Smooth swimming and fanciful flights: Using the `smoove` package

Eliezer Gurarie

December 3, 2017

Contents

Abstract

The `smoove` package is a collection of functions to work with continuous time correlated velocity movement (CVM) models as described in Gurarie et al. (in review, *Movement Ecology*). There are functions that simulate, diagnose and estimate these movement models, as well as functions for facilitating a change point analysis.

1 Correlated velocity movement models

Table 1: Summary of CVM models

Model	α	$\boldsymbol{\mu}$	Notation
Unbiased CVM	$1/\tau$	0	UCVM(τ, ν)
Advective CVM	$1/\tau$	non-zero	ACVM$(\tau, \eta, \mu_x, \mu_y)$
Rotational CVM	$1/\tau + i\omega$	0	RCVM(τ, η, ω)
Rotational-Advective CVM	$1/\tau + i\omega$	non-zero	RACVM$(\tau, \eta, \omega, \mu_x, \mu_y)$

Briefly, CVM models are movement models in which the velocity is assumed to be an autocorrelated stochastic process taking the form of an Ornstein-Uhlenbeck equation:

$$\begin{aligned} d\mathbf{v} &= \alpha(\boldsymbol{\mu} - \mathbf{v})\, dt + \frac{\eta}{\sqrt{\tau}}\, d\mathbf{w}_t, \\ \mathbf{v}(0) &= \mathbf{v}_0, \end{aligned} \tag{1}$$

where the key parameters are τ - the characteristic time scale of autocorrelation, η - the root mean squared speed of the movement process. Different valeus of the additional parameters α and $\boldsymbol{\mu}$ are related to variations of a CVM process, as per table 1. We present the models together with functions that estimate them sequentially below.

2 Simulation

2.1 Unbiased CVM

The unbiased CVM is the continuous time equivalent of an unbiased correlated random walk, i.e. a movement that has local persistence in direction. This is the most "fundamental" CVM model and its mean speed is given by a simple expression $\nu = \eta\sqrt{\pi}/2$, the a convenient parmeterization is as: UCVM(τ, ν). Also, it is for this model that the complete set of estimation methods discussed in the paper is available. For these reasons, it is treated slightly differently than the (R/A)CVM models in the `smoove` package.

The function for simulating the UCVM is `simulateUCVM`. There are two methods of simulation, *direct* and *exact*.

2.1.1 Direct method

The direct method works by discretizing the differential equation 1, i.e.:

$$V_{i+1} = V_i - V_i(dt/\tau) + \frac{2\nu}{\sqrt{\pi\tau}} dW_i$$

where V is a vector of complex numbers and dW_i is drawn from a bivariate normal distribution with variance $\sqrt{dt}$. The direct method provides good simulations relatively quickly at high resolution ($\Delta t \ll \tau$).

Loading the package:

```
require(smoove)
require(magrittr)
require(plyr)
require(scales)
```

Note, the package has many dependencies related to matrix manipulations.

An example of a track with $\nu = 2$, $\tau = 5$, for a time interval of 1000:

```
nu <- 2
tau <- 5
dt <- .1
ucvm1 <- simulateUCVM(nu=nu, tau=tau, T.max = 1000, dt = dt)
```

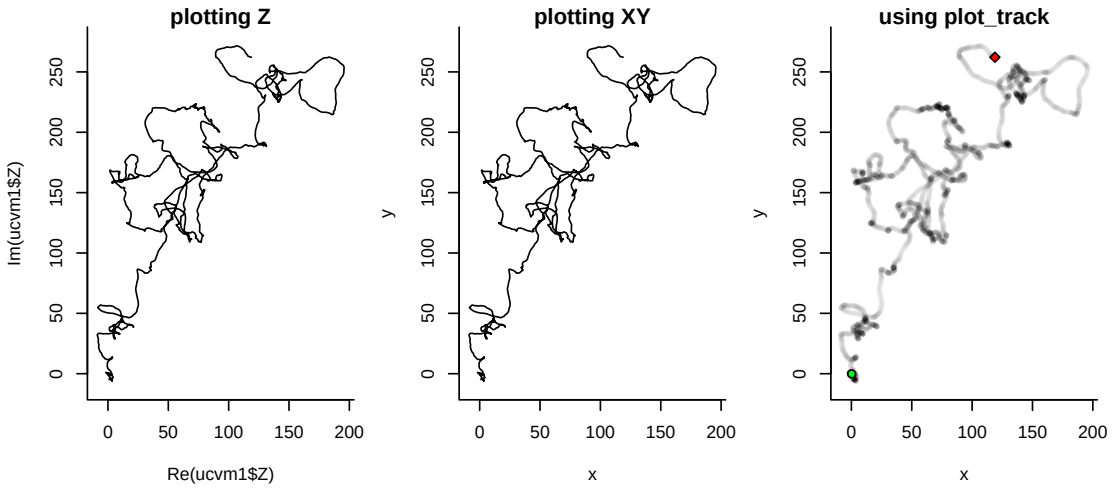
The resulting object contains complex velocity and position vectors, a time vector, an XY matrix, and the values of the simulated parameters.

```
str(ucvm1)
```

```
## List of 5
##  $ T          : num [1:10001] 0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 ...
##  $ V          : cplx [1:10001] 1.55-1.27i 1.7-1.19i 1.99-0.95i ...
##  $ XY         : num [1:10001, 1:2] 0.155 0.325 0.524 0.727 0.868 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:2] "x" "y"
##  $ Z          : cplx [1:10001] 0.155-0.127i 0.325-0.246i 0.524-0.341i ...
##  $ parameters:List of 3
##   ..$ tau: num 5
##   ..$ nu : num 2
##   ..$ v0 : cplx 1.55-1.27i
```

