Supplementary Materials II: Using the `waddle` *R package*

Contents

These supplementary materials illustrate features of the exploratory change point behavioral analysis described in the manuscript text. They also serve to illustrate the use of the `waddle`, along with the `bcpa`, `mrw` and `adehabitatLT` packages on which `waddle` depends. We first present the code used to simulate the multistate random walks, and then the various analysis tools.

```
library(waddle)
```

A Simulating multi-state tracks

We simulated two kinds of multi-state tracks: one based on the an integrated Ornstein-Uhlenbeck velocity process (the CVM) and one based on a biased correlated random walk (BCRW). Functions for generating these tracks are also provided in the `waddle` package.

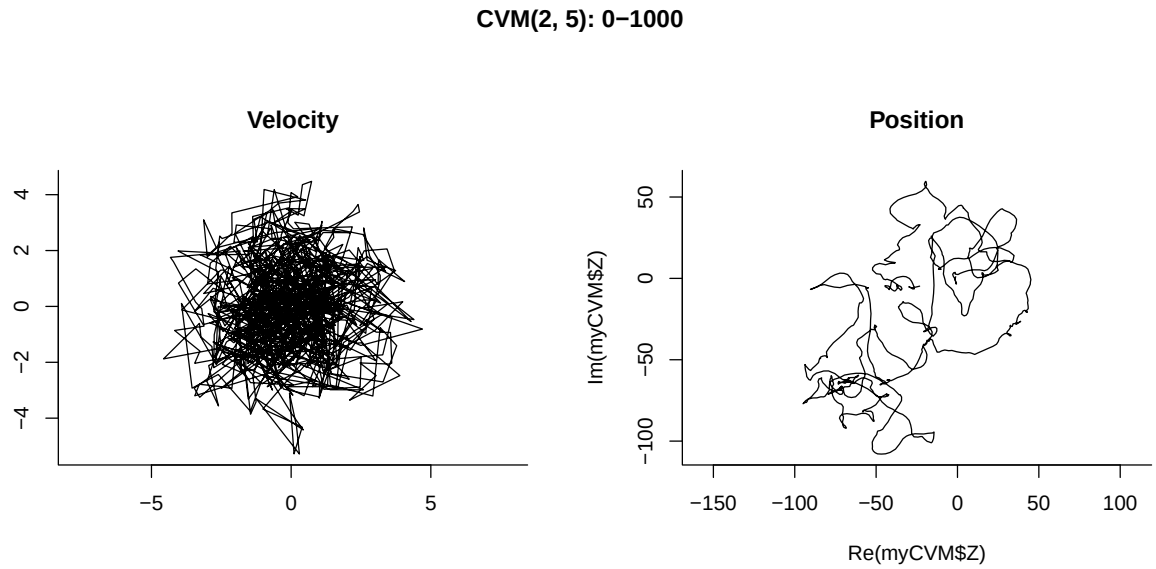
A.1 Correlated velocity movement

The `CVM` function simulate a correlated velocity movement process at (arbitrary) times T, with mean speed ν and time scale τ

```
T <- 1:1000
myCVM <- CVM(T, nu=2, tau=5)
```

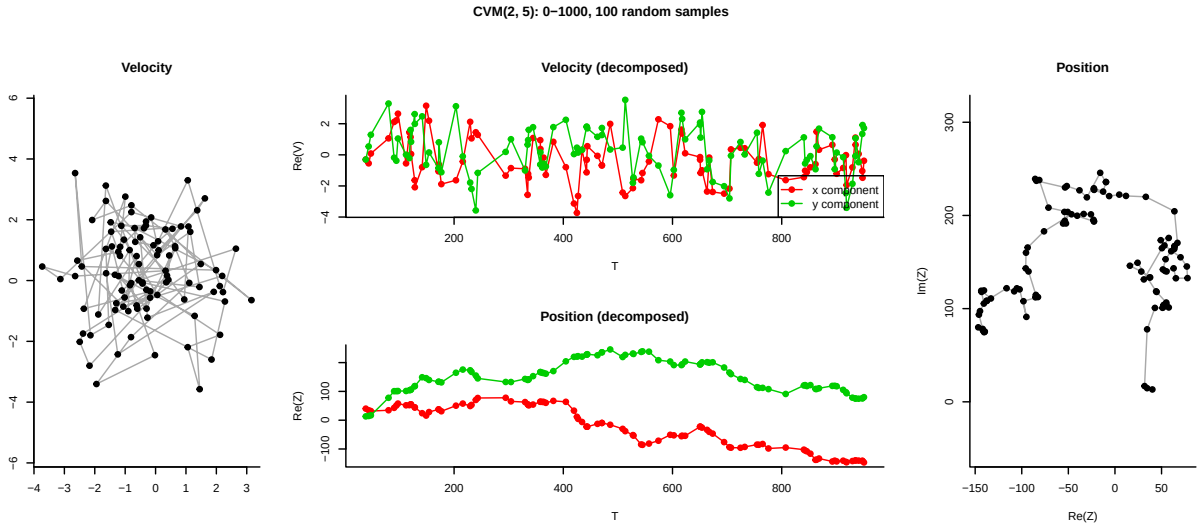
This object contains V and Z, which are complex velocity and position coordinates, respectively. We illustrate the output of a simulation below:

```
plot(myCVM$V, type="l", asp=1, main="Velocity", xlab="", ylab="")
plot(myCVM$Z, type="l", main="Position", asp=1)
title("CVM(2, 5): 0-1000", outer=TRUE, cex=1.5)
```



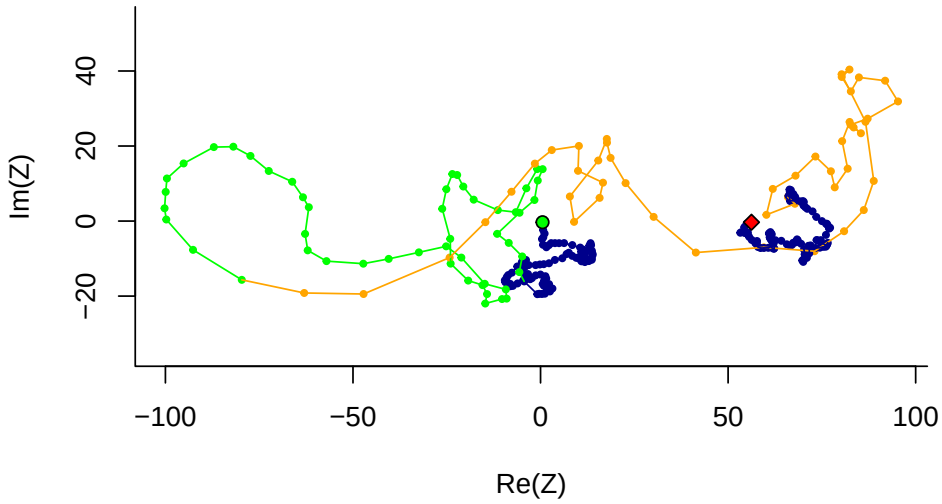
Importantly, this process can be sampled from at arbitrary time intervals (note the relatively sparse random sampling in the figure):

```
T <- cumsum(rexp(100,1/10))
myCVM2 <- CVM(T, nu=2, tau=5)
```



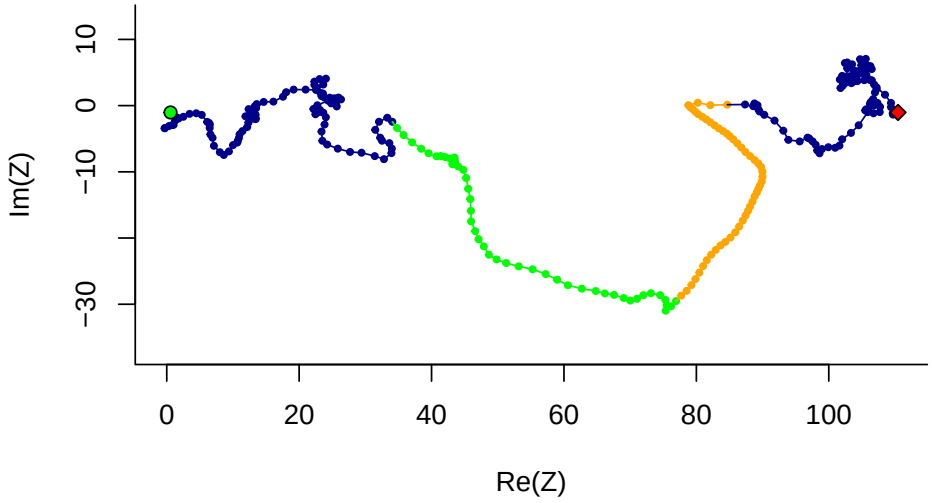
The multistate CRWs analyzed in the manuscript were generated using the `multiCVM` function, which also has a plotting method. Thus, the velocity change simulation was generated as follows:

```
nus <- c(1,5,10,1)
taus <- rep(2,4)
Ts <- c(100,50,50,100)
Nu.sim <- multiCVM(taus, nus, Ts)
# rotating to a primarily horizontal axis
Nu.sim$Z <- Nu.sim$Z * complex(mod = 1, arg = -Arg(tail(Nu.sim$Z,1) - Nu.sim$Z[1]))
plot(Nu.sim, col = c("darkblue", "green", "orange", "darkblue"))
```



and the time scale shift simulation:

