Model Selection, Checking & State Decoding

hmezer

2025-07-31

We consider a time series giving the step lengths of the female elephant Habiba in the package ‘moveHMM‘.

```r
install.packages("moveHMM")
```

```
## Installing package into '/cloud/lib/x86_64-pc-linux-gnu-library/4.5'
## (as 'lib' is unspecified)
```

```r
library(moveHMM)
```

```r
rawdata <- read.table("http://www.rolandlangrock.com/HMMs/elephant_rawdata.txt",
header=T)
head(rawdata)
```

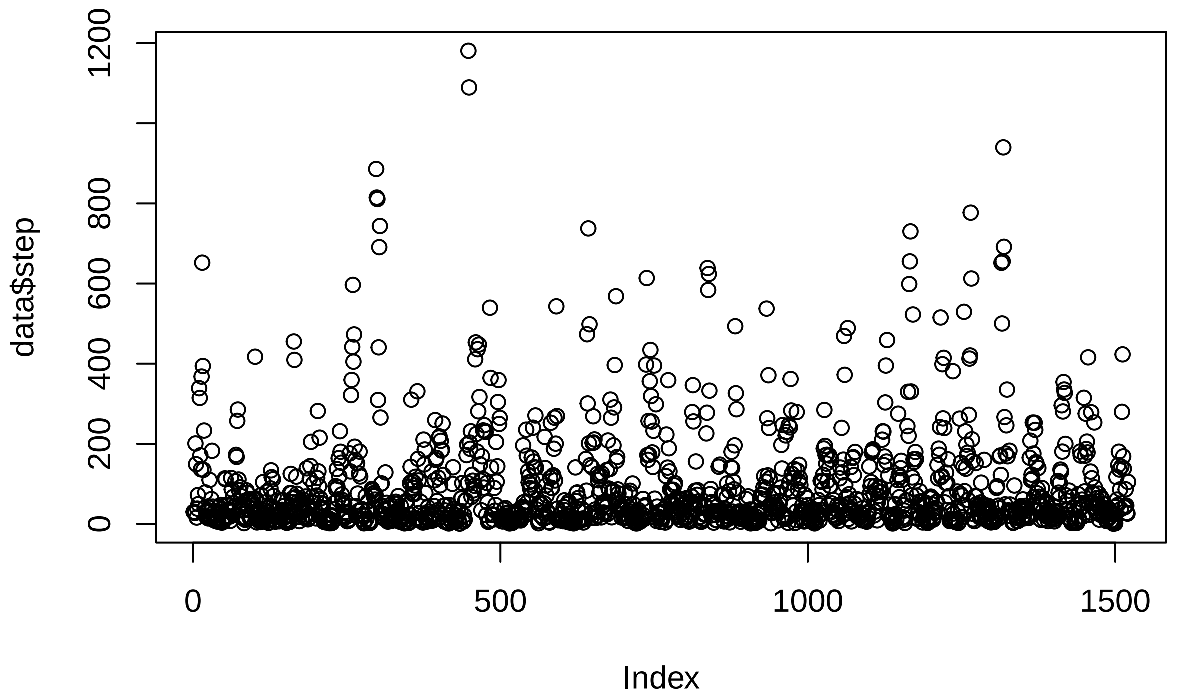
```
##               x        y               timestamp
## 21973 340611.9 62801.22 2014-01-23 11:30:42.000
## 21974 340624.2 62828.47 2014-01-23 11:45:11.000
## 21975 340633.1 62857.41 2014-01-23 12:00:13.000
## 21976 340659.8 62860.71 2014-01-23 12:15:13.000
## 21977 340852.4 62804.10 2014-01-23 12:30:11.000
## 21978 340988.1 62742.88 2014-01-23 12:45:13.000
```

```r
data <- prepData(rawdata, type="UTM", coordNames=c("x","y"))
head(data)
```

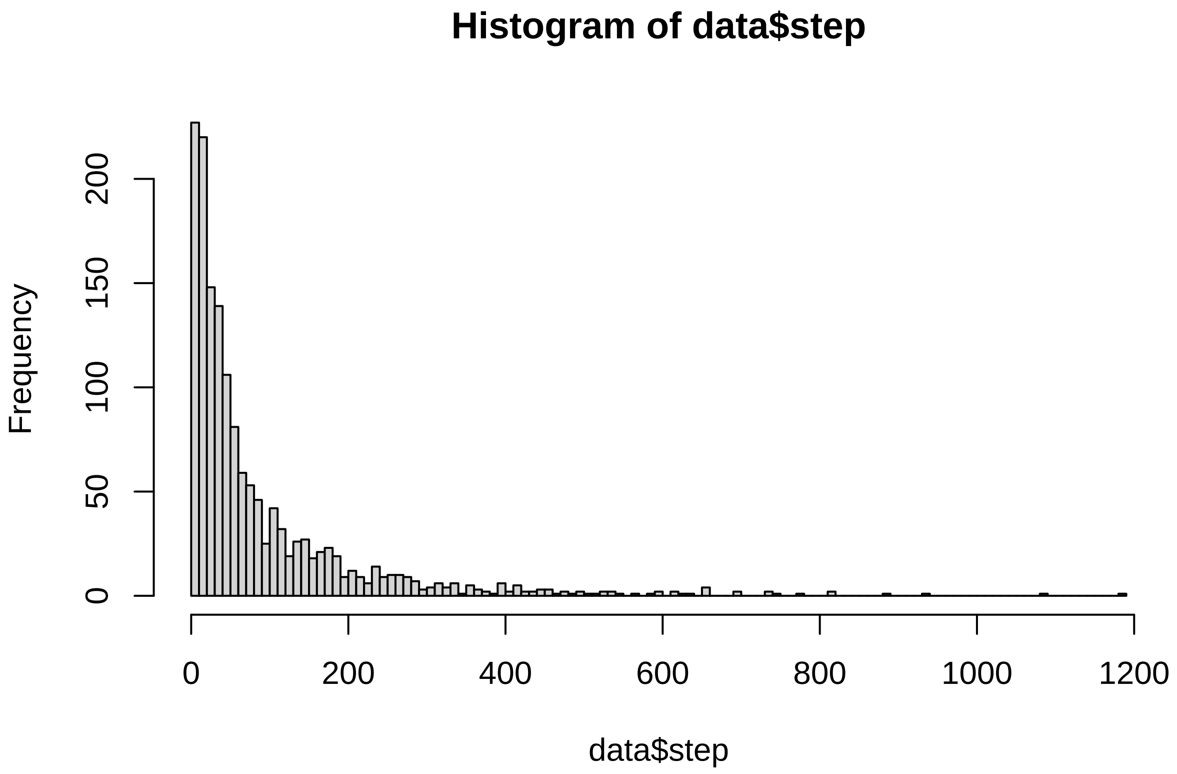
```
##        ID      step      angle        x        y               timestamp
## 1 Animal1  29.87805         NA 340611.9 62801.22 2014-01-23 11:30:42.000
## 2 Animal1  30.28453  0.1237818 340624.2 62828.47 2014-01-23 11:45:11.000
## 3 Animal1  26.91687 -1.1492332 340633.1 62857.41 2014-01-23 12:00:13.000
## 4 Animal1 200.69303 -0.4088491 340659.8 62860.71 2014-01-23 12:15:13.000
## 5 Animal1 148.93990 -0.1376260 340852.4 62804.10 2014-01-23 12:30:11.000
## 6 Animal1  15.82305 -0.5497463 340988.1 62742.88 2014-01-23 12:45:13.000
```

To determine the number of states and candidate parameters for the initial distribution, we consult the histogram:

```r
plot(data$step)
```



```
hist(data$step, breaks=100)
```



We fit a 2-state model:

```
stepMean2 <- c(25, 100) # mean vector of gamma distribution (step lengths)
stepSD2 <- c(10, 30) # SD vector of gamma distribution (step lengths)
stepPar2 <- c(stepMean2, stepSD2)
mod2 <- fitHMM(data,
               nbStates=2,
               stepPar0=stepPar2,
               verbose=1,
               angleDist="none",
               stationary=T)
```

```
## iteration = 0
## Step:
## [1] 0 0 0 0 0 0
## Parameter:
## [1]  3.218876  4.605170  2.302585  3.401197 -1.500000 -1.500000
## Function Value
## [1] 11160.99
## Gradient:
## [1]  3082.62143   566.72336 -2444.69331 -6131.00806    17.99539   -13.91029
```

```
## Warning in nlm(f = nLogLike, p = wpar, nbStates = nbStates, bounds = bounds, :
## NA/NaN replaced by maximum positive value
## Warning in nlm(f = nLogLike, p = wpar, nbStates = nbStates, bounds = bounds, :
## NA/NaN replaced by maximum positive value
```

```
## iteration = 33
## Parameter:
## [1]  3.365238  5.229511  3.193954  4.942918 -2.641668 -2.117366
## Function Value
## [1] 7922.079
## Gradient:
## [1] -4.392401e-04 -3.961611e-04  5.323197e-04  1.354384e-04  3.228045e-05
## [6]  5.045170e-05
##
## Relative gradient close to zero.
## Current iterate is probably solution.
```