The curve can be plotted in several ways:

```
plot(ucvm1$Z, asp=1, type="l", main = "plotting Z")
plot(ucvm1$XY, asp=1, type="l", main = "plotting XY")
plot_track(ucvm1$XY, col=rgb(0,0,0,.01), main = "using plot_track")
```



The `plot_track` function is a simple convenience function for plotting tracks. According to theory, the mean speed of this track should just be $\nu = 2$:

```
mean(Mod(ucvm1$V))
```

```
## [1] 2.031516
```

and the root mean square speed should be $2\nu/\sqrt{\pi} = 2.2567583$:

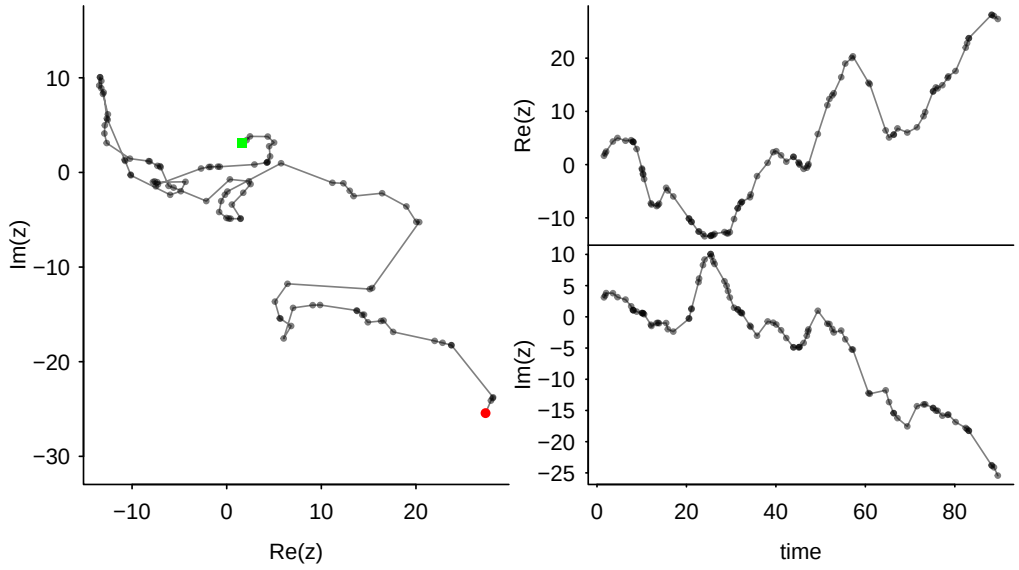
```
sqrt(mean(Mod(ucvm1$V)^2))
```

```
## [1] 2.288807
```

2.1.2 Exact method

The *exact* method is more flexible, as it samples the position and velocity process simultaneously from the complete mean and variance-covariance structure of the integreated OU process. The main advantage is that the process can be simulated for irregular (and totally arbitrary) times of observation:

```
T <- cumsum(rexp(100))
ucvm2 <- simulateUCVM.exact(T = T, nu = 2, tau = 2)
with(ucvm2, scan_track(time = T, z = Z))
```



The `scan_track` function is another function that is convenient for seeing a process in time (right plots). Note the irregularity of the sampling.

```
summary(diff(ucvm2$T))
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## 0.001575 0.224599 0.594203 0.889439 1.305699 5.066716
```

and that the velocity vector has the correct statistical properties (mean and 95% C.I.):

```
mean(Mod(ucvm2$V))
```

```
## [1] 2.069103
```

```
with(ucvm2, mean(Mod(V)) + c(-2,2)*sd(Mod(V)) / sqrt(length(V)))
```