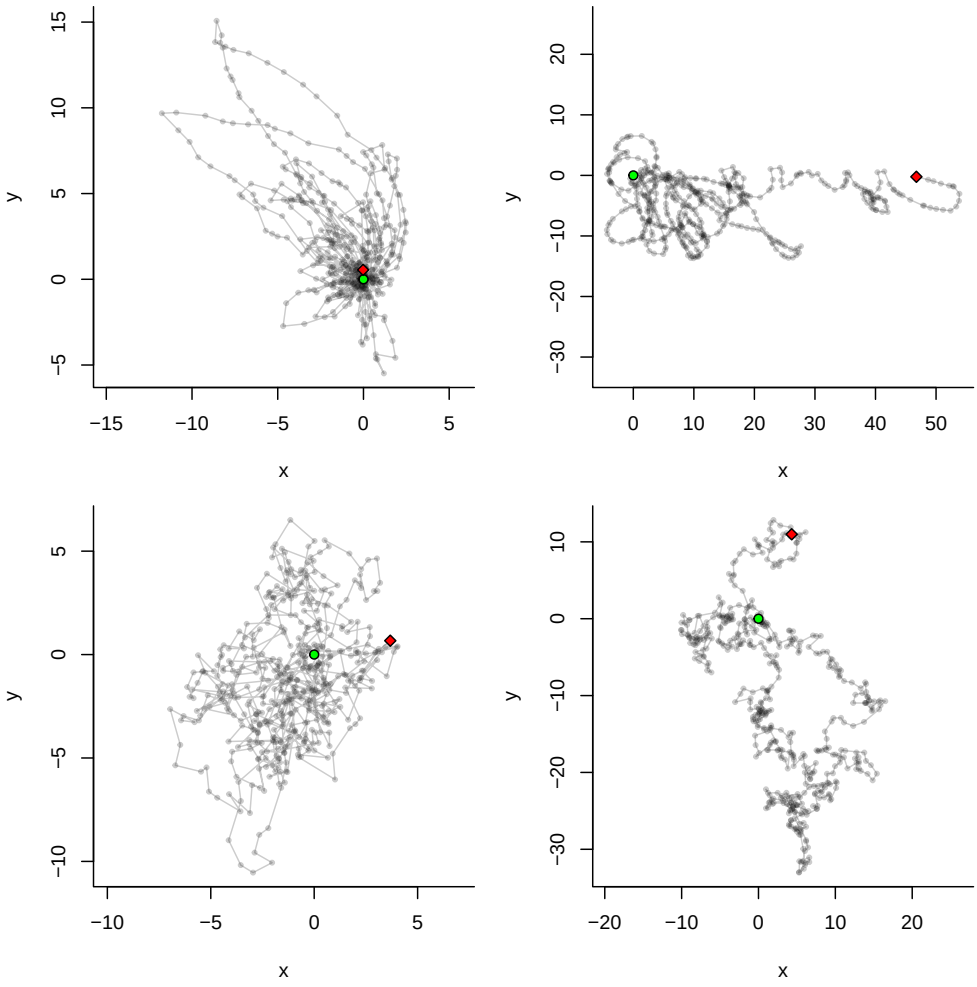
```
taus <- c(2,10,100,2)
nus <- rep(1,4)
Ts <- c(100,50,50,100)
Tau.sim <- multiCVM(taus, nus, Ts)
plot(Tau.sim, col = c("darkblue", "green", "orange", "darkblue"))
```



A.2 Biased correlated random walk

The `BCRW` function simulated the process described in the Methods section of the text, with two parameters: the angle concentration parameters ρ and the strength of attraction A. Below, we illustrate tracks that are highly correlated (upper panels) and less correlated (lower panels), and more attractive (left panels) and less strongly attractive (right panels).

```
plot(BCRW(n = 500, rho=0.9, attraction = 0.9))
plot(BCRW(n = 500, rho=0.9, attraction = 0.2))
plot(BCRW(n = 500, rho=0.2, attraction = 0.9))
plot(BCRW(n = 500, rho=0.2, attraction = 0.2))
```



A multistate BCRW with k phases is generated with the `multiBCRW` function, which takes vectors of length k of the relevant parameters, including attraction points. The simulation in the text was generated using the following parameter values:

```
Ts <- c(100,50,50,100)
Z.centers <- c(0,25-10i,50+10i, 50+10i)
attractions <- c(0.5,0.9,0.9,0.5)
rhos <- rep(0.5, 4)
BCRW.sim <- multiBCRW(rhos, attractions, Z.centers, Ts)
```

The output of this function is a list with the complex coordinates, a vector of the phases, and the duration of the phases:

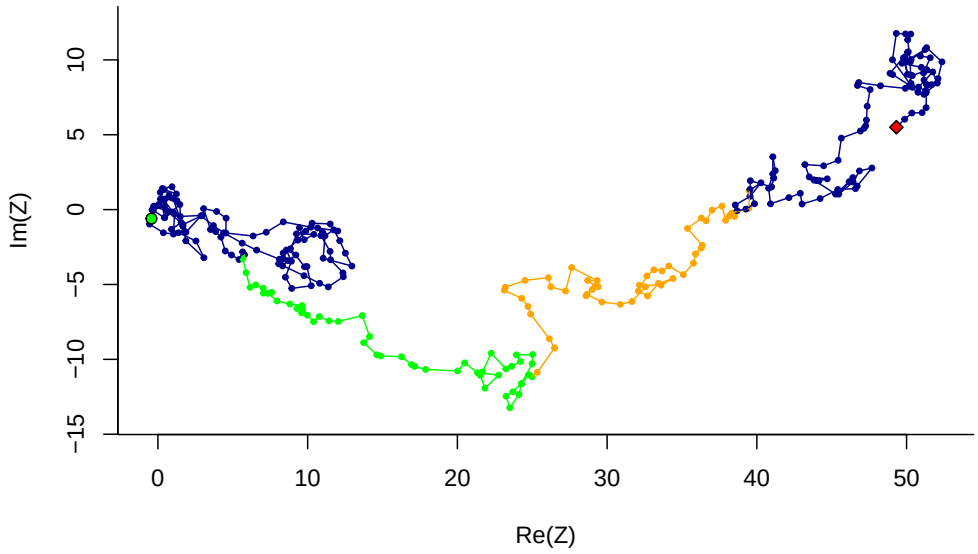
```
str(BCRW.sim)


## List of 3
```

```
##  $ Z    : cplx [1:296] -0.431-0.602i -0.556-0.972i 0.368-1.546i ...
##  $ Phase: int [1:300] 1 1 1 1 1 1 1 1 1 1 ...
##  $ ns   : num [1:4] 100 50 50 100
##  - attr(*, "class")= chr "multipath"
```

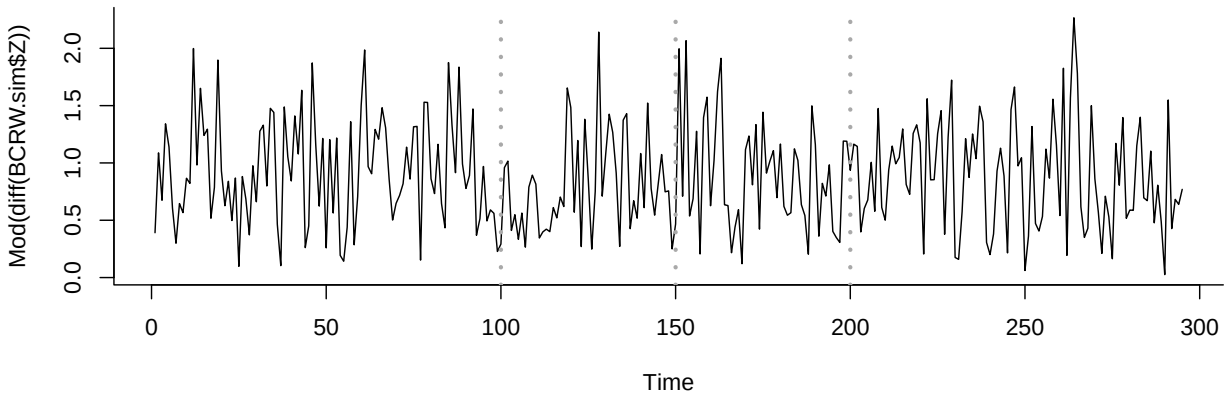
There is a plotting method for the multistate BCRW object:

```
plot(BCRW.sim, col = c("darkblue", "green", "orange", "darkblue"))
```



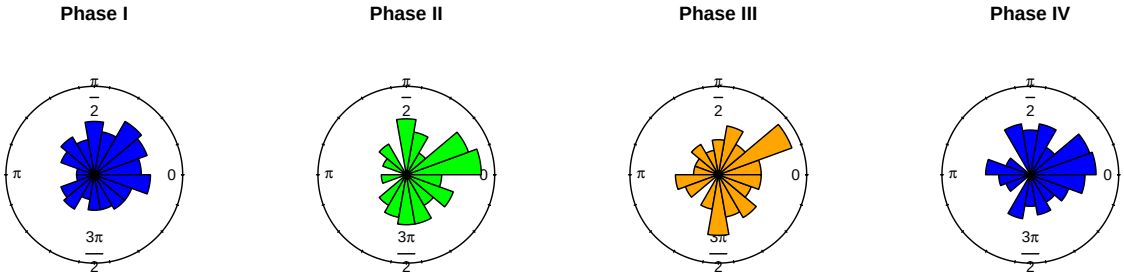
Note that in this example, the step-length does not change over the duration of the track:

```
plot.ts(Mod(diff(BCRW.sim$Z)))
abline(v = c(100,150,200), lwd=3, col="darkgrey", lty=3)
```



And the clustering is similarly low across all four phases:

```
# step vectors
S <- diff(BCRW.sim$Z)
# absolute orientation
Phi <- Arg(diff(BCRW.sim$Z))
# turning angles
Theta <- diff(Phi)
# rose diagrams
require(circular)
rose.diag(Theta[1:99], bins=18, main="Phase I", prop=2, col="blue")
rose.diag(Theta[100:149], bins=18, main="Phase II", prop=2, col="green")
rose.diag(Theta[150:200], bins=18, main="Phase III", prop=2, col="orange")
rose.diag(Theta[201:350], bins=18, main="Phase IV",  prop=2, col="blue")
```



B Change point analyses

B.1 First-Passage Time

We explore the FPT results for the lamprey data set at three different radii, without and with smoothing.