We fit a 3-state model:

```
stepMean3 <- c(5, 50, 150) # mean vector of gamma distribution (step lengths)
stepSD3 <- c(5, 10, 50) # SD vector of gamma distribution (step lengths)
stepPar3 <- c(stepMean3, stepSD3)
mod3 <- fitHMM(data,
               nbStates=3,
               stepPar0=stepPar3,
               verbose=1,
               angleDist="none",
               stationary=T)
```

```
## iteration = 0
## Step:
##  [1] 0 0 0 0 0 0 0 0 0 0 0 0
## Parameter:
##  [1]  1.609438  3.912023  5.010635  1.609438  2.302585  3.912023 -1.500000
##  [8] -1.500000 -1.500000 -1.500000 -1.500000 -1.500000
## Function Value
## [1] 9294.366
## Gradient:
##  [1]  -568.508294  1564.336528   220.587067  -306.597448  -677.672830
##  [6] -1981.787414   -41.382820    32.458988   -70.684296    -5.418658
## [11]    40.281062    -2.595033
```

```
## Warning in nlm(f = nLogLike, p = wpar, nbStates = nbStates, bounds = bounds, :
## NA/NaN replaced by maximum positive value
```

```
## iteration = 88
## Parameter:
##  [1]  3.188337  4.640428  5.788146  2.979941  4.146594  5.215611 -2.322311
##  [8] -5.269481 -1.629989 -2.175968 -4.255974 -1.192184
## Function Value
## [1] 7842.911
## Gradient:
##  [1] -5.539686e-04 -6.148323e-04 -2.603653e-04  1.629798e-04  8.633040e-04
##  [6]  8.457397e-05 -3.575613e-04  3.676309e-05  1.422839e-04 -9.195395e-06
## [11]  4.060269e-06 -2.113181e-04
##
## Relative gradient close to zero.
## Current iterate is probably solution.
```

Finally, we fit a 4-state HMM:

```
stepMean4 <- c(5, 30, 100, 300) # mean vector of gamma distribution (step lengths)
stepSD4 <- c(5, 10, 25, 150) # SD vector of gamma distribution (step lengths)
stepPar4 <- c(stepMean4, stepSD4)
mod4 <- fitHMM(data,
               nbStates=4,
               stepPar0=stepPar4,
               verbose=1,
               angleDist="none",
               stationary=T)
```

```
## iteration = 0
## Step:
##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## Parameter:
##  [1]  1.609438  3.401197  4.605170  5.703782  1.609438  2.302585  3.218876
##  [8]  5.010635 -1.500000 -1.500000 -1.500000 -1.500000 -1.500000 -1.500000
## [15] -1.500000 -1.500000 -1.500000 -1.500000 -1.500000 -1.500000
## Function Value
## [1] 8228.915
## Gradient:
##  [1] -362.459043  139.055795  584.544515  216.753221  122.660725 -466.291334
##  [7] -240.968567 -105.110637  -39.719092   16.976745   32.444694  -16.922892
## [13]    5.100000   39.160475   19.546411  -36.022514   -1.866028   34.350085
## [19]    6.854785   -7.432995
```

```
## Warning in nlm(f = nLogLike, p = wpar, nbStates = nbStates, bounds = bounds, :
## NA/NaN replaced by maximum positive value
```

```
## iteration = 136
## Parameter:
##  [1]   2.312195   3.612622   4.861916   5.905087   1.956030   3.114501
##  [7]   4.210841   5.271723  -1.104223  -2.603545  -4.824480  -1.654802
## [13]  -2.087830  -4.423115  -5.911706  -1.429120  -2.186579 -15.376580
## [19]  -2.398733  -1.236545
## Function Value
## [1] 7786.559
## Gradient:
##  [1] -1.142673e-03 -1.220004e-03  4.169681e-04  7.266610e-04  1.697605e-03
##  [6] -3.460430e-04  2.481712e-04 -3.800687e-04  2.380353e-04 -7.929777e-05
## [11]  3.280189e-05  0.000000e+00 -2.121456e-04 -1.583298e-05  2.769235e-06
## [16]  1.406448e-04  1.580588e-05  1.892738e-06 -7.583126e-06 -1.676970e-04
##
## Relative gradient close to zero.
## Current iterate is probably solution.
```

Which model is the best fit among these three? We can utilize the information criteria to guide us:

```
# AIC
aic <- numeric(3)
aic[1] <- 2*mod2$mod$minimum + 2*length(mod2$mod$estimate)
aic[2] <- 2*mod3$mod$minimum + 2*length(mod3$mod$estimate)
aic[3] <- 2*mod4$mod$minimum + 2*length(mod4$mod$estimate)
aic
```

```
## [1] 15856.16 15709.82 15613.12
```

```
which.min(aic)
```

```
## [1] 3
```

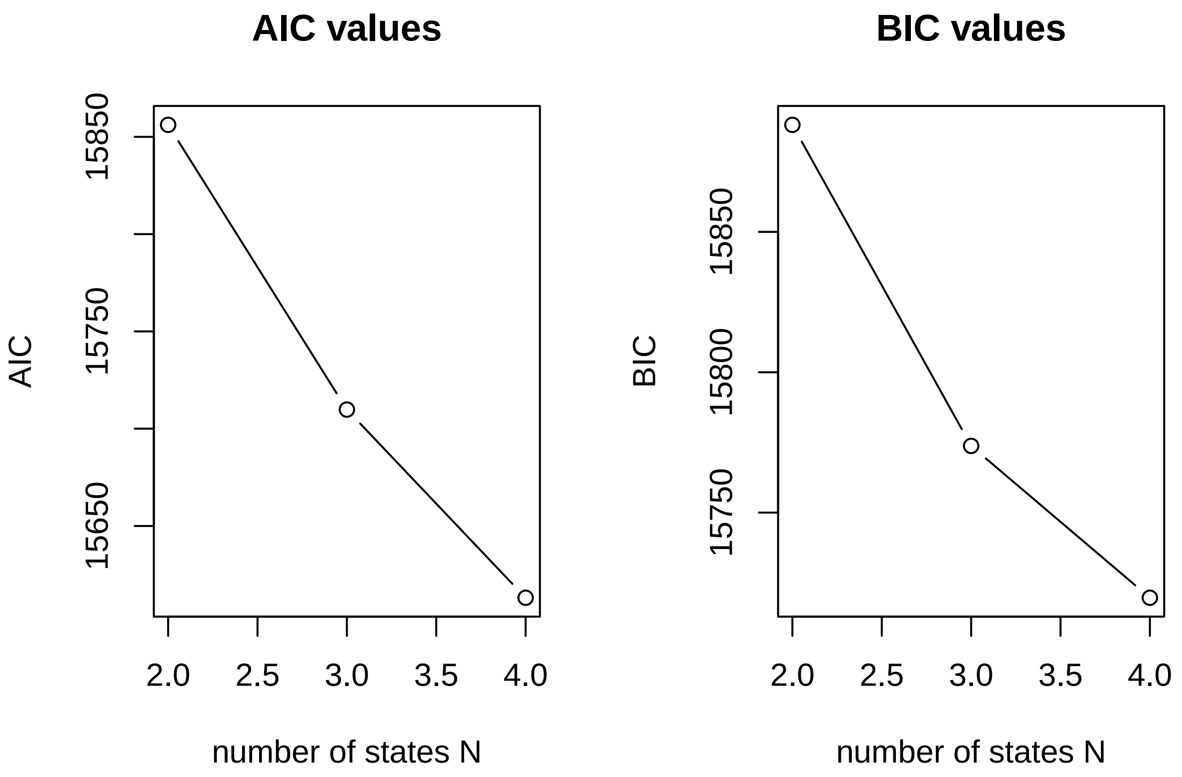
```
# BIC
bic <- numeric(3)
T <- dim(data)[1]
bic[1] <- 2*mod2$mod$minimum + log(T)*length(mod2$mod$estimate)
bic[2] <- 2*mod3$mod$minimum + log(T)*length(mod3$mod$estimate)
bic[3] <- 2*mod4$mod$minimum + log(T)*length(mod4$mod$estimate)
bic
```

```
## [1] 15888.12 15773.76 15719.67
```

```
which.min(bic)
```

```
## [1] 3
```

```
par(mfrow=c(1,2))
plot(2:4,aic,type='b',main='AIC values',xlab='number of states N',ylab='AIC')
plot(2:4,bic,type='b',main='BIC values',xlab='number of states N',ylab='BIC')
```



```
mod2$mle
```

```
## $stepPar
##       state 1  state 2
## mean 28.94037 186.7015
## sd   24.38466 140.1787
##
## $anglePar
## NULL
##
## $beta
##              1 -> 2    2 -> 1
## intercept -2.641668 -2.117366
##
## $delta
## [1] 0.6176254 0.3823746
##
## $gamma
##           [,1]       [,2]
## [1,] 0.9334956 0.06650444
## [2,] 0.1074204 0.89257962
```

```
mod3$mle
```

```
## $stepPar
##       state 1   state 2  state 3
## mean 24.24808 103.58869 326.4074
## sd   19.68666  63.21832 184.1243
##
## $anglePar
## NULL
```

```
##
## $beta
##              1 -> 2    1 -> 3    2 -> 1    2 -> 3    3 -> 1    3 -> 2
## intercept -2.322311 -5.269481 -1.629989 -2.175968 -4.255974 -1.192184
##
## $delta
## [1] 0.5417274 0.3293862 0.1288865
##
## $gamma
##            [,1]       [,2]        [,3]
## [1,] 0.90645971 0.08887539 0.004664898
## [2,] 0.14963129 0.76369109 0.086677616
## [3,] 0.01076032 0.23036280 0.758876882
```

```
mod4$mle
```