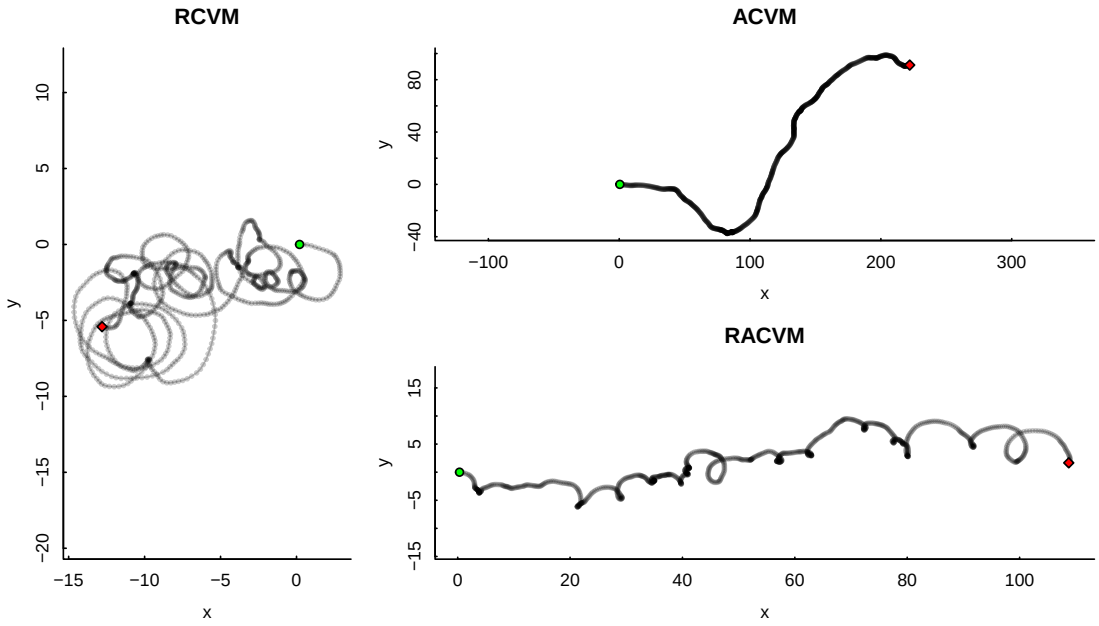
```
## [1] 1.869962 2.268244
```

2.2 Rotational-Advective CVM

The (R/A)CVM models are parameterized in terms of τ, the random rms speed η, and then some combination of the advection vector $\boldsymbol{\mu}$ and/or angular speed ω.

```
rcvm <- simulateRACVM(tau = 6, eta = 2, omega = 1, mu = 0, Tmax = 1000, dt = .1)
acvm <- simulateRACVM(tau = 6, eta = 2, omega = 0, mu = 2, Tmax = 1000, dt = .1)
racvm <- simulateRACVM(tau = 6, eta = 2, omega = 1, mu = 1, Tmax = 1000, dt = .1)
```

```
plot_track(rcvm$Z[rcvm$T < 100],  main = "RCVM")
plot_track(acvm$Z[acvm$T < 100], main = "ACVM")
plot_track(racvm$Z[racvm$T < 100], main = "RACVM")
```



Note that we only plot these curves within the first 100 time units. For the moment, the RACVM models can only be simulated "directly", i.e. via direct forward simulations rather than sampling from the complete autocorrelated process.

3 Empirical velocity auto-correlation function

The velocity autocovariance function (VAF) is a very useful way to visualize the autocorrelation structure of the movement models. It is defined as the expectation of a dot product of the velocity vector with itself as a function of lags:

$$C_v(\Delta t) = \mathrm{E}\left[\mathbf{v}(t_0 + \Delta t) \cdot \mathbf{v}(t_0)\right], \tag{2}$$

S At lag 0, the value is mean squared speed of the process, and at long lags it is the square of the mean drift of the process. For CVM models, $C_v(0) = \eta^2 + \mu^2$, $\lim_{\Delta t \to \infty} C_v(\Delta t) = \mu^2$, and the function decays exponentially at rate τ with an oscillatory component with angular velocity ω. The main functions for computing the empirical VAF is the `getEVAF` function:

```
ucvm.vaf <- with(ucvm1, getEVAF(Z = Z, T = T, lagmax = 30))
acvm.vaf <- with(acvm, getEVAF(Z = Z, T = T, lagmax = 30))
rcvm.vaf <- with(rcvm, getEVAF(Z = Z, T = T, lagmax = 30))
racvm.vaf <- with(racvm, getEVAF(Z = Z, T = T, lagmax = 30))
```