Loading the package and the data:

```
require(waddle)
data(Lamprey)
```

Creating a smoothed version of the lamprey data:

```
L1 <- Lamprey
L2 <- SmoothTrack(Lamprey, 3)
```

Converting to a `traj` class, the native class of trajectories in `adehabitatLT`:

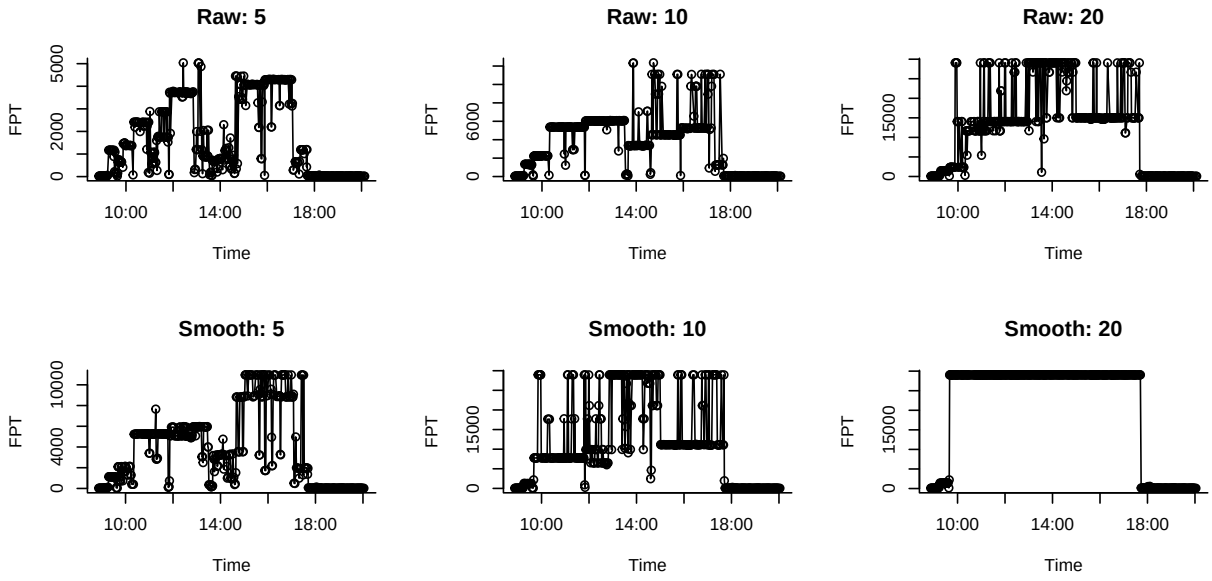
```
L1.traj <- as.ltraj(data.frame(L1$X, L1$Y),L1$Time, id = "Lamprey")
L2.traj <- as.ltraj(data.frame(L2$X, L2$Y),L2$Time, id = "Lamprey")
```

Calculating the fpt for three different radii:

```
radii <- c(5,10,20)
L1.fpt <- fpt(L1.traj, radii)
L2.fpt <- fpt(L2.traj, radii)
```

Plotting the FPT:

```
plot.fpt(L1.fpt, radii, xlab="Time", main="Raw: ")
plot.fpt(L2.fpt, radii, xlab="Time", main="Smooth: ")
```



The smoothed 20 m FPT provides the clearest behavioral separation.

B.2 BPMM: running and diagnostics

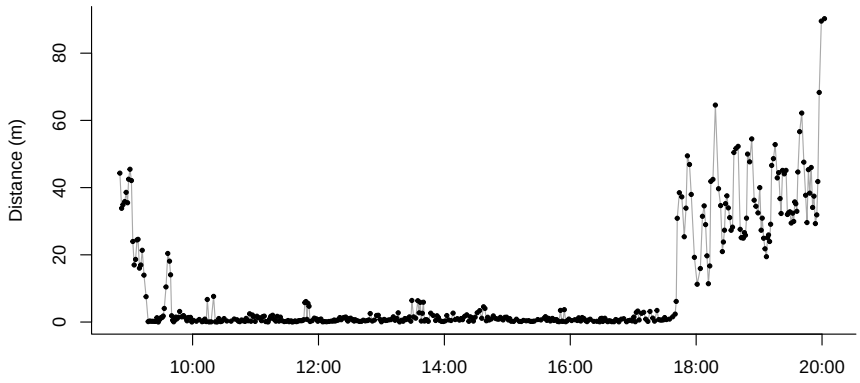
The segmentation is performed using tools in `adehabitatLT` with a few convenience wrappers in `waddle`. Note that the `waddle` implementation is highly simplified, e.g. applicable only to step lengths or their log transforms. For a more flexible implementation, the reader is encouraged to study the `modpartltraj` help file in `adehabitatLT`.

To perform the segmentation, we must first regularize the trajectory:

```
L2.reg <- InterpolatePoints(L2, 1, "min")$Data
L2.traj <- as.ltraj(data.frame(L2$X, L2$Y),L2$Time, id = "Lamprey")
```

We plot the distance steps:

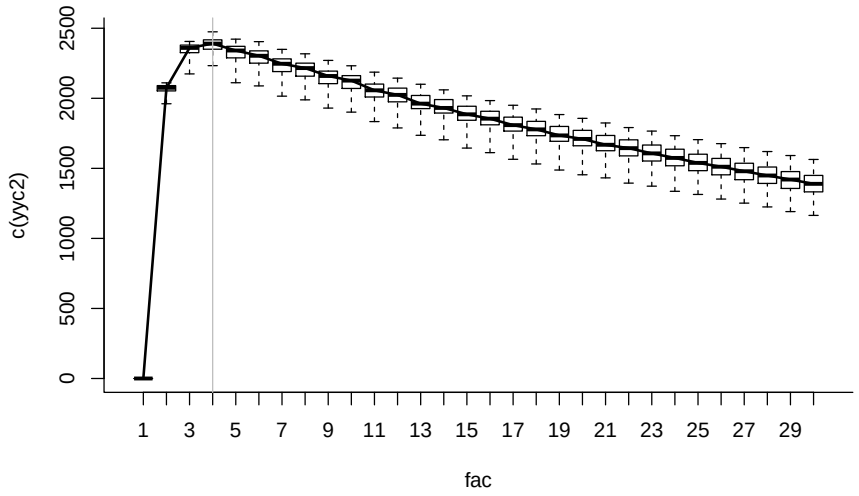
```
L.VT <- GetVT(L2)
plot(L2.traj[[1]]$date, L2.traj[[1]]$dist, xlab="", ylab="Distance (m)", type="l", col="darkgrey")
points(L2.traj[[1]]$date, L2.traj[[1]]$dist, pch=19, cex=0.5)
```



A visual inspection suggests that a standard deviation of 5 m might be appropriate for the variance within the homogeneous sections, (*here, we are not too concerned about the difference in variance - in other analyses, we examine this more rigorously and end up using the log transformation.*).

After the data prepping, the first step is an assessment of the number of partitions that might be present in these data. In the convenience function `PrepSegments`, the `nmodels` argument sets the number of candidate models, i.e. the μ_j's ranging from 0 to the maximum value of the step length.

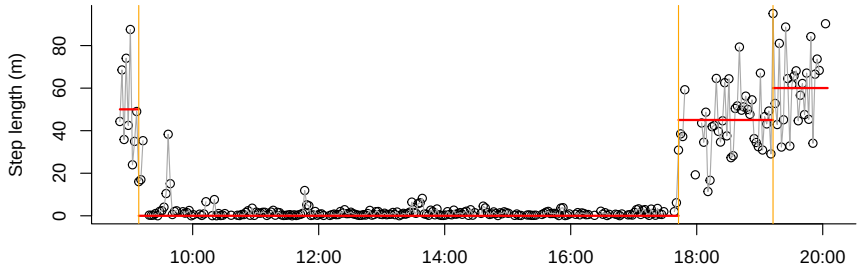
```
L2.segments <- Prep.segments(L2.traj, units="min", dt = 2, sd=5, nmodels=20)
```



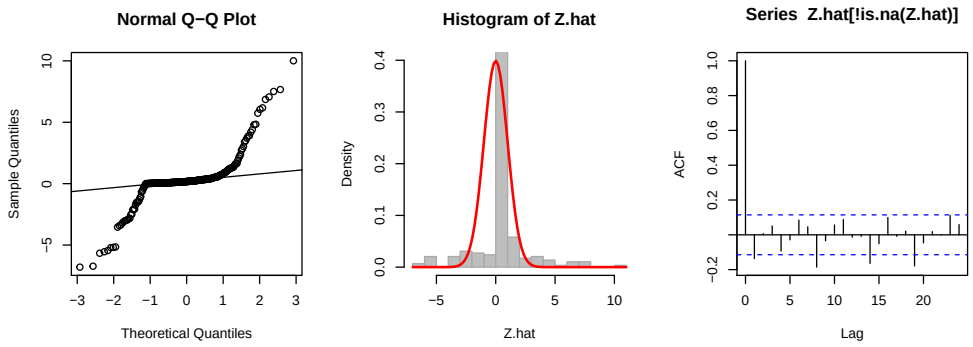

```
## Maximum likelihood for K =  4
```

The likelihood assessment suggests 4 partitions - one more than the 3 partitions that we might have expected (note, that because of the randomization, the number can vary - some runs suggest 3 partitions). We now fit the four partition model, and plot the segments:

```
L2.partition <- Partition.segments(L2.segments)
plot.segments(L2.partition, xlab="", ylab="Step length (m)")
```



The diagnostic plot of the Lamprey residuals:



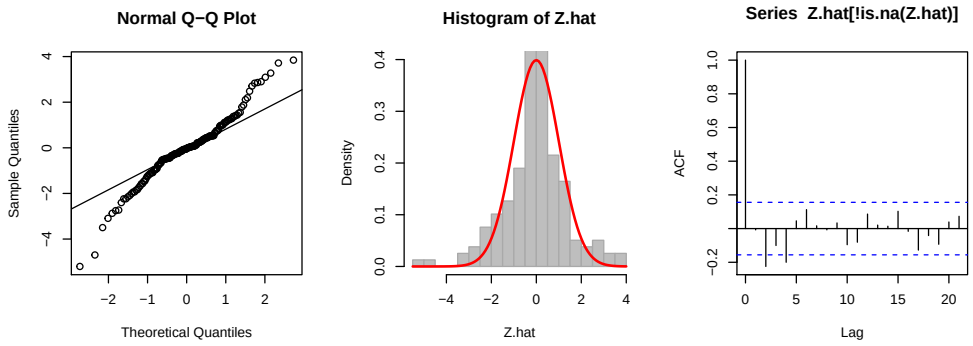
Clearly, the normality assumption is violated (left panel). The central panel is useful for "tuning" the standard deviation.