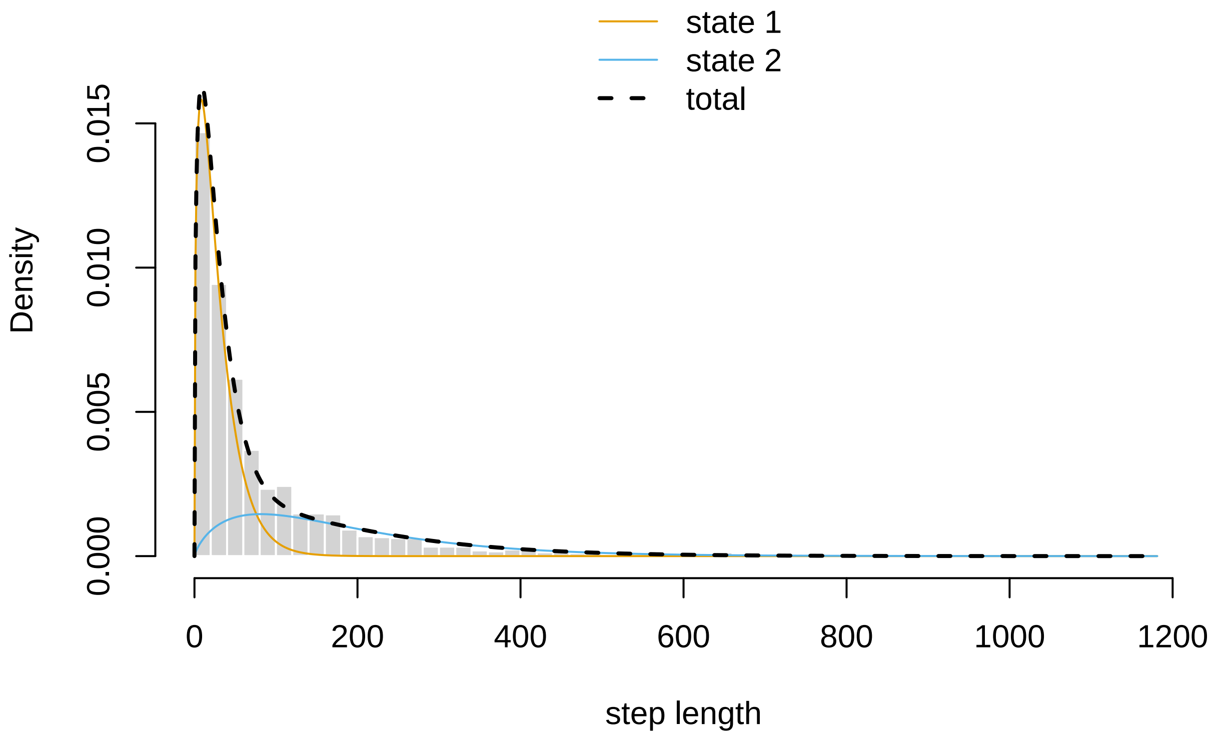
```
## $stepPar
##        state 1  state 2  state 3  state 4
## mean 10.096558 37.06309 129.2716 366.8990
## sd    7.071197 22.52219  67.4132 194.7513
##
## $anglePar
## NULL
##
## $beta
##              1 -> 2    1 -> 3   1 -> 4    2 -> 1   2 -> 3    2 -> 4    3 -> 1
## intercept -1.104223 -2.603545 -4.82448 -1.654802 -2.08783 -4.423115 -5.911706
##             3 -> 2    3 -> 4    4 -> 1    4 -> 2    4 -> 3
## intercept -1.42912 -2.186579 -15.37658 -2.398733 -1.236545
##
## $delta
## [1] 0.20863821 0.42001865 0.27168173 0.09966142
##
## $gamma
##              [,1]      [,2]       [,3]        [,4]
## [1,] 7.074587e-01 0.2345003 0.05235956 0.005681407
## [2,] 1.440227e-01 0.7535326 0.09340479 0.009039980
## [3,] 1.998900e-03 0.1768289 0.73826485 0.082907373
## [4,] 1.519755e-07 0.0657629 0.21023882 0.723998132
```

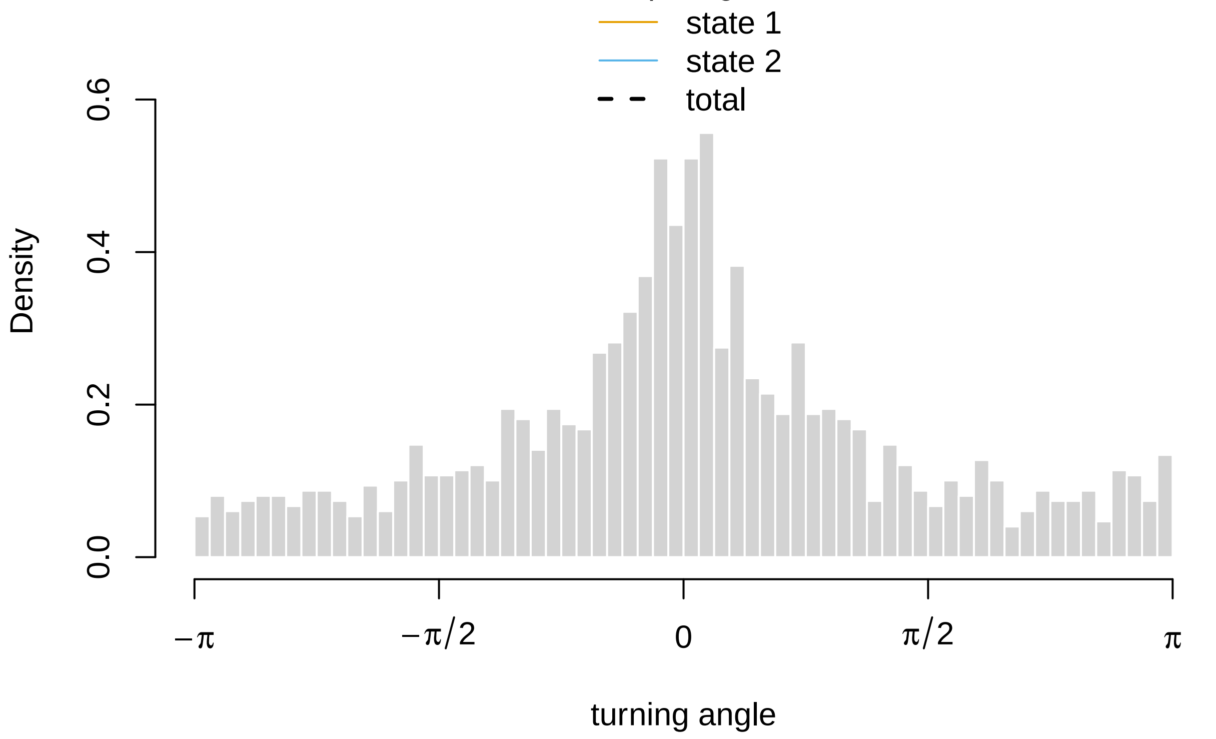
When we observe the states, especially in the 4-state setting, it is apparent that the states rather "mop up" the structure.

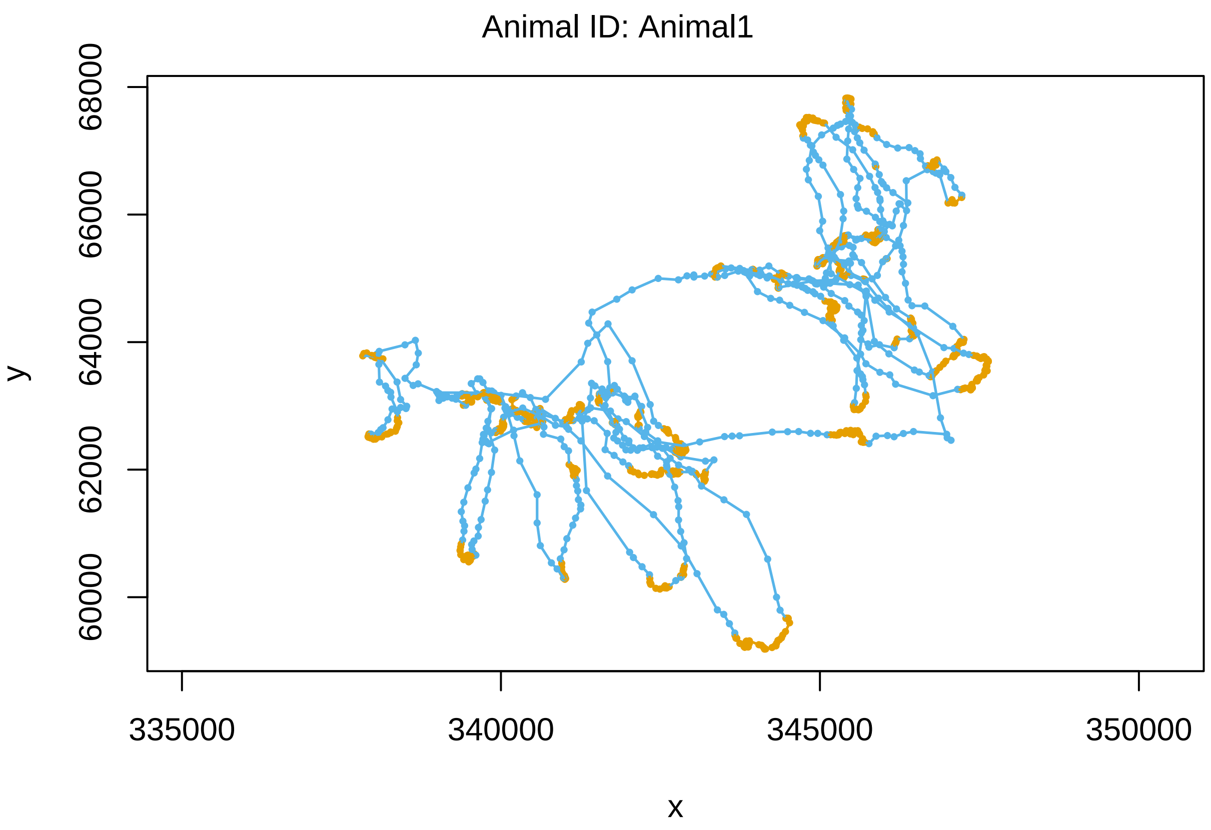
Now, considering this set of candidate models for the data set we have, how could we select the best one? What criteria to utilize besides the information criteria? Let's plot the marginal distribution of the states to see how well they fir to empirical distributions:

```
plot(mod2, breaks=60)
```

```
## Decoding states sequence... DONE
```