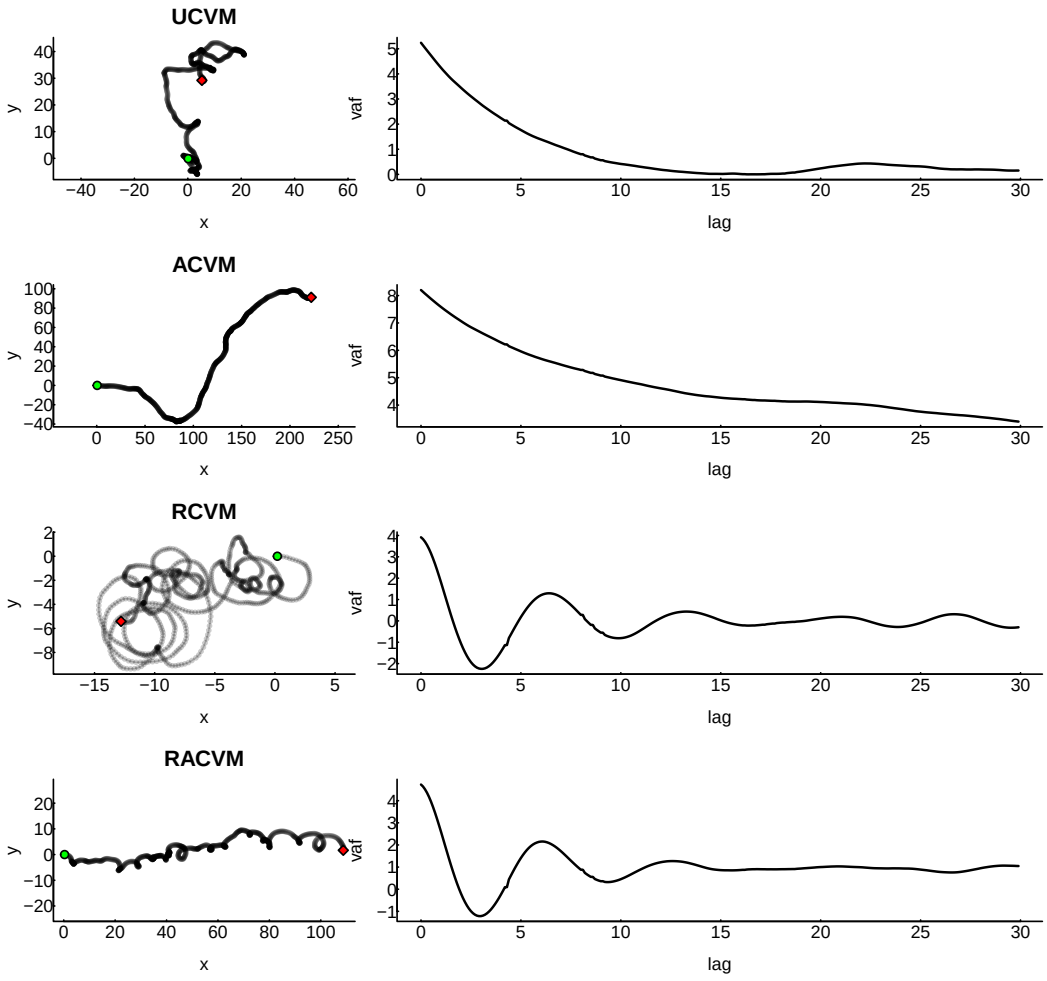
This function produces a two column data frame with the lag and the respective computed evaf:

```
head(ucvm.vaf)

##   lag      vaf
## 1 0.0 5.238760
```

```
## 2 0.1 5.136165
## 3 0.2 5.034438
## 4 0.3 4.933219
## 5 0.4 4.833052
## 6 0.5 4.734453
```

A plot of the four curves and their respective empirical autocovariance functions:

```
plot_track(ucvm1$Z[1:1e3], main = "UCVM"); plot(ucvm.vaf, type="l", lwd=1.5)
plot_track(acvm$Z[1:1e3], main = "ACVM"); plot(acvm.vaf, type="l", lwd=1.5)
plot_track(rcvm$Z[1:1e3], main = "RCVM"); plot(rcvm.vaf, type="l", lwd=1.5)
plot_track(racvm$Z[1:1e3], main = "RACVM"); plot(racvm.vaf, type="l", lwd=1.5)
```



The velocity autocovariance functions is a very useful visual summary of a movement track because it refeals . Furthermore These curves can be used to estimate the parameters of these processes.

4 Estimation of CVM models

The main estimation functions are `estimateUCVM` and `estimateRACVM`. The respective help files have abundant examples of implementation.