The complete analysis and diagnostic code for the wolf:

```
data(Wolf)
# calculate daily mean
Wolf2 <- data.frame(X = tapply(Wolf$X, substr(Wolf$Time, 1, 10), mean),
                    Y = tapply(Wolf$Y, substr(Wolf$Time, 1, 10), mean))
Wolf2$Time <- as.POSIXct(row.names(Wolf2))
# convert to traj
Wolf.traj <- as.ltraj(data.frame(Wolf2$X, Wolf2$Y),as.POSIXct(Wolf2$Time), id = "Wolf")
# prep segments
Wolf.prep <- Prep.segments(Wolf.traj, units="day", dt = 2, sd=1, Km=40, log=TRUE, plotit=FALSE)


## Maximum likelihood for K =  27


# fit segments
Wolf.segments <- Partition.segments(Wolf.prep)
# plot diagnostics
DiagPlot.segments(Wolf.segments)
```



The diagnostic plots suggest that the normality and independence assumptions and the choice for standard

deviation are reasonably justified for the log transform of step length, though with some left skew due to an over-correction in the log transform.

B.3 BCPA: running and diagnostics

The `bcpa` package similarly streamlines the implementation of the BCPA to movement data. The `GetVT()` function obtains step lengths, absolute orientations, turning angles:

```
Lamprey.VT <- GetVT(L2, units = "min")
```

Set the required "knobs":

```
windowsize <- 30
windowstep <- 1
K <- 0.5
```

Perform the analysis. Note that any function of the columns of the VT data can be passed in the second argument (e.g. `log(v)`:

```
Lamprey.ws <- WindowSweep(Lamprey.VT, "V*cos(Theta)", windowsize, windowstep,
                          plotme = FALSE, K = K, tau=TRUE, progress=FALSE)
```

A summary of the "flat' results is given by:

```
ChangePointSummary(Lamprey.ws, clusterwidth=3)
```

```
## $breaks
##   X1    middle size modelmode        middle.POSIX
## 1  1  50.16667    9         4 2010-05-02 09:41:00
## 2 15 175.39286   14         2 2010-05-02 11:46:00
## 3 16 182.73333   10         2 2010-05-02 11:54:00
## 4 27 291.06667    5         2 2010-05-02 13:42:05
## 5 31 371.66667    4         4 2010-05-02 15:02:20
## 6 48 528.30000   15         4 2010-05-02 17:39:40
##
## $phases
##         t.cut     mu.hat      s.hat     rho.hat         t0        t1
## 1 (1.33,50.2] 11.2781251 11.6950908 13.31435514   1.333333  50.16667
## 2  (50.2,175]  0.1005441  0.6124534  0.01894909  50.166667 175.39286
## 3   (175,183]  0.4616606  4.0329051  0.02748634 175.392857 182.73333
## 4   (183,291]  0.2442350  1.0237796  0.02836101 182.733333 291.06667
## 5   (291,372]  0.4689255  0.8026789  0.35100299 291.066667 371.66667
## 6   (372,528]  0.2315370  0.5701728  0.08343703 371.666667 528.30000
## 7   (528,673] 23.2158677  8.1277220  5.26228647 528.300000 673.16667
##     interval
```

```
## 1  48.833333
## 2 125.226190
## 3   7.340476
## 4 108.333333
## 5  80.600000
## 6 156.633333
## 7 144.866667
```

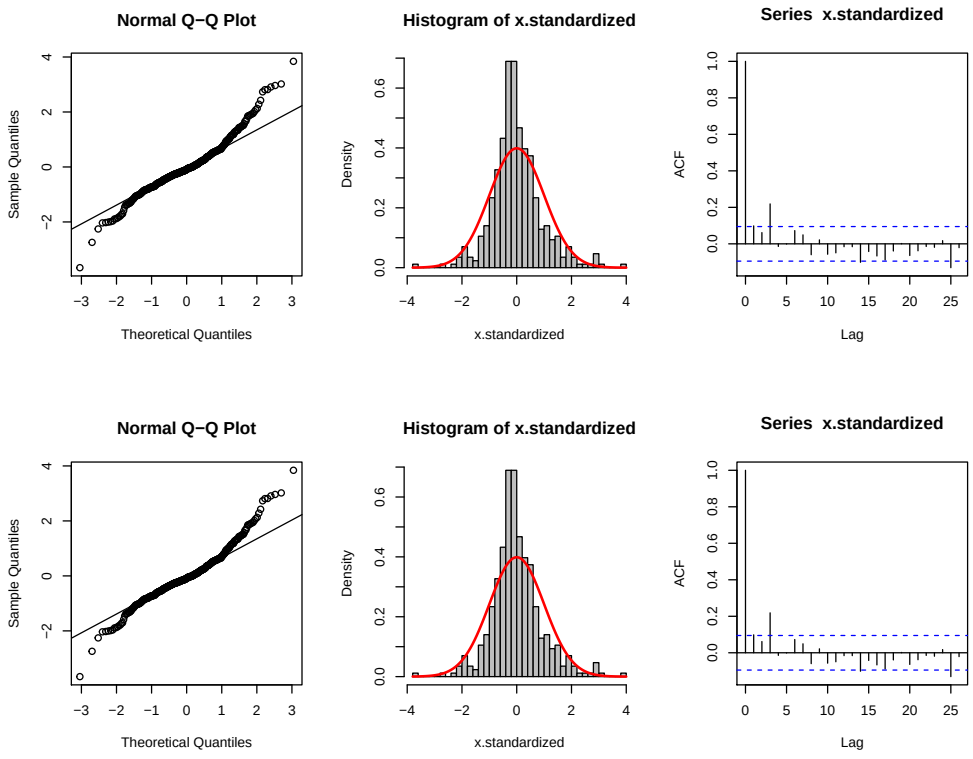
Note that model 2 in the upper table refers to only the variance changing, while model 4 refers to both means and variances changing.

The following code will produce the plots in figure 3 c and d:

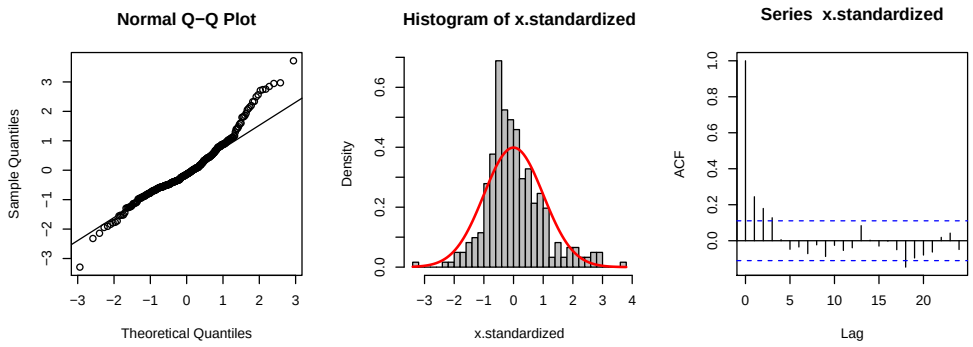
```
plot(Lamprey.ws, type="smooth", threshold = 2, legend=FALSE)
plot(Lamprey.ws, type="flat", clusterwidth = 3, legend=FALSE)
```

And the residual diagnostic plot is given by:

```
DiagPlot(Lamprey.ws, "smooth")
```



The assumptions appear to be largely satisfied. Diagnostic plots for the wolf data are below.



B.4 Multi-state random walk: fitting models and analysis

In this appendix, we first present the JAGS code for the three fitted models (double state, double switching, and triple switching), then illustrate the R code from the `mrw` package to fit the models and analyze the results.

JAGS code

Two-state model

```
data{
  for (t in 1:numT){
    ones[t] <- 1
  }
}
model{
  for (t in 1:numT){
    # likelihood for steps
    ## Weibull distriution for step length
    step[t] ~ dweib(v[t], lambda[t])
    ## shape parameter
    v[t] <- a[idx[t]]
    ## scale parameter (transform between WinBUGS and R's definition of Weibull)
    lambda[t] <- pow(b[idx[t]], -a[idx[t]])
    # likelihood for turns.
    ones[t] ~ dbern(wc[t])
    # Density function for Wrapped Cauchy distribution
    wc[t] <- ( 1/(2*Pi) * (1 - rho.t[t] * rho.t[t]) /
              (1 + rho.t[t] * rho.t[t] - 2*rho.t[t] * cos(theta[t] - mu.t[t])) ) / 300
    ## mean cosine for the circular distribution
    rho.t[t] <- rho[idx[t]]
    ## mean direction for turns
    mu.t[t] <- mu[idx[t]]
    # idx is the latent variable and the parameter index
    ## priors on p[t,1], the probability that the t-th
    idx[t] ~ dcat(p[t,])
    # observation corresponds to movement state 1.
    p[t,1] ~ dunif(0,1)
    p[t,2] <- 1 - p[t,1]
  }
```

```
  ####### priors on movement shape
  a[1] ~ dgamma(0.01, 0.01)
  a[2] ~ dgamma(0.01, 0.01)
  ####### priors on movement scale
  b[1] ~ dgamma(0.01, 0.01)
  b[2] ~ dgamma(0.01, 0.01)
  ###### priors for mean direction of turns
  mu[1] ~ dunif(-3*Pi/2, Pi/2)
  mu[2] ~ dunif(-3*Pi/2, Pi/2)
  ###### priors for mean cosine of circular distribution
  rho[1] ~ dunif(0, 1)
  rho[2] ~ dunif(0, 1)
  Pi <- 3.14159265359     # define pi
}
```