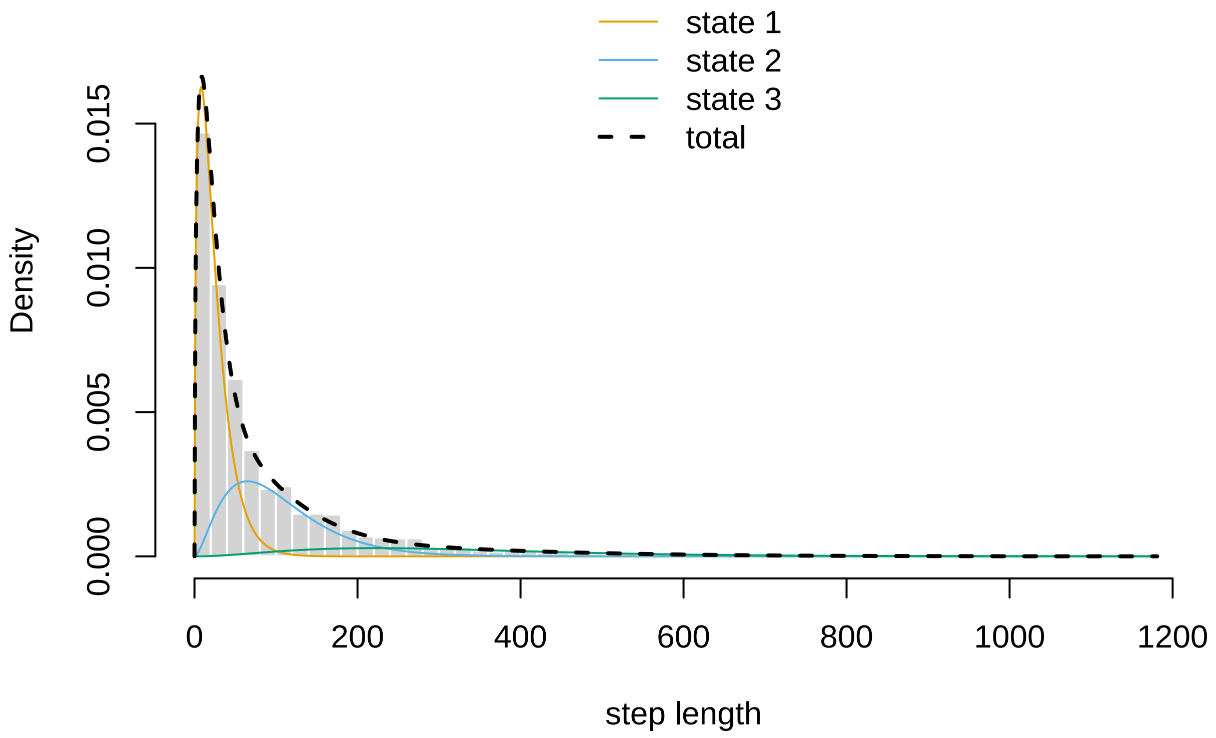

state 1
state 2
total
Density
0.000
0.005
0.010
0.015
0
200
400
600
800
1000
1200
step length

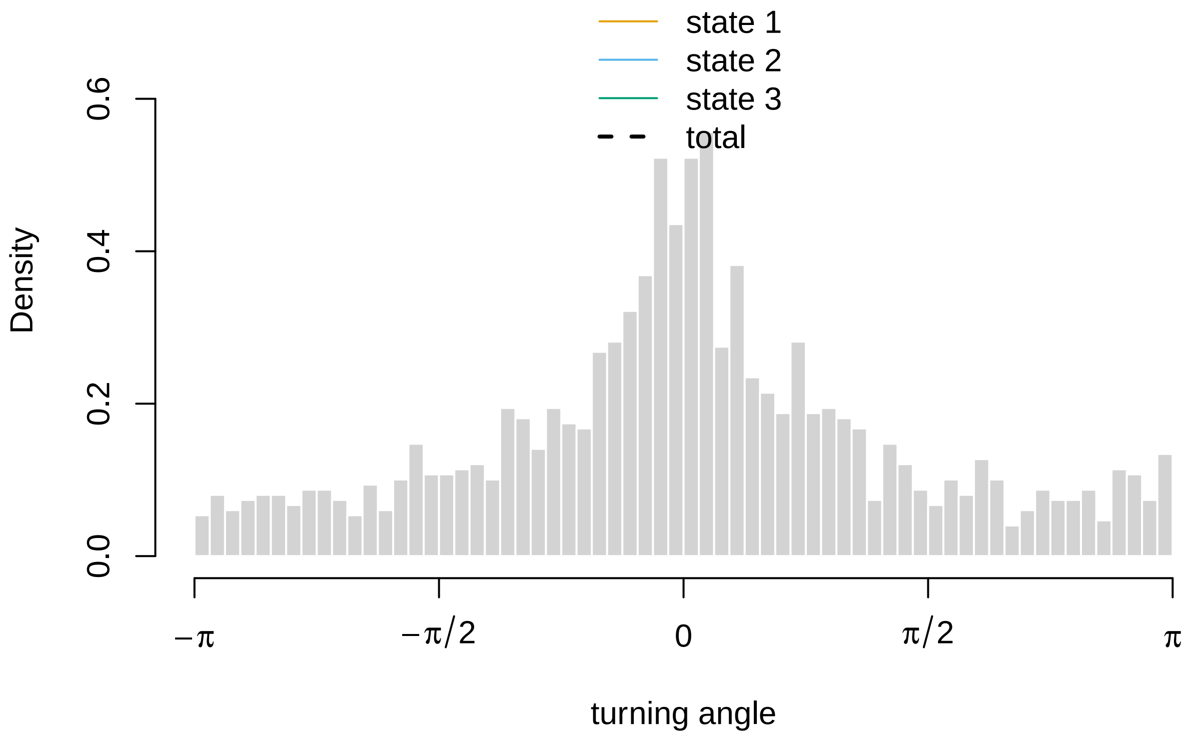

state 1
state 2
total
Density
0.0
0.2
0.4
0.6
−π
−π/2
0
π/2
π
turning angle

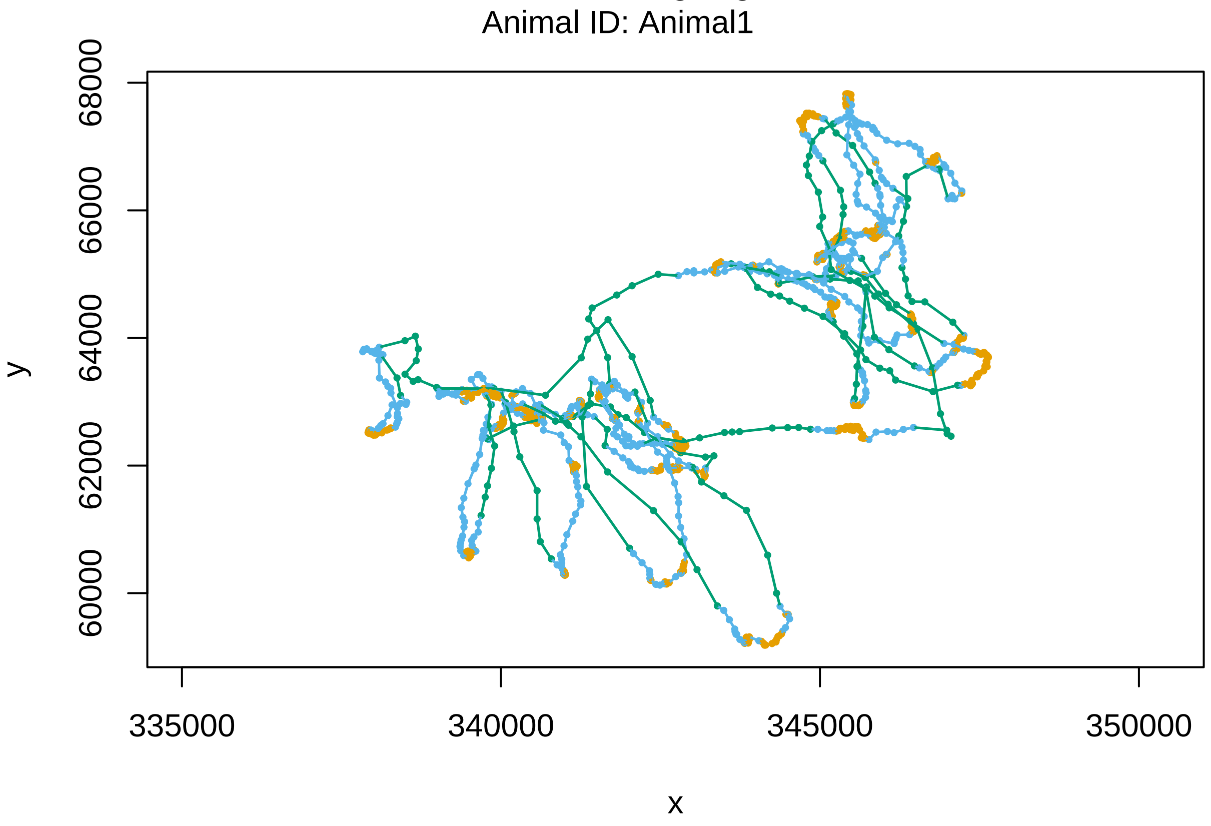


```
plot(mod3, breaks=60)
```

```
## Decoding states sequence... DONE
```