4.1 UCVM

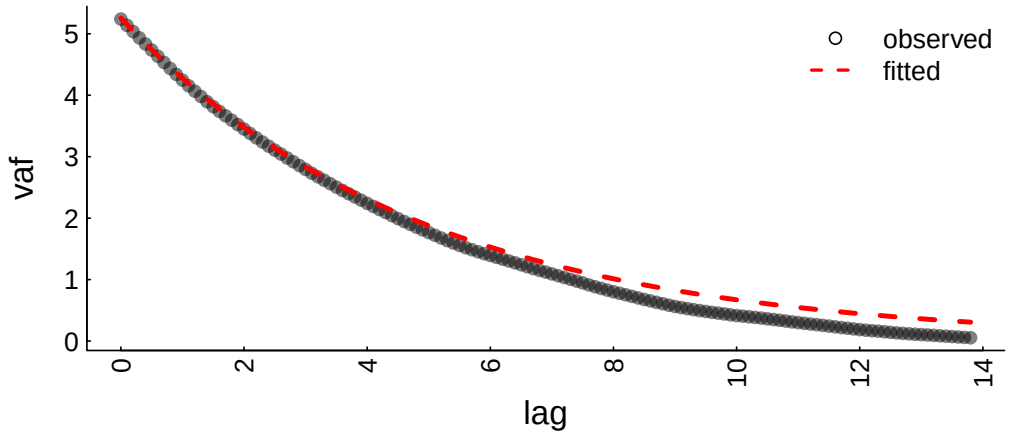
Following the structure of the Gurarie at al. (*in review*), manuscript, the UCVM, which focusses on estimates of time scale τ and speed ν, can be estimated using one of five different methods.

4.1.1 VAF fitting

This method relies on fitting the *empirical* velocity autocorrelation function (above) to the *theoretical* velocity autocorrelation function:

$$C_v(\Delta t) = \frac{4}{\pi}\nu^2 \exp(\Delta t/\tau)$$

```
estimateUCVM(Z = ucvm1$Z, T = ucvm1$T, method = "vaf", diagnose = TRUE, CI=TRUE)
```




```
##     Estimate  C.I.low C.I.high
## tau 4.858959 4.791005 4.928869
## nu  2.031519 1.802198 2.260839
```

The diagnostic plot (only plotted with `diagnose=TRUE`) illustrates the quality of the fit. The estimates should be fairly good, when compared to the true parameters:

```
ucvm1$parameters[1:2]

## $tau
## [1] 5
##
```

```
## $nu
## [1] 2
```

A lower-resolution track will have wider confidence intervals:

```
ucvm.lores <- simulateUCVM(nu=10, tau = 4, dt = 1, T.max = 1000, method = "exact")
with(ucvm.lores, estimateUCVM(Z = Z, T = T, CI=TRUE, method="vaf"))
```

```
##     Estimate  C.I.low C.I.high
## tau 4.330770 3.761759 5.102600
## nu  9.352172 8.894749 9.809595
```

The speed estimate might be improved by using a spline correction

```
with(ucvm.lores, estimateUCVM(Z = Z, T = T, CI=TRUE, method="vaf", spline = TRUE))
```

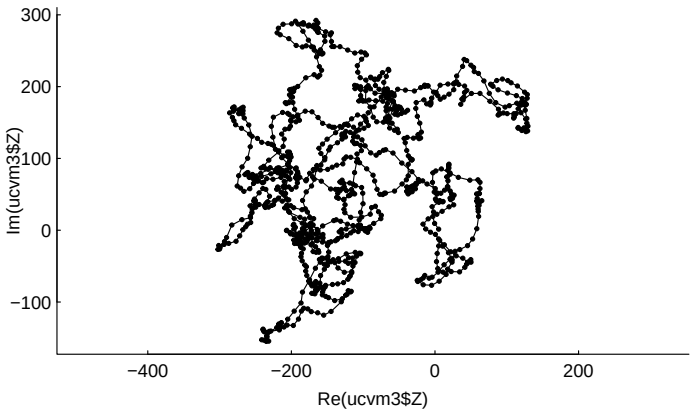
```
##     Estimate  C.I.low C.I.high
## tau 3.980474 3.548952 4.531460
## nu  9.574312 9.194147 9.954476
```

NB: This method works only for regularly sampled data, and the confidence intervals tend to be unreliable.

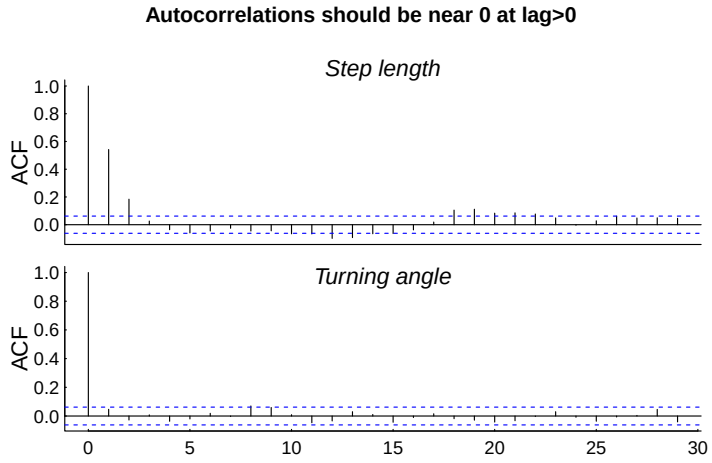
4.1.2 CRW matching

The Correlated Random Walk (CRW) matching method computes the CRW parameters (shape and scale of length steps and wrapped Cauchy clustering coefficient) and converts those to time-scales and speed estimates. Confidence intervals are obtained by Monte Carlo draws from the confidence intervals around likelihood estimates of the CRW parameters.

```
tau <- 2; nu <- 8
ucvm3 <- simulateUCVM(T=1:1000, nu = nu, tau = tau, method = "exact")
plot(ucvm3$Z, asp=1, type="o", cex=0.5, pch=19)
```



```
estimateUCVM(Z = ucvm3$Z, T = ucvm3$T, CI=TRUE, method="crw", diagnose=TRUE)
```




```
##     Estimate  C.I.low C.I.high
## tau 1.992912 1.779502 2.293856
## nu  6.661996 6.081514 7.174524
```