Two-state switching model

```
data{
  for (t in 1:numT){
    ones[t] <- 1
  }
}
model{
  for (t in 2:numT){
    # likelihood for steps
    ## Weibull distriution for step length
    step[t] ~ dweib(v[t], lambda[t])
    ## shape parameter
    v[t] <- a[idx[t]]
    ## scale parameter (transform between WinBUGS and R's definition of Weibull)
    lambda[t] <- pow(b[idx[t]], -a[idx[t]])
    # likelihood for turns.
    ones[t] ~ dbern(wc[t])
    ## Density function for Wrapped Cauchy distribution
    wc[t] <- ( 1/(2*Pi) * (1 - rho.t[t] * rho.t[t]) / (1 + rho.t[t] * rho.t[t] - 2*rho.t[t] * cos(theta[t] - mu.t[t])) ) / 300
    ## mean cosine for the circular distribution
    rho.t[t] <- rho[idx[t]]
    ## mean direction for turns
    mu.t[t] <- mu[idx[t]]
    # idx is the latent variable and the parameter index
    idx[t] ~ dcat(prob[t,])
    # prob[t,1] is the probability that the t-th observation corresponds to movement state 1
    prob[t,1] <- p[idx[t-1]]
    prob[t,2] <- 1 - p[idx[t-1]]
  }

  ####### priors on movement shape
  a[1] ~ dgamma(0.01, 0.01)
  a[2] ~ dgamma(0.01, 0.01)
  ####### priors on movement scale
  b[1] ~ dgamma(0.01, 0.01)
  b[2] ~ dgamma(0.01, 0.01)
  ###### priors for mean direction of turns
  mu[1] ~ dunif(-3*Pi/2, Pi/2)
  mu[2] ~ dunif(-3*Pi/2, Pi/2)
  ###### priors for mean cosine of circular distribution
  rho[1] ~ dunif(0, 1)
  rho[2] ~ dunif(0, 1)
  #### priors for transition probabilities
  p[1] ~ dunif(0,1)
  p[2] ~ dunif(0,1)
```

```
  ### asign state for first observation
  phi[1] ~ dunif(0, 1)
  phi[2] <- 1 - phi[1]
  idx[1] ~ dcat(phi[])


  Pi <- 3.14159265359     # define pi
}
```

Three-state switching model

```
data{
  for (t in 1:numT){
    ones[t] <- 1
  }
}
model{
  for (t in 2:numT){
 # likelihood for steps
    ## Weibull distriution for step length
step[t] ~ dweib(v[t], lambda[t])
    ## shape parameter
v[t] <- a[idx[t]]
    ## scale parameter (transform between WinBUGS and R's definition of Weibull)
lambda[t] <- pow(b[idx[t]], -a[idx[t]])
    # likelihood for turns.
ones[t] ~ dbern(wc[t])
 ## Density function for Wrapped Cauchy distribution
wc[t] <- ( 1/(2*Pi) * (1 - rho.t[t] * rho.t[t]) / (1 + rho.t[t] * rho.t[t] - 2*rho.t[t] * cos(theta[t] - mu.t[t])) ) / 300
rho.t[t] <- rho[idx[t]]     # mean cosine for the circular distribution
mu.t[t] <- mu[idx[t]]       # mean direction for turns
    # idx is the latent variable and the parameter index
idx[t] ~ dcat(prob[t,])
    # prob[t,i] is the probability that the t-th observation corresponds to movement state i
prob[t,1] <- p[idx[t-1]]
prob[t,2] <- (1 - p[idx[t-1]]) * phi[idx[t-1]]
prob[t,3] <- (1 - p[idx[t-1]]) * (1 - phi[idx[t-1]])
}
 ####### priors on movement shape
a[1] ~ dgamma(0.01, 0.01)
a[2] ~ dgamma(0.01, 0.01)
a[3] ~ dgamma(0.01, 0.01)
 ####### priors on movement scale
b[1] ~ dgamma(0.01, 0.01)
b[2] ~ dgamma(0.01, 0.01)
b[3] ~ dgamma(0.01, 0.01)
 ###### priors for mean direction of turns
mu[1] ~ dunif(-3*Pi/2, Pi/2)
mu[2] ~ dunif(-3*Pi/2, Pi/2)
mu[3] ~ dunif(-3*Pi/2, Pi/2)
 ###### priors for mean cosine of circular distribution
rho[1] ~ dunif(0, 1)
rho[2] ~ dunif(0, 1)
rho[3] ~ dunif(0, 1)
 #### priors for transition probabilities
p[1] ~ dunif(0,1)
p[2] ~ dunif(0,1)
p[3] ~ dunif(0,1)
phi[1] ~ dunif(0, 1)
phi[2] ~ dunif(0, 1)
phi[3] ~ dunif(0, 1)
```

```
 ### asign state for first observation
idx[1] ~ dcat(phi[])
Pi <- 3.14159265359     # define p
}
```

Fitting the multi-state random walk

The following steps fit the model in R. Note that fitting this model with the JAGS code above can take a long time (several hours or more) depending on the size of the dataset.

To fit a correlated random walk (or a mixture thereof) the steps must equally spaced time intervals, so we interpolate the Lamprey data to meet these requirements. Interpolating every 120 sec. (most observations are separated by 1 or 2 minutes, though the longest gap is 7 minutes) gives 338 data points (slightly less than the smoothed data used in the other analysis).

Loading and prepping the data:

```
require(waddle)
require(rjags)
require(R2jags)
data(Lamprey)
Lamprey.Data <- InterpolatePoints(Lamprey, n = 120, id = "Lamprey")$Data
Lamprey.VT <- GetVT(Lamprey.Data, units = "day")

# small offset to avoid zero sized length steps
Lamprey.VT$S[Lamprey.VT$S == 0] <- 0.000001
Lamprey.MRW <- list("numT" = nrow(Lamprey.VT), "step" = Lamprey.VT$S, "theta" = Lamprey.VT$Theta)
```

Two-state model

The first model assigns each step to one of two CRW models.

```
initsDoubleState = function() { list(a = sort(runif(2, 0.5, 2.5)),
                                     b = sort(runif(2, 0, 5)),
                                     mu = runif(2, -3*pi/2, pi/2),
                                     rho = sort(runif(2, 0, 1)) ) }
parametersDoubleState = c("a", "b", "mu", "rho", "idx")
system.time(Lamprey.doubleState <- jags(Lamprey.MRW, initsDoubleState, parametersDoubleState,
                                        "DoubleState.txt",
                                        n.chains = 2, n.iter = 750000, n.thin = 1000, n.burnin = 50000))
analyzeConvergence(Lamprey.doubleState, plotTraces = TRUE)
```

Two-state switching model

In the second model, switching between the two states is governed by a Markovian transition matrix.

```
initsDoubleStateSwitch <- function() {list (a = sort(runif(2, 0.5, 2.5)),




parametersDoubleStateSwitch <- c("a", "b", "mu", "rho", "p", "idx")
Lamprey.doubleStateSwitch <- jags(Lamprey.MRW, initsDoubleStateSwitch, parametersDoubleStateSwitch,
                                  "DoubleStateSwitch.txt",
                                  n.chains = 2, n.iter = 100000, n.thin = 50, n.burnin = 10000)
analyzeConvergence(Lamprey.doubleStateSwitch, plotTraces = TRUE)
```

Three-state switching model

Three CRW states, and a 3×3 transition matrix.

```
initsTripleStateSwitch <- function() {list (a = sort(runif(3, 0.5, 2.5)),




parametersTripleStateSwitch <- c("a", "b", "mu", "rho", "p", "phi", "idx")
Lamprey.tripleStateSwitch <- jags(Lamprey.MRW, initsTripleStateSwitch, parametersTripleStateSwitch,
                                  "TripleStateSwitch.txt",
                                  n.chains = 2, n.iter = 200000, n.thin = 100, n.burnin = 40000)
analyzeConvergence(Lamprey.tripleStateSwitch, plotTraces = TRUE)
```

Analyzing model results

Here, we walk through the steps of analyzing and comparing the multi-state random walk fits.

Two-state model

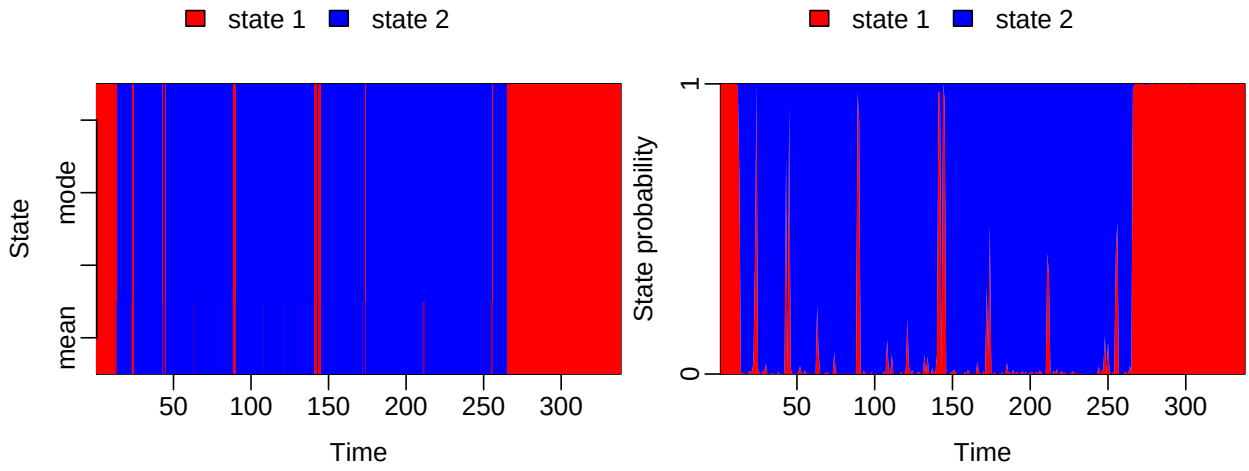
The two-state model shows no evidence of non-convergence (Gelman diagnostic near 1, Geweke z non-significant):

```
analyzeConvergence(Lamprey.doubleState)
```