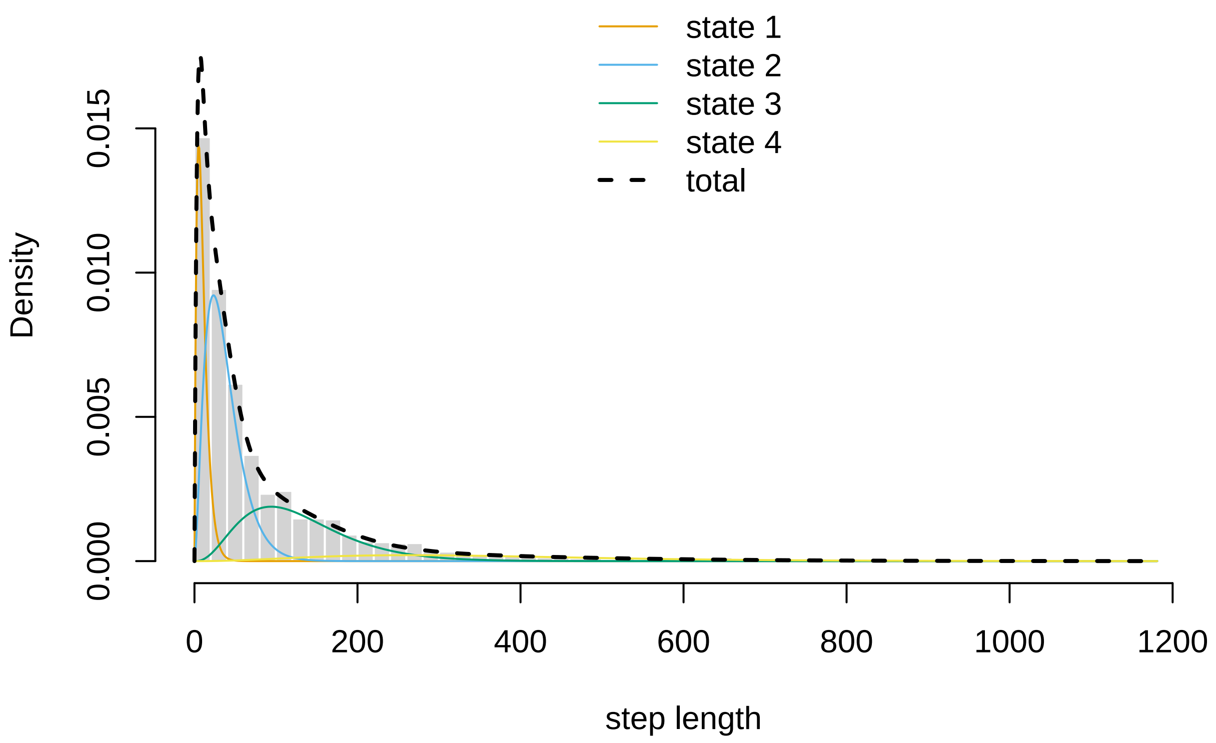


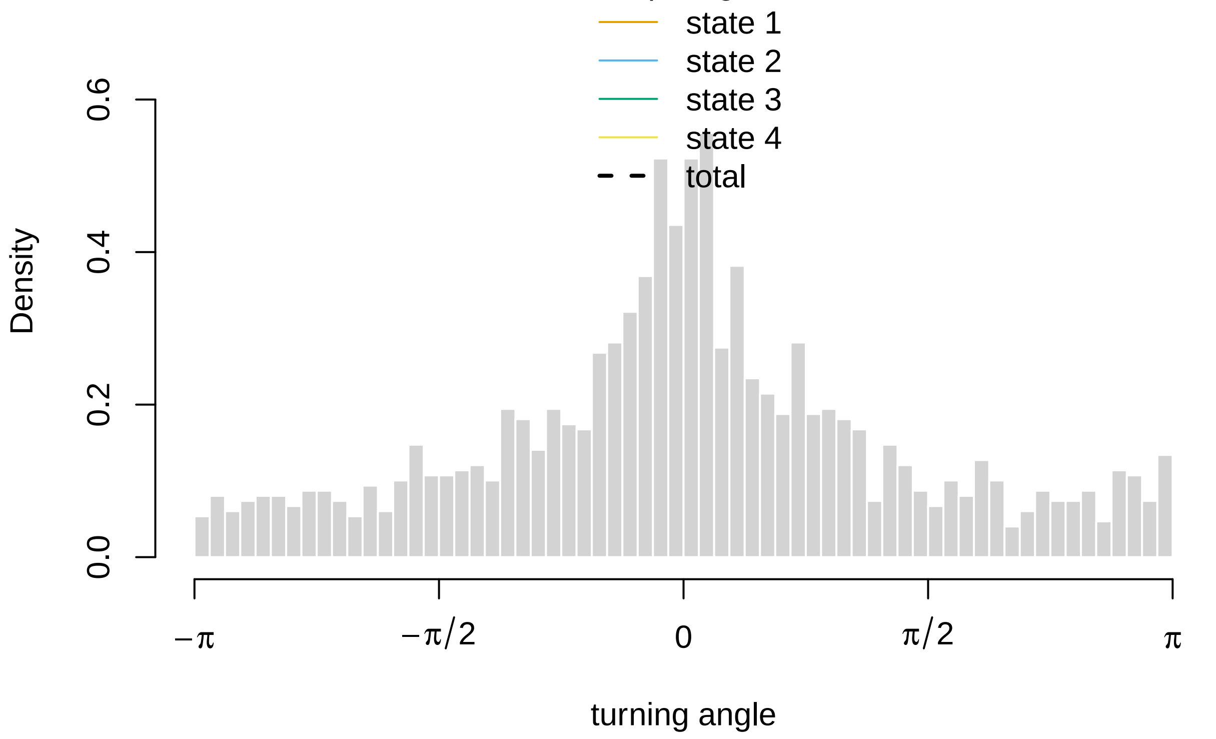


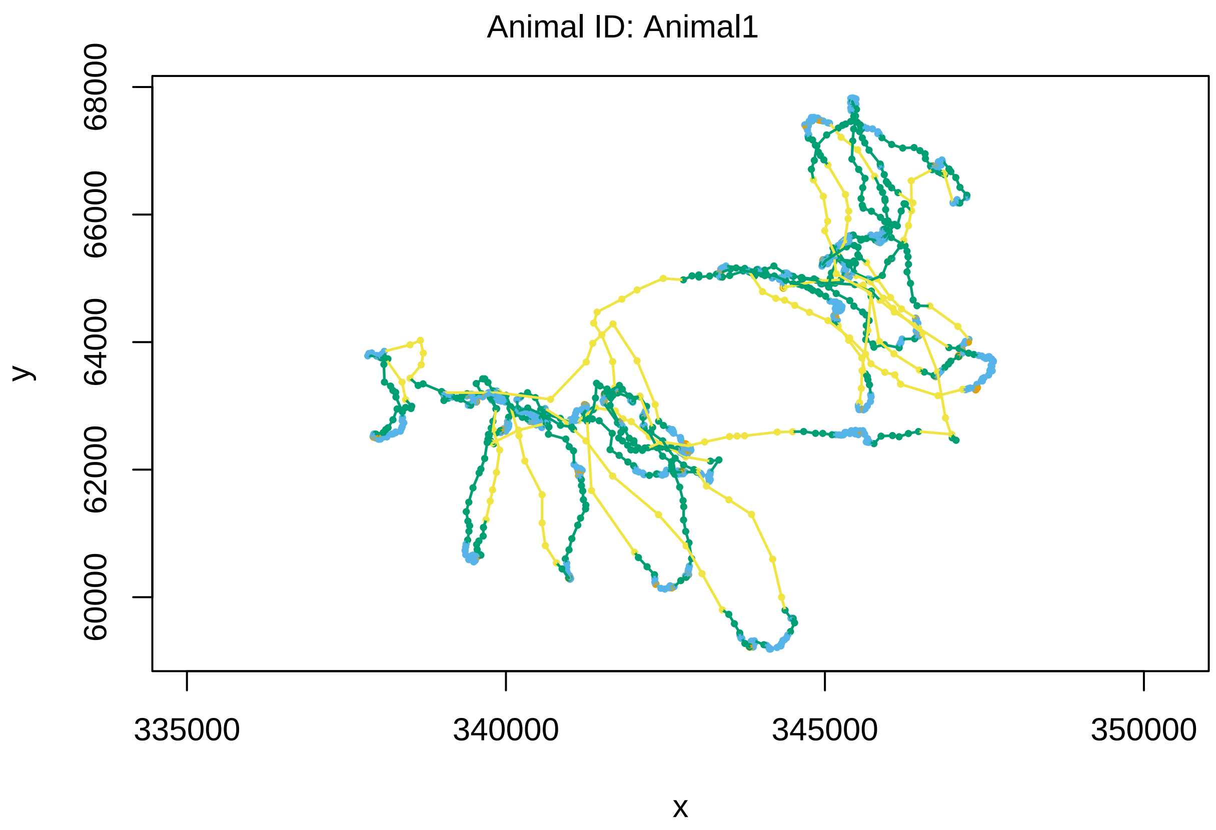


```r
plot(mod4, breaks=60)
```

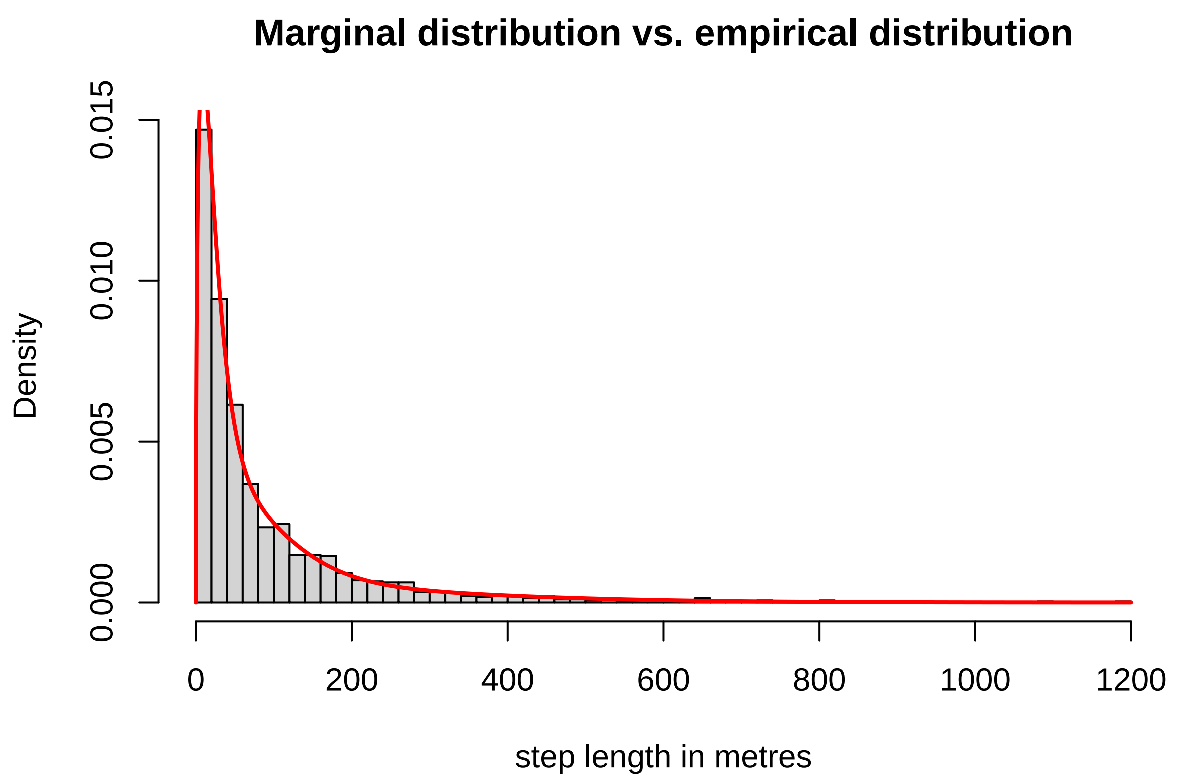
```
## Decoding states sequence... DONE
```


state 1
state 2
state 3
state 4
total
Density
0.000
0.005
0.010
0.015
0
200
400
600
800
1000
1200
step length


state 1
state 2
state 3
state 4
total
Density
0.0
0.2
0.4
0.6
−π
−π/2
0
π/2
π
turning angle



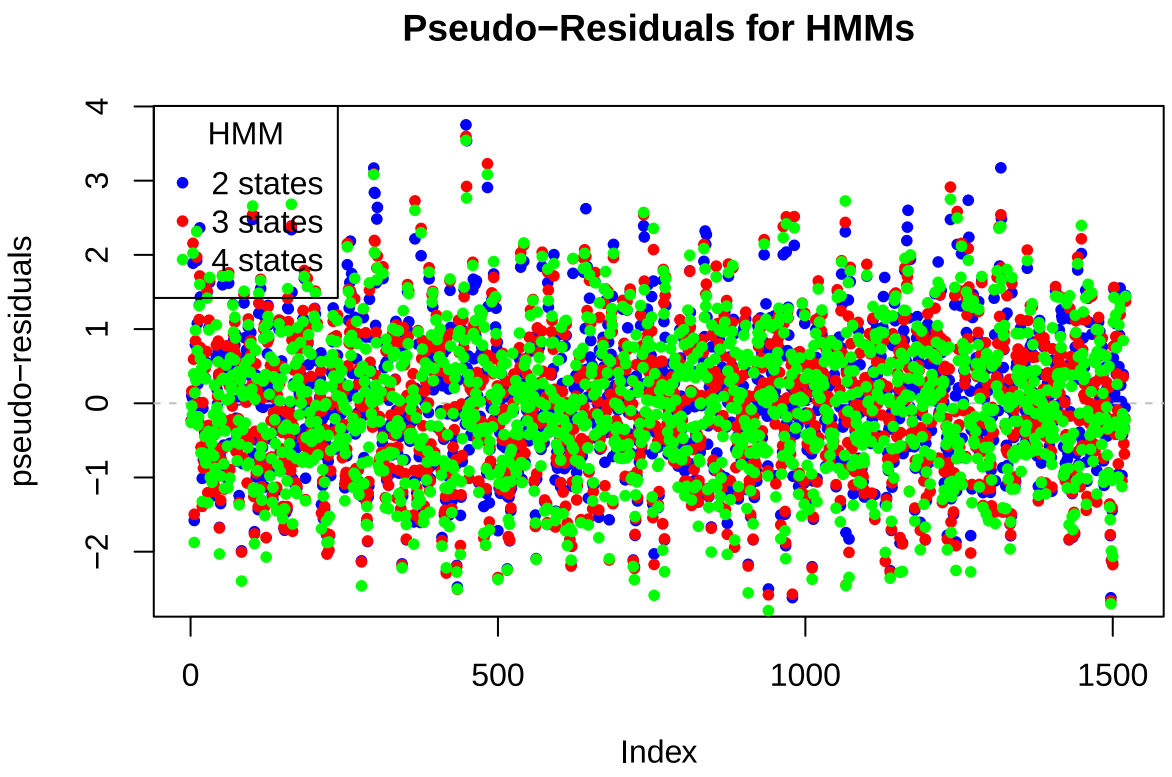
```
mu = as.vector(mod3$mle$stepPar[1,])
sigma = as.vector(mod3$mle$stepPar[2,])
delta = as.vector(mod3$mle$delta)
par(mfrow=c(1,1))
hist(data$step,probability=TRUE,breaks=60,col="light grey",xlab="step length in metres",main="Marginal d
z <- seq(0,1200,by=0.1)
lines(z,
      (delta[1]*dgamma(z,shape=mu[1]^2/sigma[1]^2,scale=sigma[1]^2/mu[1])
      + delta[2]*dgamma(z,shape=mu[2]^2/sigma[2]^2,scale=sigma[2]^2/mu[2])
      + delta[3]*dgamma(z,shape=mu[3]^2/sigma[3]^2,scale=sigma[3]^2/mu[3])),
      col='red',lwd=2)
```