The diagnosis plots (crudely) illustrates the standard autocorrelation function for the step lengths and turning angles. Under the normal assumptions of the CRW, these should both be around 0 at lags greater than 0. The more autocorrelated either of these time series is, the more biased the estimates are likely to be.

NB: This method is illustrative, generally biased, and not recommended except for data that are relatively coarsely sampled. It is, however, very fast to compute.

4.1.3 Velocity likelihood

The velocity likelihood method is based on using the known distributional properties of the integrated OU obtain a "one-step" likelihood of each velocity based on the previous velocity. This method is robust to irregularly sampled data and is fairly fast. We will estimate the original track

```
estimateUCVM(Z = ucvm1$Z, T = ucvm1$T, method = "vLike", CI=TRUE)
```

```
##     Estimate  C.I.low C.I.high
## tau 5.035226 4.375848 5.793963
## nu  2.026834 1.885991 2.167676
```

This method gives very reliable confidence intervals, but may underestimate velocities. We can also run it on the irregularly sampled track from above:

```
estimateUCVM(Z = ucvm2$Z, T = ucvm2$T, method = "vLike", CI=TRUE)
```

```
##     Estimate  C.I.low C.I.high
## tau 2.826492 1.863552 4.287003
## nu  1.802529 1.475147 2.129912
```

The confidence intervals are more wide this time (due to a much smaller data set: 100 versus 10001 data points), but the estimates should be fairly accurate.

4.1.4 Position likelihood

The position likelihood (`zLike`) takes the complete correlation structure between the velocities and the positions to estimate the parameters. It is prohibitively slow on a data set the size of `ucvm1`, but quite fast on the second (irregular, short) sampling:

```
estimateUCVM(Z = ucvm2$Z, T = ucvm2$T, method = "zLike", CI=TRUE)
```

```
##     Estimate   C.I.low C.I.high
## tau 1.799556  1.268449 2.553041
## nu  1.953387  1.570523 2.336250
## v0x 1.051625 -1.244793 3.348044
## v0y 3.160491  0.864072 5.456909
```

Note, improvements in the parameter estimates and confidence intervals, and the inclusion of estimates for the initial speed (usually a parameter of less interest).

4.1.5 Position likelihood: Kalman filter

We (OO and EG) had independently worked out many of the details of the estimation of full position likelihood before we discovered that the same likelihood was estimated in a much more efficient way by Johnson et al. (2008), and encoded in an excellent package called "crawl" (*Correlated RAndom Walk Library*). We have incorporated a wrapper for this package in `smoove` for the estimation of UCVM parameters - a fairly narrow application of the capabilities of `crawl`. First, we run it on the irregular data set:

```
with(ucvm2, estimateUCVM(Z = Z, T = T, method = "crawl"))
```

```
## Beginning SANN initialization ...
## Beginning likelihood optimization ...
```

```
##
##
## Continuous-Time Correlated Random Walk fit
##
## Models:
## --------
## Movement   ~ 1
## Error
##
##
##                       Parameter Est. St. Err. 95% Lower 95% Upper
## ln sigma (Intercept)          1.090    0.152     0.793     1.388
## ln beta (Intercept)          -0.585    0.181     -0.94    -0.231
##
##
```

```
## Log Likelihood = 85.236
## AIC = -166.472
##     Estimate        L        U
## tau 1.795195 1.259472 2.558792
## nu  1.968246 1.462124 2.649563
```

The estimates and C.I.'s for ν and τ are very similar to the full position likelihood above. The additional output is specific to the parameterization Johnson et al. use, the lower output (`nutau`) is consistent with the output for the other methods.
The `crawl` method can handle a very long ($n = 10000$) dataset as well.

```
with(ucvm1, estimateUCVM(Z = Z, T = T, method = "crawl"))
```

```
## Beginning SANN initialization ...
## Beginning likelihood optimization ...
##
##
## Continuous-Time Correlated Random Walk fit
##
## Models:
## --------
## Movement   ~ 1
## Error
##
##
##                      Parameter Est. St. Err. 95% Lower 95% Upper
## ln sigma (Intercept)          1.368    0.054     1.262     1.474
## ln beta (Intercept)          -1.069    0.055    -1.177    -0.962
##
##
## Log Likelihood = 58195.74
## AIC = -116387.481
##     Estimate        L        U
## tau 2.913263 2.616068 3.244220
## nu  2.038729 1.833277 2.267206
```