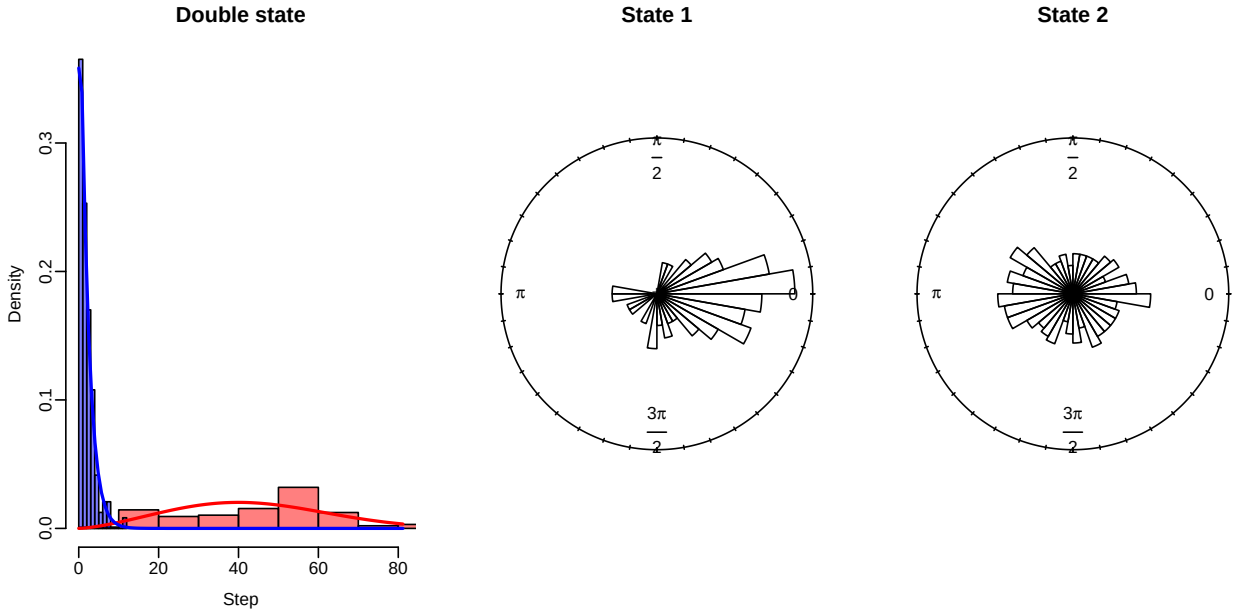
```
##                   Mean          SD         X2.5.        X97.5.
## a[1]      2.464143e+00  0.37383981    1.76984646    3.19205869
## a[2]      1.049607e+00  0.06662145    0.92333646    1.18467350
## b[1]      4.931853e+01  3.23675033   43.43154371   54.37937564
## b[2]      2.007867e+00  0.15337830    1.71320686    2.30596230
## deviance  6.607187e+03 18.15410551 6586.93898363 6646.10521592
## mu[1]    -2.030754e-03  0.04752768   -0.09847493    0.09222159
## mu[2]    -2.804701e+00  0.60960448   -3.75738667   -1.52746609
## rho[1]    7.287340e-01  0.03490390    0.65690801    0.78988256
## rho[2]    1.161659e-01  0.05870660    0.01117720    0.22755002
##          effectiveSize       acLag1 gelman.diag geweke[[i]]$z
## a[1]          1400.000 -0.006336417   1.0071205     0.6429455
## a[2]          1400.000  0.021615661   0.9998870     1.0435767
## b[1]          1410.656  0.039997170   1.0230590    -0.5033199
```

```
## b[2]          1400.000 -0.013327198   1.0085056    -0.5673558
## deviance      1261.469 -0.011667350   1.0411716    -1.6931750
## mu[1]         1426.106 -0.048890621   0.9998426    -1.6103701
## mu[2]         1400.000  0.027496496   1.0095541    -1.3642942
## rho[1]        1400.000 -0.026510167   1.0018636    -0.5061270
## rho[2]        1370.294 -0.054549612   1.0045038    -0.2916946
##          geweke[[i]]$z
## a[1]        -0.1164879
## a[2]         1.0142880
## b[1]         0.9070817
## b[2]         2.0031802
## deviance     0.3698925
## mu[1]        0.2262456
## mu[2]       -0.4283367
## rho[1]      -0.1338814
## rho[2]      -1.5355511
```

The beginning and end of the trajectory are assigned to the active state (state 1), as well as some steps in the middle sedentary phase (state 2).



We examine the assignment of steps and turning angles to each state. The larger steps were assigned to the active state, while the small steps were assigned to the sedentary state, with only a small degree of overlap. There is more overlap in turning angle distributions between states, though turning angle near 0 (straight ahead) are mostly assigned to the active state and those near π (reverse course) mostly assigned to the sedentary state.



Two-state switching model

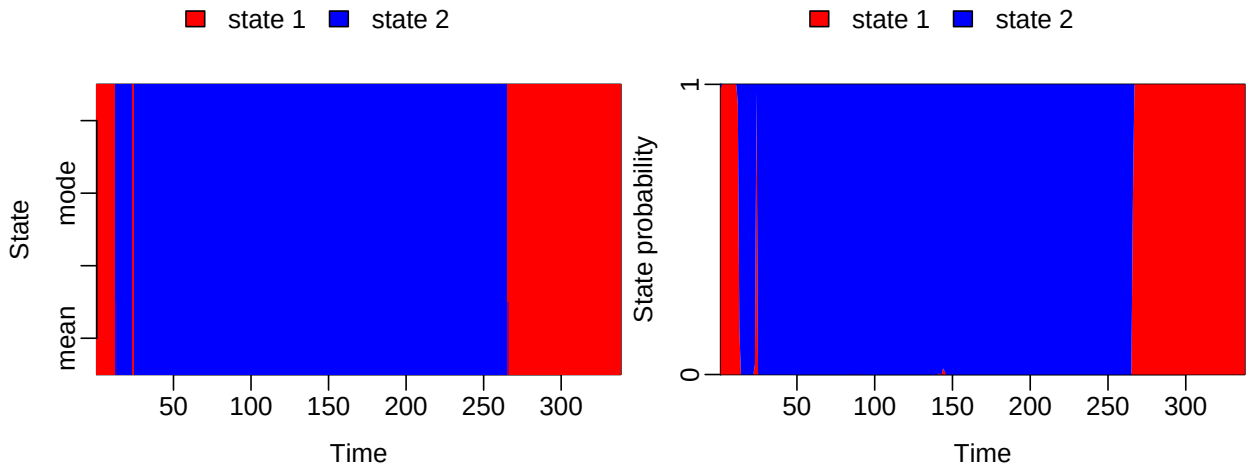
Again, no evidence of non-convergence:

```
analyzeConvergence(Lamprey.doubleStateSwitch)
```

```
##                   Mean          SD         X2.5.        X97.5.
## a[1]        3.53187201 0.321767382  2.920625e+00    4.19884526
## a[2]        0.89757987 0.043658588  8.134354e-01    0.98229215
## b[1]       54.24703422 1.809152150  5.073268e+01   57.87996119
## b[2]        2.37916778 0.175876503  2.049216e+00    2.73812815
## deviance 6589.23642760 5.938071121  6.581414e+03 6604.54868911
## mu[1]      -0.01459541 0.048392265 -1.155302e-01    0.07602620
## mu[2]      -2.77412344 0.553648639 -3.605591e+00   -1.63357678
## p[1]        0.96532500 0.019498636  9.188865e-01    0.99230559
## p[2]        0.01198622 0.006970837  2.222964e-03    0.02899685
## rho[1]      0.74771902 0.033302170  6.788436e-01    0.80819255
## rho[2]      0.11698437 0.050954968  1.734347e-02    0.22079849
##          effectiveSize       acLag1 gelman.diag geweke[[i]]$z
## a[1]          3479.633  0.011023468   1.0012949     0.5661434
## a[2]          3600.000  0.003704661   1.0001827    -0.5683059
## b[1]          3600.000  0.018894431   1.0004766     1.2696181
## b[2]          3573.779  0.020605721   0.9999525     0.1083452
## deviance      3825.184 -0.011413562   1.0001396     0.6582386
## mu[1]         3600.000  0.013536206   1.0004188     0.6934114
## mu[2]         3470.131  0.002176044   1.0014782    -0.4884161
## p[1]          3961.421  0.018975394   1.0013417     0.5979218
## p[2]          3463.041  0.004919769   1.0018417    -0.5320003
## rho[1]        3569.486  0.006049729   0.9998365     0.7582199
## rho[2]        3798.539 -0.014078373   1.0003313     0.1373218
##          geweke[[i]]$z
## a[1]        -1.5495825
## a[2]        -0.6700394
## b[1]        -3.0086735
## b[2]         0.4542364
```

```
## deviance    -1.1581677
## mu[1]       -1.1150146
## mu[2]       -0.1631495
## p[1]        -0.9612567
## p[2]         1.4792954
## rho[1]       0.3925924
## rho[2]       1.5312835
```

The state assignment is very definite, in that the probability of being in a particular state is always near 1. Again, the beginning and end are in the active state (state 1), but now with only a single brief blip in the middle also assigned to the active state.



