuck in a local maximum of the function.*

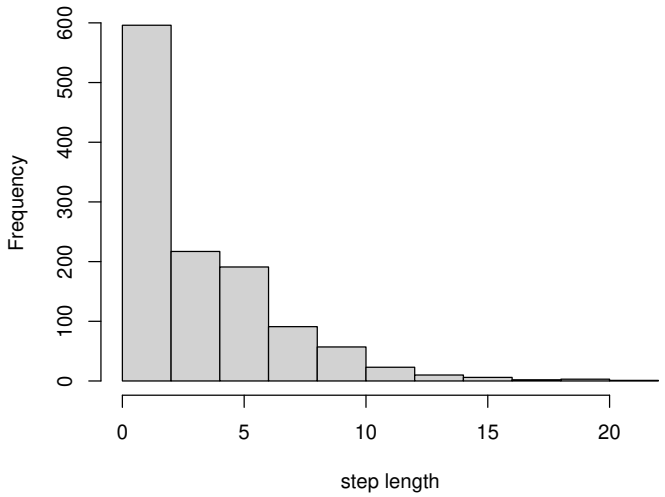
```
# Derive step lengths and turning angles
hmmdata <- prepData(haggis_data, type = "UTM")

# Display first rows of data set
head(hmmdata)
```

3 Step length parameters

We first consider the starting values for the parameters of the gamma distributions of step lengths. In moveHMM, the gamma distribution has two parameters, the mean and the standard deviation. In a 2-state model, we therefore need four starting parameters for the step lengths, two means and two standard deviations.

The general idea, also applicable for the turning angle distribution, is to select 'plausible' values for the starting parameters, given the data. The rationale is that, the closer our starting point is to the optimal point (of the likelihood function), the more likely we are going to find the optimal point. Indeed, as shown in Figure 1, convergence issues can arise when the starting parameter values are too far from the maximum likelihood estimates (in parameter space). We want to choose initial parameter values that are not too far from the estimates, and we can make an educated guess by inspecting the data. A good starting point is often to plot a histogram of the observations.

```
# Plot histogram of step lengths
hist(hmmdata$step, xlab = "step length", main = "")
```



In this data set, the observed step lengths range between 0 and about 15km. From this, we know for example that the mean parameter of the step length distribution is smaller than 15km in both states, and that we should not use starting values larger than 15km. We need to choose two initial mean parameters, one for each state. In most analyses based on 2-state hidden Markov models, one of the states captures periods of little movement activity (e.g. resting or foraging) and the other state captures periods with higher movement activity (e.g. travelling). We can therefore select one starting value near the lower end of the range of observed step lengths (for state 1, say), and the other starting value such that the corresponding state can capture the steps which we would intuitively associate with more active behaviour (for state 2). Based on the histogram, we may choose an initial mean equal to say 1 km for state 1, to capture the short step lengths of the data set. In state 2, we may take a starting value equal to 5 or 10 km, to capture the longer step lengths.

When the gamma distribution is used to model step lengths, the standard deviation is usually of the same order of magnitude as the mean of the distribution. For example, if the mean is 1, we would typically expect the standard deviation to be around 1, rather than around 0.01 or around 100. In our example, we also expect the standard deviation to be smaller in state 1 (short step lengths) than in state 2 (long step lengths). Here, we simply choose the same starting values for the mean and the standard parameter in each state. (If in doubt, it is usually better to choose larger rather than smaller standard deviations, such that the corresponding distributions capture a wide range of possible steps. Otherwise, the risk of the optimiser ending up in the wrong part of the parameter space is higher.)

```
# Starting values for the step length parameters
stepMean0 <- c(1, 5) # initial means (one for each state)
stepSD0 <- c(1, 5) # initial standard deviations (one for each state)
stepPar0 <- c(stepMean0, stepSD0)
```

If there were steps of length zero in the data, an additional parameter would be needed for the step length distribution: the zero mass parameter. In each state, the zero mass gives the proportion of step lengths exactly equal to zero, because those cannot be modelled with the gamma distribution. The initial zero mass parameters should be between 0 and 1, and it can usually be set to the proportion of steps of

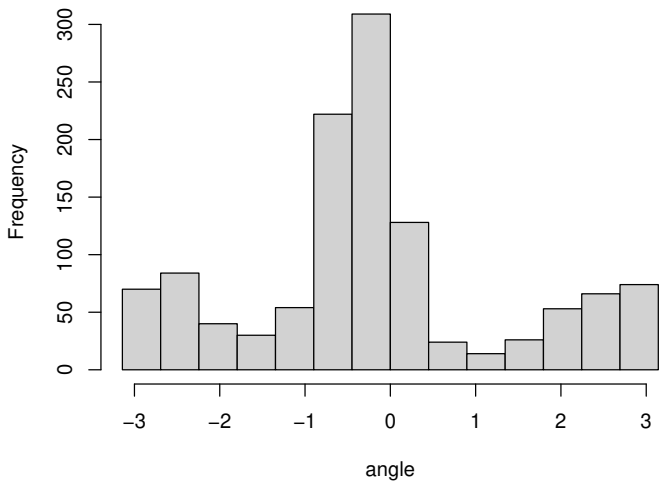
length zero in the data, obtained as follows. Here, we do not include it, because there are no steps of length zero in the data.

```
# Indices of steps of length zero
whichzero <- which(hmmdata$step == 0)

# Proportion of steps of length zero in the data set
length(whichzero)/nrow(hmmdata)

## [1] 0
```