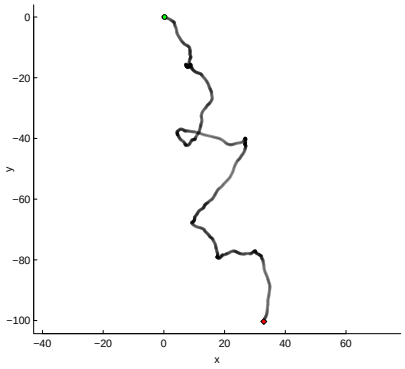
with correspondingly smaller confidence intervals. There are many additional options that can be passed to the `crawl` solver, detailed in the help file for `crwMLE`.

4.2 RACVM

Estimating the RACVM processes is different mainly in that there are now four different models to compare (with and without each of rotation and advection), and a model selection method is intrinsically built in. For the time being, the method implemented for fitting these models is the *velocity likelihood*. In the examples below, we estimate the parameters from each of the RACVM simulations and them from a portion of actual kestrel flight data:

```
ucvm <- simulateRACVM(tau = 5, eta = 2, omega = 0, mu = 0, Tmax = 100, dt = .1)
acvm <- simulateRACVM(tau = 5, eta = 2, omega = 0, mu = 2, Tmax = 100, dt = .1)
rcvm <- simulateRACVM(tau = 5, eta = 2, omega = 2, mu = 0, Tmax = 100, dt = .1)
racvm <- simulateRACVM(tau = 5, eta = 2, omega = 2, mu = 2, Tmax = 100, dt = .1)
```

4.2.1 Unbiased CVM




```
with(ucvm, estimateRACVM(Z=Z, T=T, compare.models=TRUE))
```

```
## $results
##                eta       omega       mu.x       mu.y      tau      rms
## Estimate 1.706008  0.03174857  0.2129594 -1.0759083 3.506698 2.072366
## CI.low   1.383267 -0.07776853 -0.4245069 -1.7133763 2.388803 1.633966
## CI.high  2.028749  0.14126567  0.8504257 -0.4384404 5.147736 2.510766
##
## $LL
## [1] -320.2632
##
## $CompareTable
##         logLike k      AIC deltaAIC      BIC  deltaBIC
## UCVM  -324.7869 2 653.5738 4.711784 663.3893  0.000000
## ACVM  -320.4310 4 648.8620 0.000000 668.4930  5.103726
## RCVM  -324.5393 3 655.0786 6.216608 669.8019  6.412580
## RACVM -320.2632 5 650.5264 1.664399 675.0652 11.675880
```

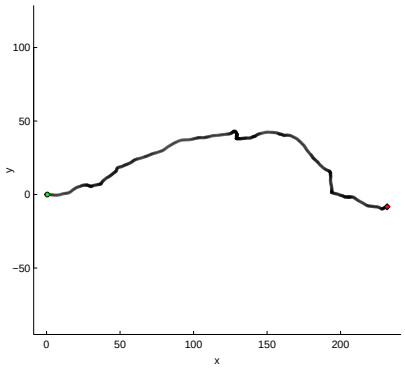
The estimation function (by default) fits the most complex RACVM model. The comparison table indicates that the most parsimonious model, according to both AIC and the more conservative BIC, is the UCVM. Note, also, that the confidence intervals around ω, μ_x and μ_y all include 0 - additional evidence that the model may not be advective or rotational. The following code estimates the "selected" model.

```
with(ucvm, estimateRACVM(Z=Z, T=T, model="UCVM", compare.models=FALSE))
```

```
## $results
##                eta      tau      rms
## Estimate 2.030165 4.984732 2.030165
## CI.low   1.567436 3.144865 1.567436
## CI.high  2.492894 7.900992 2.492894
##
## $LL
## [1] -324.7869
```

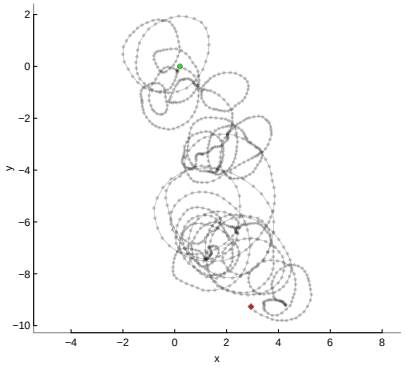
Every time this document is compiled, the results change somewhat. But in most cases the model selection should return the correct model and the estimated confidence intervals should include the true values.