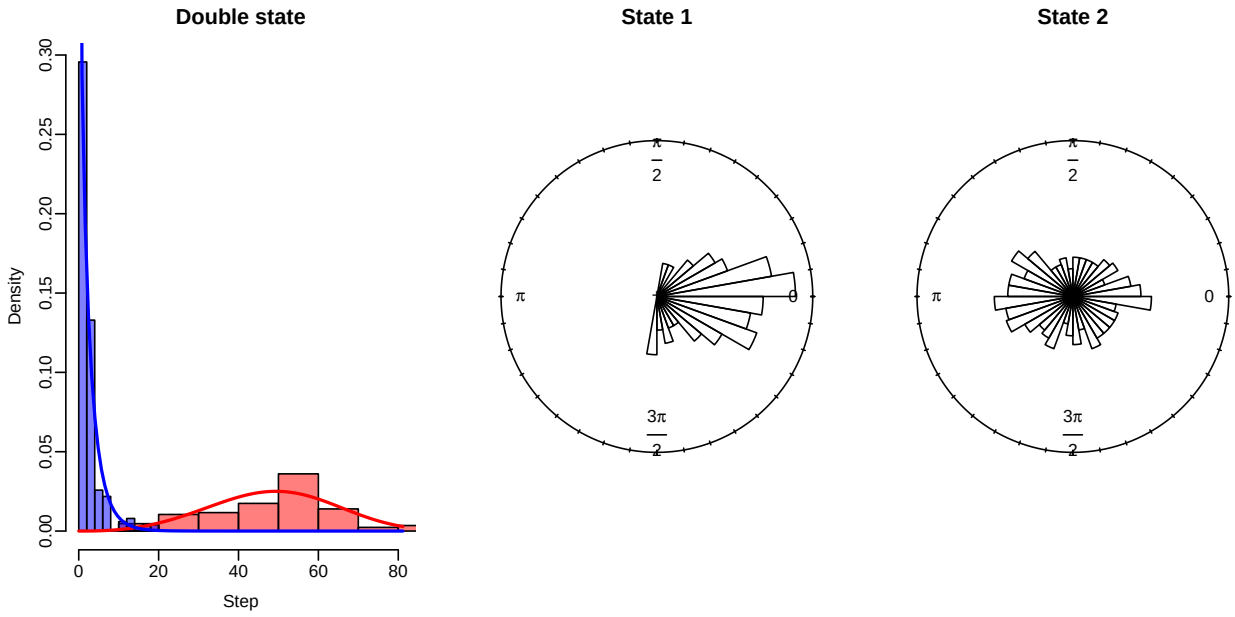
Again the states appear to be separated by both step size and turning angle, with more by step size with the larger steps being assigned to the active state and the smaller steps to the sedentary state. The separation of straight ahead and reverse turning angles is stronger than in the double state model.

```
## Warning in as.circular(xx[, 1]):  an object is coerced to the class 'circular' using default
value for the following components:
##  type:  'angles'
##  units:  'radians'
##  template:  'none'
##  modulo:  'asis'
##  zero:  0
##  rotation:  'counter'
## rose.diagtheta[getModeStates(doubleState) == 1]362State 1

## Warning in as.circular(x):  an object is coerced to the class 'circular' using default value
for the following components:
##  type:  'angles'
##  units:  'radians'
##  template:  'none'
##  modulo:  'asis'
##  zero:  0
##  rotation:  'counter'
## conversion.circularxradiansmodulo
```

```
## Warning in as.circular(xx[, 1]):  an object is coerced to the class 'circular' using default
value for the following components:
##  type:  'angles'
##  units:  'radians'
##  template:  'none'
##  modulo:  'asis'
##  zero:  0
##  rotation:  'counter'
## rose.diagtheta[getModeStates(doubleState) == 2]362State 2

## Warning in as.circular(x):  an object is coerced to the class 'circular' using default value
for the following components:
##  type:  'angles'
##  units:  'radians'
##  template:  'none'
##  modulo:  'asis'
##  zero:  0
##  rotation:  'counter'
## conversion.circularxradiansmodulo
```



And we can also extract the transition matrix.

```
getTransitionMatrix(Lamprey.doubleStateSwitch)


##             [,1]      [,2]
## [1,] 0.96532500 0.0346750
## [2,] 0.01198622 0.9880138
```

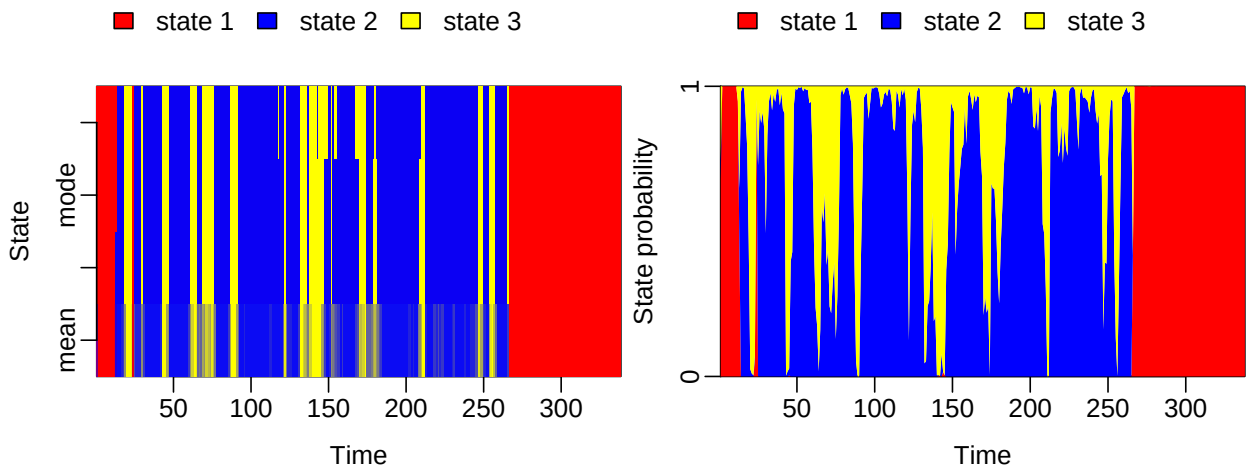
Three-state switching model
Finally, we can examine the triple state switching model, to see if there is any evidence of another state. There is no evidence of non-convergence.

```
analyzeConvergence(Lamprey.tripleStateSwitch)
```

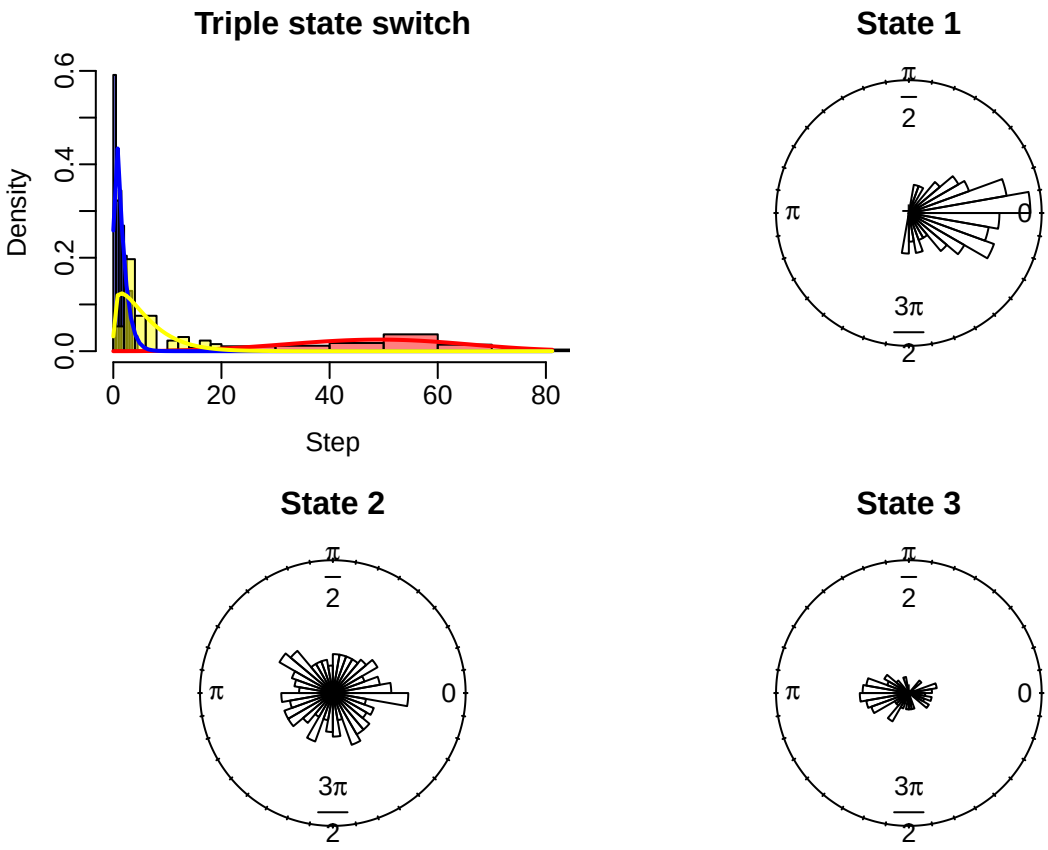
```
##                   Mean           SD         X2.5.        X97.5.
## a[1]      3.483051e+00  0.324886948  2.875438e+00    4.16663036
## a[2]      1.153309e+00  0.079296981  1.002557e+00    1.31379522
## a[3]      1.212444e+00  0.174263582  9.275370e-01    1.61188537
## b[1]      5.414744e+01  1.804433230  5.069147e+01   57.69839270
## b[2]      1.460715e+00  0.162724113  1.147375e+00    1.78757231
## b[3]      6.047274e+00  1.367573149  4.098217e+00    9.62623103
## deviance  6.434709e+03 16.420139214  6.404735e+03 6470.09986235
## mu[1]    -1.367527e-02  0.046527557 -1.071714e-01    0.07273961
## mu[2]    -1.004372e+00  1.439683707 -4.336549e+00    1.24858118
## mu[3]    -3.019270e+00  0.102898270 -3.209378e+00   -2.79590215
## p[1]      9.579842e-01  0.022993688  9.030782e-01    0.99099371
## p[2]      8.426396e-03  0.007773594  2.821249e-04    0.02731204
## p[3]      4.041539e-02  0.029502504  4.629346e-03    0.11737983
## phi[1]    6.280342e-01  0.258871272  9.577220e-02    0.98507055
## phi[2]    8.926572e-01  0.037915676  8.025432e-01    0.95104377
## phi[3]    2.634488e-01  0.096450601  1.102083e-01    0.48275622
## rho[1]    7.504407e-01  0.033471559  6.789035e-01    0.81157917
## rho[2]    7.070364e-02  0.056588459  2.283723e-03    0.21307958
## rho[3]    5.876642e-01  0.086918760  4.095890e-01    0.74860286
##          effectiveSize       acLag1 gelman.diag geweke[[i]]$z
## a[1]          3200.000 -0.008828643   1.0053891   -1.80885268
## a[2]          3330.954 -0.008878090   1.0004809    0.27534062
## a[3]          3044.500  0.042461785   1.0029876    0.54138947
## b[1]          3646.147 -0.022464237   1.0005004   -1.18237917
## b[2]          3200.000 -0.002556758   1.0059264    0.80168699
## b[3]          3200.000  0.008882482   1.0015657    0.95927979
## deviance      3200.000  0.008840805   1.0112769    0.73914788
## mu[1]         2765.519  0.024185564   1.0004277   -0.69111950
## mu[2]         3385.721 -0.023274794   1.0000795    0.76779986
## mu[3]         3200.000 -0.009608040   0.9995613   -0.08102209
## p[1]          3200.000  0.026256933   1.2637334   -0.47883227
## p[2]          2806.812  0.009230566   1.0006739   -0.06711875
## p[3]          3207.021  0.029380982   1.0165744   -0.25810561
## phi[1]        3324.620 -0.028578995   1.0128243   -0.89564692
## phi[2]        3382.655 -0.038029201   0.9998304    0.87510246
## phi[3]        3200.000  0.018995233   1.0218359    0.18374466
## rho[1]        3200.000 -0.013221142   1.0000597    0.68738963
## rho[2]        3820.892 -0.009980454   0.9997134   -0.57659438
## rho[3]        3414.584 -0.024022292   1.0022904    0.83732991
##          geweke[[i]]$z
## a[1]        0.09444167
## a[2]       -1.22349118
## a[3]       -1.20801613
## b[1]       -2.24044941
## b[2]       -0.66181541
## b[3]       -1.75438178
## deviance   -0.76208750
## mu[1]      -0.19818666
## mu[2]      -0.00137399
## mu[3]      -0.20556070
## p[1]       -1.25612771
## p[2]       -1.48339200
## p[3]        0.27753196
## phi[1]      1.92156700
## phi[2]     -1.80912309
## phi[3]     -0.11138220
```

```
## rho[1]      0.48145105
## rho[2]      0.26499277
## rho[3]      0.25256349
```