4 Turning angle parameters

The von Mises distribution has two parameters, the mean turning angle, and the concentration. The mean turning angle is between $-\pi$ and π, and the concentration is a positive number which measures how concentrated the turning angles are around the mean. The concentration parameter can be viewed as an inverse measure of variance: the larger the concentration, the smaller the variance. We need to select two initial means, and two initial concentration parameters. The figure below shows a histogram of the observed turning angles.

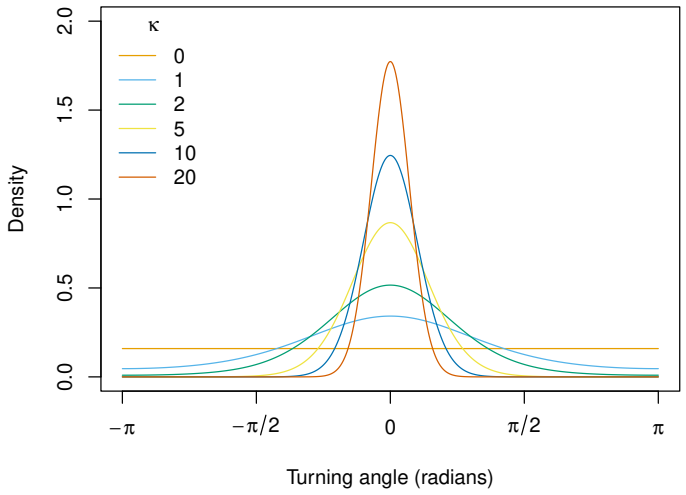
```
# Plot histogram of turning angles
hist(hmmdata$angle, breaks = seq(-pi, pi, length = 15), xlab = "angle", main = "")
```



The mean turning angle is usually close to 0, which corresponds to the case where there is persistence in the direction of movement. Indeed, in the histogram above, we see that the distribution of observed turning angles has a peak close to 0. There is also a peak of turning angles around π (or $-\pi$). This typically happens for low-resolution data, when the animal's movement is undirected and clustered. At a low resolution, the animal may appear to change direction at each time step, which results in a cluster of turning angles around $-\pi/\pi$. In most applications, we would not expect any other value than 0 or π for the mean turning angle.

Here, we set the initial turning angle mean to π in state 1, as we expect that slow movement will also be associated with more changes in direction, and 0 in state 2, because fast movement tends to be directional.

The concentration of the von Mises distribution measures how concentrated the turning angles are around the mean value. A large concentration parameter indicates a peaked distribution centred on the mean, i.e. strong persistence in direction if the mean is zero. On the other hand, a concentration parameter close to 0 corresponds to a uniform distribution of turning angles over $(-\pi, \pi]$, i.e. undirected movement. It is difficult to give general guidelines about what is a 'large' or a 'small' value for the concentration parameter, because that may depend on the study species and the time resolution of the data. It is therefore good to try different values, to get a better feel for the shape of the von Mises distribution. The plot below shows the von Mises probability density function, for different values of the concentration parameter κ.



We need to select one initial concentration parameter value for each state of the model. We usually expect the concentration to be larger in the state with larger step lengths. Indeed, fast movement tends to be directed (e.g. 'transiting' behaviour), and slow movement tends to be undirected (e.g. 'resting' or 'foraging' behaviour). So, we choose a larger initial concentration parameter in state 2, for which we also chose a larger initial mean step length parameter. Eventually, we define the starting values as follows.

```
# Starting values for the turning angle parameters
angleMean0 <- c(pi, 0) # initial means (one for each state)
angleCon0 <- c(1, 10) # initial concentrations (one for each state)
anglePar0 <- c(angleMean0, angleCon0)
```

We can then pass the starting values to the optimiser, in the function `fitHMM`.

```
m <- fitHMM(data = hmmdata, nbStates = 2, stepPar0 = stepPar0, anglePar0 = anglePar0)
```

5 Try many starting values

There is always a risk of convergence failure and, in general, it is not enough to try one set of starting values. Instead, we should fit the model with many different sets of starting values, and select the best model fit

among those. Going back to the example of the hiker in search of the highest peak in a mountain range, if they simply walk uphill from their starting point, then it would be best to have them start the search from many different locations. Even if there is only *one* initial location which is close to the highest peak in the mountain range, then the chances are very good that they will find it, even if all the other searches lead to other (local) peaks.