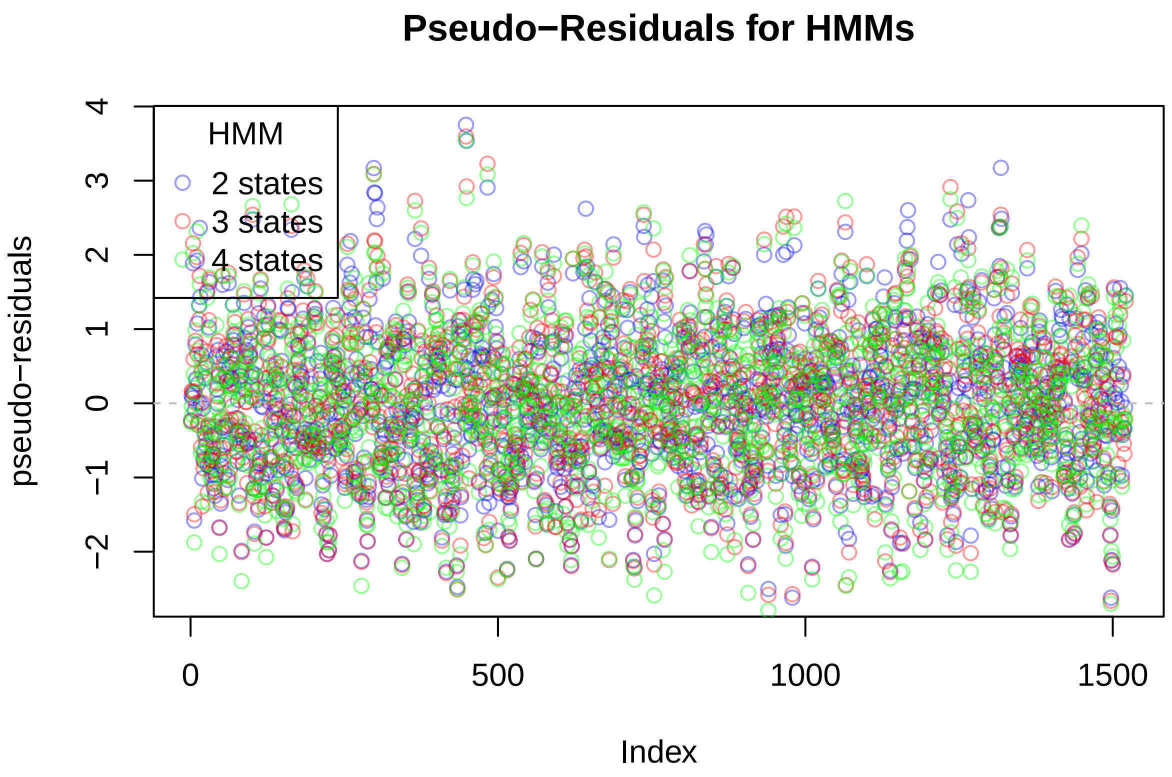
Now we move on to the residual checking for the model fit:

```r
pr2 <- pseudoRes(mod2)
pr3 <- pseudoRes(mod3)
pr4 <- pseudoRes(mod4)
```

```r
# Plot the residuals for the first series (e.g., pr2)
plot(pr2$stepRes, ylab = "pseudo-residuals", main = "Pseudo-Residuals for HMMs",
     col = "blue", pch = 20)

# Add a horizontal line at y=0
abline(h = 0, lty = 2, col = "grey")

# Add the residuals for the second series (pr3) in a different color
points(pr3$stepRes, col = "red", pch = 20)

# Add the residuals for the third series (pr4)
points(pr4$stepRes, col = "green", pch = 20)

# Add a legend to distinguish the series
legend("topleft", legend = c("2 states", "3 states", "4 states"),
       col = c("blue", "red", "green"), pch = 20, title = "HMM")
```



```r
# Define semi-transparent colors
blue_alpha <- rgb(0, 0, 255, 100, maxColorValue = 255)
red_alpha <- rgb(255, 0, 0, 100, maxColorValue = 255)
green_alpha <- rgb(0, 255, 0, 100, maxColorValue = 255)

# Plot the residuals for the first series (pr2) using open circles
plot(pr2$stepRes, ylab = "pseudo-residuals", main = "Pseudo-Residuals for HMMs",
     col = blue_alpha, pch = 1)

# Add a horizontal line at y=0
abline(h = 0, lty = 2, col = "grey")

# Add the residuals for the second series (pr3) with open circles
points(pr3$stepRes, col = red_alpha, pch = 1)

# Add the residuals for the third series (pr4) with open circles
points(pr4$stepRes, col = green_alpha, pch = 1)

# Add a legend with open circles
legend("topleft", legend = c("2 states", "3 states", "4 states"),
       col = c(blue_alpha, red_alpha, green_alpha), pch = 1, title = "HMM")
```



Now, after checking out these things, and that we are content with our choice of the model with 3-states. We move on to state decoding.

But there is a catch with it! There are two kinds of state decoding, namely, global and local. In local state decoding, we are asking our algorithm to point out what is the most likely state at each individual point t. And for global state decoding, we are asking the respective algorithm of ours which path is the most likely one that could occur. So instead of checking our ass at each step, we are gazing into the horizon and trying to point out which road to follow.

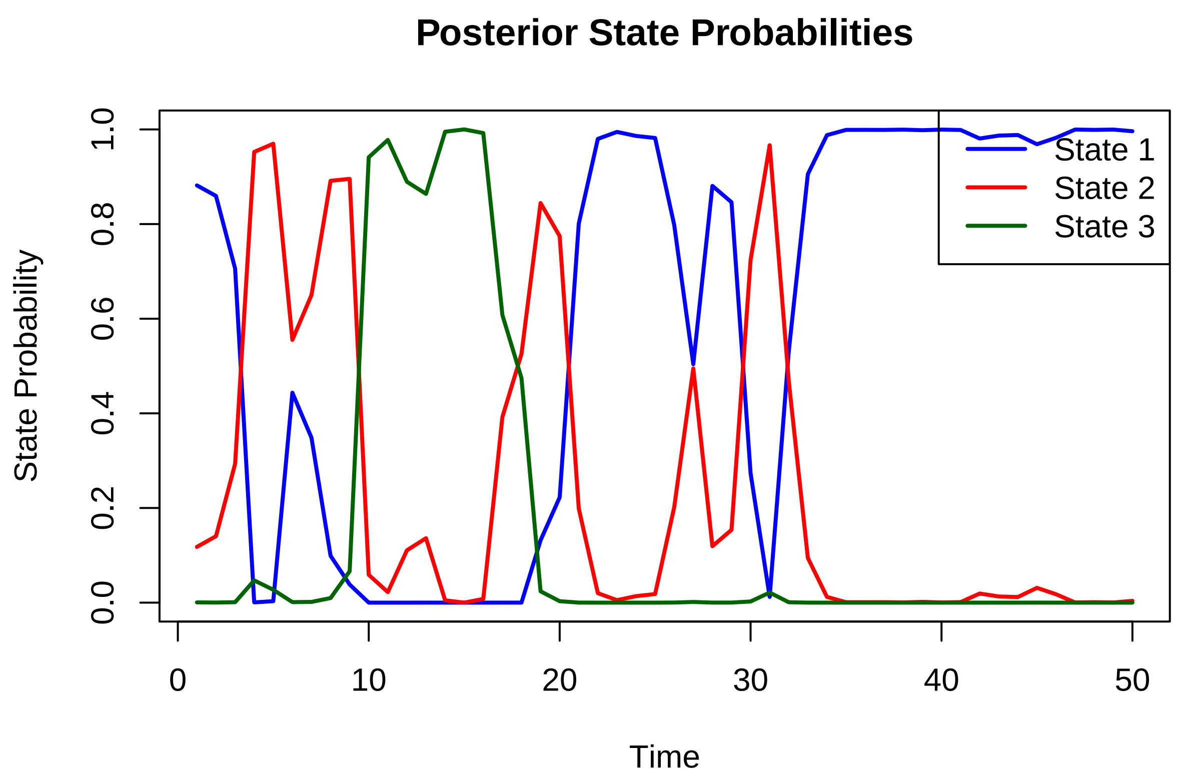
They can be both good and bad, depending on what we expect from them. Local decoding could possible miss the temporal structure and as if 'overfit' to data. But it is quite precise in decoding what is going on at each point, regardless of the neighboring activity. Zig-zagging due to that is inevitable.

Meanwhile, global decoding rather smooths out the state transition process, possibly ignoring local anomalies, so as to be blind to these.

```
sp3 <- stateProbs(mod3)
round(sp3[1:50,],3)
```

```
##        [,1]  [,2]  [,3]
##  [1,] 0.882 0.118 0.001
##  [2,] 0.859 0.140 0.000
##  [3,] 0.706 0.293 0.001
##  [4,] 0.001 0.952 0.047
##  [5,] 0.003 0.970 0.027
##  [6,] 0.444 0.555 0.001
##  [7,] 0.348 0.650 0.002
##  [8,] 0.099 0.891 0.010
##  [9,] 0.038 0.895 0.066
## [10,] 0.000 0.059 0.941
## [11,] 0.000 0.022 0.978
## [12,] 0.000 0.111 0.889
## [13,] 0.000 0.136 0.864
## [14,] 0.000 0.005 0.995
## [15,] 0.000 0.000 1.000
## [16,] 0.000 0.008 0.992
## [17,] 0.000 0.392 0.608
## [18,] 0.000 0.526 0.474
## [19,] 0.132 0.844 0.024
## [20,] 0.222 0.775 0.003
## [21,] 0.801 0.199 0.000
## [22,] 0.980 0.020 0.000
## [23,] 0.995 0.005 0.000
## [24,] 0.986 0.014 0.000
## [25,] 0.982 0.018 0.000
## [26,] 0.798 0.202 0.000
## [27,] 0.504 0.495 0.001
## [28,] 0.881 0.119 0.000
## [29,] 0.846 0.154 0.000
## [30,] 0.273 0.724 0.002
## [31,] 0.012 0.967 0.021
## [32,] 0.532 0.468 0.001
## [33,] 0.905 0.094 0.000
## [34,] 0.988 0.012 0.000
## [35,] 0.999 0.001 0.000
## [36,] 0.999 0.001 0.000
## [37,] 0.999 0.001 0.000
## [38,] 1.000 0.000 0.000
## [39,] 0.998 0.002 0.000
## [40,] 1.000 0.000 0.000
## [41,] 0.999 0.001 0.000
## [42,] 0.981 0.019 0.000
## [43,] 0.987 0.013 0.000
## [44,] 0.988 0.012 0.000
## [45,] 0.969 0.031 0.000
## [46,] 0.982 0.018 0.000
## [47,] 1.000 0.000 0.000
## [48,] 0.999 0.001 0.000
## [49,] 1.000 0.000 0.000
## [50,] 0.996 0.004 0.000
```

```r
matplot(sp3[1:50,], type = "l", lty = 1, lwd = 2,
        col = c("blue", "red", "darkgreen"),
        xlab = "Time", ylab = "State Probability",
        main = "Posterior State Probabilities")
legend("topright", legend = c("State 1", "State 2", "State 3"),
       col = c("blue", "red", "darkgreen"), lty = 1, lwd = 2)
```