The active state (state 1) is very similar to the active state in the double state switch model, and the sedentary state is broken into two states.



The active state parameters are very similar to the double state switch model. The two sedentary states consist of a state with small steps in all directions (state 2) and a state with larger steps and direction reversals (state 3).



Change point comparison

Looking at where each model identifies changes from one behavior state to the next, we see that the double state switching model has the fewest change points, and is a subset of both the double state and triple state switch models. Interestingly, the triple state switch model has the most change points, even though the double state model includes no persistence, so there is less constraining state assignment.

```
# Double state
cbind(Lamprey.VT, state = getModeStates(Lamprey.doubleState))[
  which(diff(getModeStates(Lamprey.doubleState)) != 0), c(3:5, 10:12)]
```

```
##              S        Phi        Theta          V             T.POSIX
## 14   7.5369389  1.3526608  0.004821148  5426.5960 2010-05-02 09:16:00
## 24   7.7652215 -2.6327930 -5.583655147  5590.9595 2010-05-02 09:36:00
## 25  52.4119407  2.8676294  5.500422403 37736.5973 2010-05-02 09:38:00
## 43   0.3466376 -2.8908766 -0.759987608   249.5791 2010-05-02 10:14:00
## 44  13.5509032  1.3132600  4.204136594  9756.6503 2010-05-02 10:16:00
## 45   3.3514993 -1.8113462 -3.124606174  2413.0795 2010-05-02 10:18:00
## 46  10.1547235 -1.8192143 -0.007868138  7311.4009 2010-05-02 10:20:00
## 89   3.8859063  3.1397913  3.321768205  2797.8525 2010-05-02 11:46:00
## 91  15.6577475 -2.5295508 -3.088392823 11273.5782 2010-05-02 11:50:00
```

```
## 141  7.9436381 -0.4825744 -2.621215743  5719.4194 2010-05-02 13:30:00
## 143 19.7105991  2.9831474  3.312293550 14191.6314 2010-05-02 13:34:00
## 144  0.7123038  0.1921093 -2.791038132   512.8587 2010-05-02 13:36:00
## 146 17.5054739 -2.7776056 -3.168361248 12603.9412 2010-05-02 13:40:00
## 174  3.6667265 -0.4990921 -0.460431126  2640.0431 2010-05-02 14:36:00
## 175 12.0887090  3.0489773  3.548069377  8703.8705 2010-05-02 14:38:00
## 256  6.5711688 -0.3857890  0.256880441  4731.2416 2010-05-02 17:20:00
## 257 12.2872229  2.7554431  3.141232067  8846.8005 2010-05-02 17:22:00
## 266  1.2159359  2.5969419  0.577263624   875.4738 2010-05-02 17:40:00
##     state
## 14      1
## 24      2
## 25      1
## 43      2
## 44      1
## 45      2
## 46      1
## 89      2
## 91      1
## 141     2
## 143     1
## 144     2
## 146     1
## 174     2
## 175     1
## 256     2
## 257     1
## 266     2
```

```
# Double state switch
cbind(Lamprey.VT, state = getModeStates(Lamprey.doubleStateSwitch))[
  which(diff(getModeStates(Lamprey.doubleStateSwitch)) != 0), c(3:5, 10:12)]
```

```
##             S       Phi       Theta          V             T.POSIX state
## 13  14.901510  1.347840  0.04514548 10729.0869 2010-05-02 09:14:00     1
## 24   7.765222 -2.632793 -5.58365515  5590.9595 2010-05-02 09:36:00     2
## 25  52.411941  2.867629  5.50042240 37736.5973 2010-05-02 09:38:00     1
## 266  1.215936  2.596942  0.57726362   875.4738 2010-05-02 17:40:00     2
```

```
# Double state switch

cbind(Lamprey.VT, state = getModeStates(Lamprey.tripleStateSwitch))[
  which(diff(getModeStates(Lamprey.tripleStateSwitch)) != 0), c(3:5, 10:12)]
```

```
##              S         Phi         Theta          V             T.POSIX
## 14   7.5369389  1.35266081  4.821148e-03  5426.5960 2010-05-02 09:16:00
## 19   0.5806422  2.07280956  3.945381e+00   418.0624 2010-05-02 09:26:00
## 24   7.7652215 -2.63279304 -5.583655e+00  5590.9595 2010-05-02 09:36:00
## 25  52.4119407  2.86762936  5.500422e+00 37736.5973 2010-05-02 09:38:00
## 30   2.3506419  2.02384672 -8.411939e-01  1692.4622 2010-05-02 09:48:00
## 31   6.3796991 -0.09565711 -2.119504e+00  4593.3834 2010-05-02 09:50:00
## 43   0.3466376 -2.89087658 -7.599876e-01   249.5791 2010-05-02 10:14:00
## 48   3.1906592  0.74449828  3.222119e+00  2297.2746 2010-05-02 10:24:00
## 61   1.3475807 -0.12948179 -2.317286e+00   970.2581 2010-05-02 10:50:00
## 66   3.3138771  0.53796647  2.914544e+00  2385.9915 2010-05-02 11:00:00
## 69   1.8868385  0.66648867  1.060967e+00  1358.5237 2010-05-02 11:06:00
## 77   2.2511970  0.33729429  3.070737e+00  1620.8618 2010-05-02 11:22:00
## 87   0.8607901 -3.06251307 -2.230365e+00   619.7689 2010-05-02 11:42:00
## 92   2.6131093  1.48727373  4.016825e+00  1881.4387 2010-05-02 11:52:00
```

```
## 122  4.3149519 -0.10137203 -7.037516e-02  3106.7653 2010-05-02 12:52:00
## 123  6.8246386  3.09649562  3.197868e+00  4913.7398 2010-05-02 12:54:00
## 132  1.8105262  2.77852309  4.875661e+00  1303.5789 2010-05-02 13:12:00
## 137  1.5407770  3.03299244  3.400855e+00  1109.3595 2010-05-02 13:22:00
## 138  1.5781567  2.91665558 -1.163369e-01  1136.2728 2010-05-02 13:24:00
## 148  2.6515621 -2.68754197 -2.724325e+00  1909.1247 2010-05-02 13:44:00
## 152  1.2677510 -0.35318517  1.355578e-01   912.7807 2010-05-02 13:52:00
## 153  4.7217068  2.57284601  2.926031e+00  3399.6289 2010-05-02 13:54:00
## 170  2.0785929 -0.04379361 -1.521235e+00  1496.5869 2010-05-02 14:28:00
## 175 12.0887090  3.04897731  3.548069e+00  8703.8705 2010-05-02 14:38:00
## 179  2.3228243  2.73873214  2.432000e+00  1672.4335 2010-05-02 14:46:00
## 182  2.4655032 -0.86095584 -3.893890e+00  1775.1623 2010-05-02 14:52:00
## 209  2.8751417 -2.65852404 -3.297681e+00  2070.1021 2010-05-02 15:46:00
## 213 11.0073713 -2.66716405 -3.080943e+00  7925.3073 2010-05-02 15:54:00
## 247  2.0071698 -0.36155797 -6.059777e-01  1445.1623 2010-05-02 17:02:00
## 250  3.4623084 -0.15207400  2.957999e+00  2492.8621 2010-05-02 17:08:00
## 254  0.2475636 -3.10523036 -3.759465e-09   178.2458 2010-05-02 17:16:00
## 258  2.1548286 -0.72235162 -3.477795e+00  1551.4766 2010-05-02 17:24:00
## 266  1.2159359  2.59694187  5.772636e-01   875.4738 2010-05-02 17:40:00
## 267 17.9403404  1.00141741 -1.595524e+00 12917.0451 2010-05-02 17:42:00
##     state
## 14      1
## 19      2
## 24      3
## 25      1
## 30      2
## 31      3
## 43      2
## 48      3
## 61      2
## 66      3
## 69      2
## 77      3
## 87      2
## 92      3
## 122     2
## 123     3
## 132     2
## 137     3
## 138     2
## 148     3
## 152     2
## 153     3
## 170     2
## 175     3
## 179     2
## 182     3
## 209     2
## 213     3
## 247     2
## 250     3
## 254     2
## 258     3
## 266     2
## 267     3
```

Model comparison

We can compare models using DIC (a function of the deviance and the estimated number of parameters, pD).

```
getDIC(list(Lamprey.doubleState, Lamprey.doubleStateSwitch, Lamprey.tripleStateSwitch))
```

```
##                             DIC        pD
## DoubleState.txt       6771.697 164.50965
## DoubleStateSwitch.txt 6606.869  17.63259
## TripleStateSwitch.txt 6569.137 134.42775
```

For these data, the DIC supports selecting the three-state switching model, but the two-state switching model is quite close.