One possible way to do this is to generate starting values at random, from a distribution of plausible values. Here, we suggest using uniform distributions over plausible ranges of values, determined by inspecting the histograms as before. In R, we use a 'for' loop to iterate over several sets of initial parameters. At each iteration, we generate random parameter values with the function `runif`, and we pass them to `fitHMM` as starting values. We save all the fitted models in a list.

```
# For reproducibility
set.seed(12345)

# Number of tries with different starting values
niter <- 25

# Save list of fitted models
allm <- list()

for(i in 1:niter) {
    # Step length mean
    stepMean0 <- runif(2,
                       min = c(0.5, 3),
                       max = c(2, 8))

    # Step length standard deviation
    stepSD0 <- runif(2,
                     min = c(0.5, 3),
                     max = c(2, 8))

    # Turning angle mean
    angleMean0 <- c(0, 0)

    # Turning angle concentration
    angleCon0 <- runif(2,
                       min = c(0.5, 5),
                       max = c(2, 15))

    # Fit model
    stepPar0 <- c(stepMean0, stepSD0)
    anglePar0 <- c(angleMean0, angleCon0)
    allm[[i]] <- fitHMM(data = hmmdata, nbStates = 2, stepPar0 = stepPar0,
                        anglePar0 = anglePar0)
}
```

The object 'allm' is a list of 25 fitted models, and we want to compare their likelihoods, to find the best-fitting model. We can extract the negative log-likelihood values of the fitted models.

```
# Extract likelihoods of fitted models
allnllk <- unlist(lapply(allm, function(m) m$mod$minimum))
allnllk
```

```
##  [1] 3462.487 3462.487 3462.487 3462.487 3462.487 3462.487 3462.487 3462.487
##  [9] 3462.487 3462.487 3462.487 3462.487 3462.487 3462.487 3462.487 3462.487
## [17] 3462.487 3462.487 3462.487 3462.487 3462.487 3462.487 3462.487 3462.487
## [25] 3462.487
```

We see that, in all runs, the optimiser converged to the same maximum likelihood value. This is usually a sign of numerical stability: regardless of the starting values, the optimiser managed to find the maximum likelihood estimates. However, it can happen that, in some of the experiments, the optimisation fails, and this would lead to a smaller likelihood, i.e. a larger negative log-likelihood. To present the results of the analysis, we keep the best-fitting model, i.e. the model with largest likelihood.

```
# Index of best fitting model (smallest negative log-likelihood)
whichbest <- which.min(allnllk)

# Best fitting model
mbest <- allm[[whichbest]]
mbest
```

```
## Value of the maximum log-likelihood: -3462.487
##
## Step length parameters:
## ----------------------
##        state 1  state 2
## mean 0.9977758 5.009974
## sd   0.4951517 3.044710
##
## Turning angle parameters:
## ------------------------
##                  state 1    state 2
## mean          -3.109854 -0.3082704
## concentration  1.035838  8.7681838
##
## Regression coeffs for the transition probabilities:
## --------------------------------------------------
##              1 -> 2    2 -> 1
## intercept -1.736608 -2.002033
##
## Transition probability matrix:
## -----------------------------
##           [,1]      [,2]
```

```
## [1,] 0.8502557 0.1497443
## [2,] 0.1189896 0.8810104
##
## Initial distribution:
## --------------------
## [1] 0.06367063 0.93632937
```

6 Speed things up with parallel

For complex models or large data sets, the procedure described in Section 5 can be very time-consuming, because the model needs to be fitted many times. The computational cost can be mitigated using parallelised routines in R. In the example of Section 5, the 25 model fits are independent, and they can be run on separate cores. In the code below, we use the R package parallel to fit 25 models in parallel.

```
# Package for parallel computations
library(parallel)

# Create cluster of size ncores
ncores <- detectCores() - 1
cl <- makeCluster(getOption("cl.cores", ncores))
# Export objects needed in parallelised function to cluster
clusterExport(cl, list("hmmdata", "fitHMM"))

# Number of tries with different starting values
niter <- 25

# Create list of starting values
allPar0 <- lapply(as.list(1:niter), function(x) {
    # Step length mean
    stepMean0 <- runif(2,
                       min = c(0.5, 3),
                       max = c(2, 8))

    # Step length standard deviation
    stepSD0 <- runif(2,
                     min = c(0.5, 3),
                     max = c(2, 8))

    # Turning angle mean
    angleMean0 <- c(0, 0)

    # Turning angle concentration
    angleCon0 <- runif(2,
                       min = c(0.5, 5),
```

```
                        max = c(2, 15))

    # Return vectors of starting values
    stepPar0 <- c(stepMean0, stepSD0)
    anglePar0 <- c(angleMean0, angleCon0)
    return(list(step = stepPar0, angle = anglePar0))
})

# Fit the niter models in parallel
allm_parallel <- parLapply(cl = cl, X = allPar0, fun = function(par0) {
    m <- fitHMM(data = hmmdata, nbStates = 2, stepPar0 = par0$step,
                anglePar0 = par0$angle)
    return(m)
})

# Then, we can extract the best-fitting model from allm_parallel
# as before
```

References

Michelot, T., Langrock, R., and Patterson, T. A. (2016). moveHMM: an R package for the statistical modelling of animal movement data using hidden Markov models. *Methods in Ecology and Evolution*, 7(11):1308–1315.