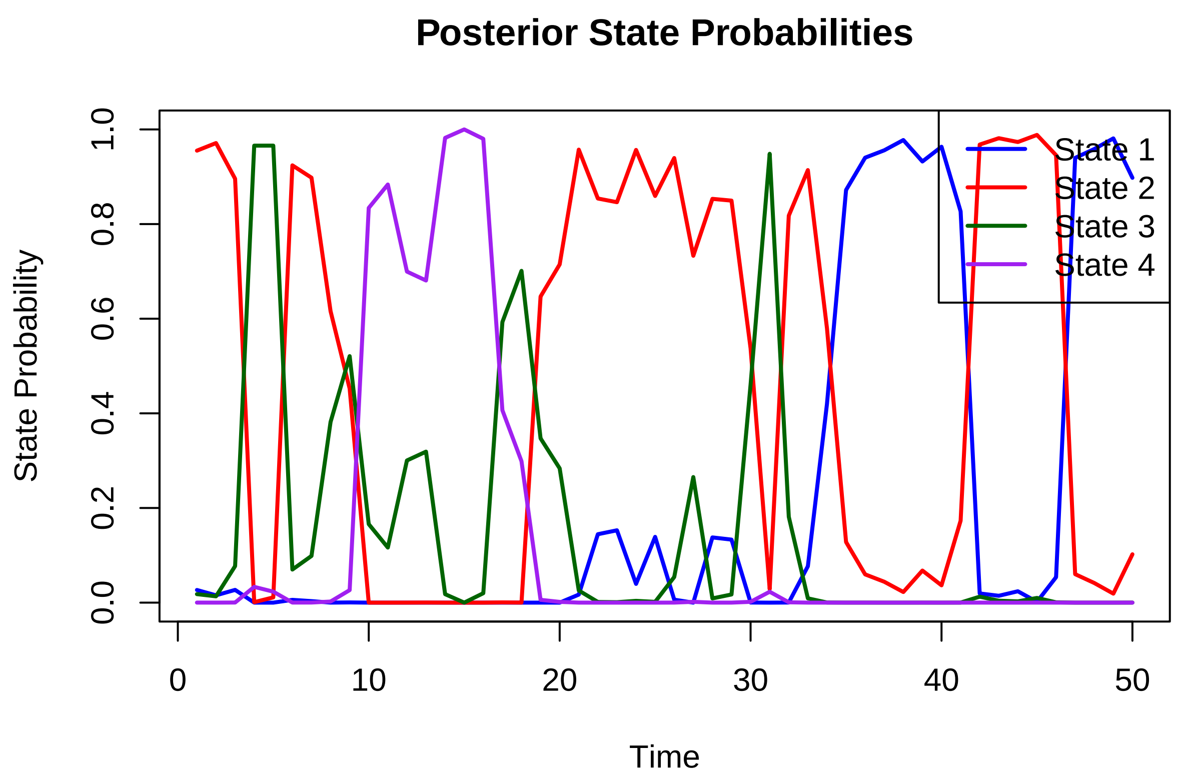
We observe here that the probabilities are mostly gradually rising and falling, making it indeed hard to determine the most probable sequence of states.

I wonder, what difference would we observe for the first 50 values if we check the 4-state model. Let's PLOT!

```
sp4 <- stateProbs(mod4)
round(sp4[1:50,],3)
```

```
##        [,1]  [,2]  [,3]  [,4]
##  [1,] 0.027 0.955 0.018 0.000
##  [2,] 0.015 0.971 0.013 0.000
##  [3,] 0.027 0.896 0.077 0.000
##  [4,] 0.000 0.001 0.966 0.033
##  [5,] 0.000 0.011 0.966 0.023
##  [6,] 0.006 0.924 0.070 0.000
##  [7,] 0.003 0.898 0.099 0.000
##  [8,] 0.000 0.616 0.382 0.002
##  [9,] 0.000 0.452 0.521 0.026
## [10,] 0.000 0.000 0.166 0.834
## [11,] 0.000 0.000 0.116 0.884
## [12,] 0.000 0.000 0.300 0.700
## [13,] 0.000 0.000 0.319 0.681
## [14,] 0.000 0.000 0.018 0.982
## [15,] 0.000 0.000 0.000 1.000
## [16,] 0.000 0.000 0.020 0.980
## [17,] 0.000 0.001 0.593 0.407
## [18,] 0.000 0.000 0.701 0.299
## [19,] 0.000 0.647 0.347 0.006
## [20,] 0.000 0.715 0.284 0.002
## [21,] 0.017 0.957 0.026 0.000
## [22,] 0.145 0.854 0.001 0.000
## [23,] 0.153 0.846 0.001 0.000
## [24,] 0.040 0.957 0.004 0.000
## [25,] 0.139 0.859 0.002 0.000
## [26,] 0.006 0.940 0.054 0.000
## [27,] 0.000 0.733 0.265 0.002
## [28,] 0.138 0.853 0.009 0.000
## [29,] 0.133 0.850 0.017 0.000
## [30,] 0.000 0.537 0.461 0.002
## [31,] 0.000 0.029 0.949 0.023
## [32,] 0.000 0.818 0.181 0.001
## [33,] 0.077 0.914 0.009 0.000
## [34,] 0.420 0.580 0.000 0.000
## [35,] 0.872 0.128 0.000 0.000
## [36,] 0.940 0.060 0.000 0.000
## [37,] 0.956 0.044 0.000 0.000
## [38,] 0.978 0.022 0.000 0.000
## [39,] 0.932 0.068 0.000 0.000
## [40,] 0.963 0.037 0.000 0.000
## [41,] 0.827 0.173 0.000 0.000
## [42,] 0.019 0.968 0.013 0.000
## [43,] 0.014 0.981 0.004 0.000
## [44,] 0.024 0.973 0.003 0.000
## [45,] 0.002 0.988 0.010 0.000
## [46,] 0.054 0.946 0.001 0.000
## [47,] 0.940 0.060 0.000 0.000
## [48,] 0.959 0.041 0.000 0.000
## [49,] 0.981 0.019 0.000 0.000
## [50,] 0.898 0.102 0.000 0.000
```

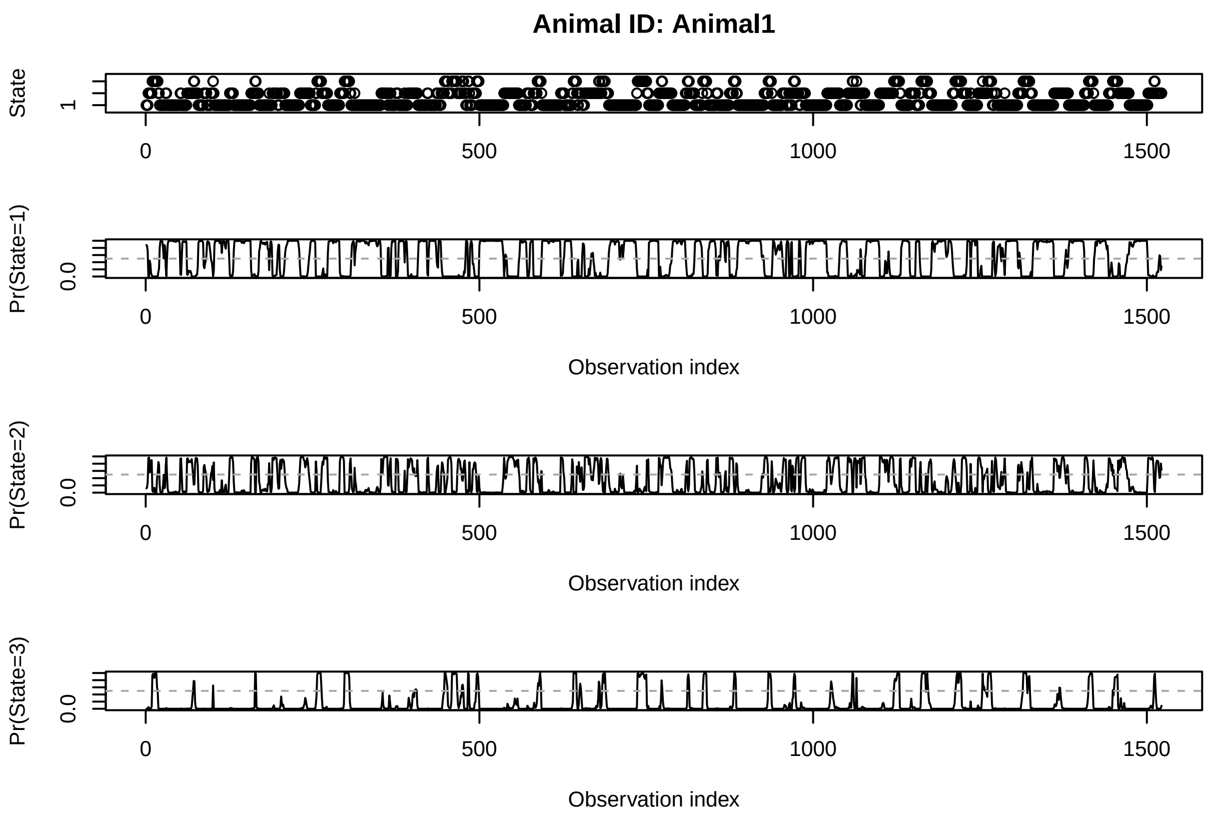
```r
matplot(sp4[1:50,], type = "l", lty = 1, lwd = 2,
        col = c("blue", "red", "darkgreen", "purple"),
        xlab = "Time", ylab = "State Probability",
        main = "Posterior State Probabilities")
legend("topright", legend = c("State 1", "State 2", "State 3", "State 4"),
       col = c("blue", "red", "darkgreen", "purple"), lty = 1, lwd = 2)
```



From the visual examination, the most we can claim is that 4-state setting makes the state probabilities act more wiggly and less apparent. At least, it can be claimed that the state probabilities seem unsure and rather noisy.

```
plotStates(mod3)
```

```
## Decoding states sequence... DONE
## Computing states probabilities... DONE
```


Animal ID: Animal1
State
1
0
500
1000
1500
Pr(State=1)
0.0
Observation index
Pr(State=2)
Pr(State=3)