4.2.2 Advective CVM



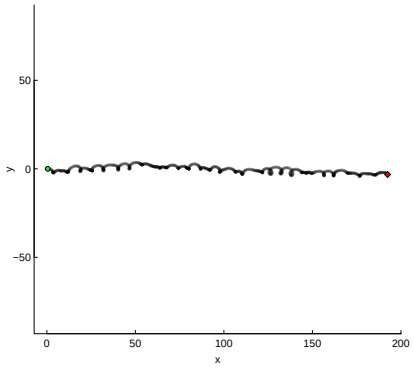

```
with(acvm, estimateRACVM(Z=Z, T=T, model = "ACVM",
                         compare.models=TRUE))

## $results
##                eta      mu.x        mu.y      tau      rms
## Estimate 1.870189 2.156809 -0.05208218 4.309585 2.896652
## CI.low   1.489074 1.377325 -0.82907917 2.853999 2.261046
## CI.high  2.251305 2.936293  0.72491482 6.507544 3.532257
##
## $LL
## [1] -303.1154
##
## $CompareTable
##          logLike k      AIC   deltaAIC      BIC deltaBIC
## UCVM  -309.3298 2 622.6595  9.2285851 632.4750 0.000000
## ACVM  -303.1154 4 614.2309  0.7999298 633.8619 1.386855
## RCVM  -308.8182 3 623.6364 10.2054900 638.3597 5.884660
## RACVM -301.7155 5 613.4310  0.0000000 637.9697 5.494681
```

4.2.3 Rotational CVM




```
with(rcvm, estimateRACVM(Z=Z, T=T, model = "RCVM",
                         compare.models=TRUE))

## $results
##                eta    omega      tau      rms
## Estimate 2.107711 1.911959 5.418521 2.107711
## CI.low   1.613949 1.824789 3.376816 1.613949
## CI.high  2.601473 1.999128 8.694689 2.601473
##
## $LL
## [1] -317.9043
##
## $CompareTable
##          logLike k       AIC     deltaAIC       BIC   deltaBIC
## UCVM  -985.7452 2 1975.4905 1333.6818179 1985.3060 1328.77406
## ACVM  -985.7236 4 1979.4471 1337.6384656 1999.0782 1342.54622
## RCVM  -317.9043 3  641.8087    0.0000000  656.5319    0.00000
## RACVM -316.3394 5  642.6787    0.8700291  667.2175   10.68554
```

4.2.4 Rotational-Advective CVM




```
with(racvm, estimateRACVM(Z=Z, T=T, model = "RACVM",
                          compare.models=TRUE))

## $results
##                eta    omega     mu.x          mu.y       tau      rms
## Estimate 2.683952 1.957792 1.914226  0.006262927  8.871261 3.304948
## CI.low   1.908777 1.876463 1.822127 -0.085867701  4.952038 2.672787
## CI.high  3.459126 2.039121 2.006324  0.098393556 15.892301 3.937109
##
## $LL
## [1] -331.6268
##
## $CompareTable
##          logLike k       AIC  deltaAIC       BIC  deltaBIC
## UCVM  -1085.4011 2 2174.8021 1501.5486 2184.6177 1486.8253
## ACVM  -1078.4676 4 2164.9353 1491.6817 2184.5663 1486.7740
## RCVM   -725.9937 3 1457.9873  784.7338 1472.7106  774.9182
## RACVM  -331.6268 5  673.2536    0.0000  697.7923    0.0000
```

Note that the results of these fits also report the root mean squared (*rms*) speed. This is a derived quantity equal to:

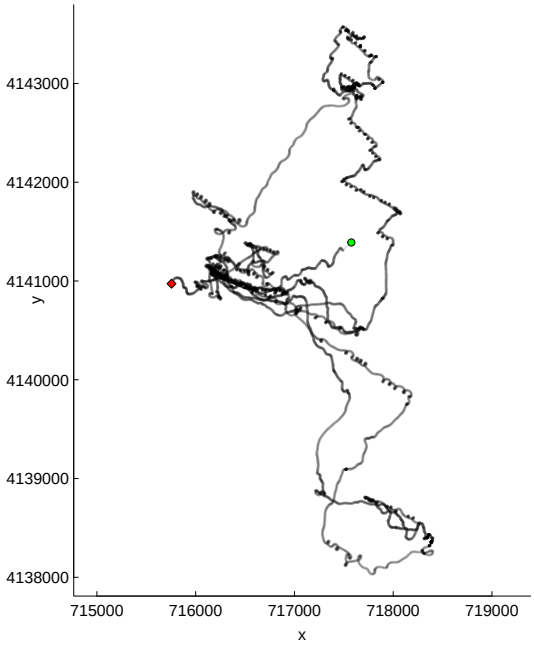
$$rms = \sqrt{\eta^2 + \mu_x^2 + \mu_y^2}$$

For unbiased CVM's, it is simple equal to η. The rms speed is useful for comparing across advective and non-advective models. There are some technical nuances to computing the confidence intervals around the rms (see Appendix).

4.3 Kestrel data

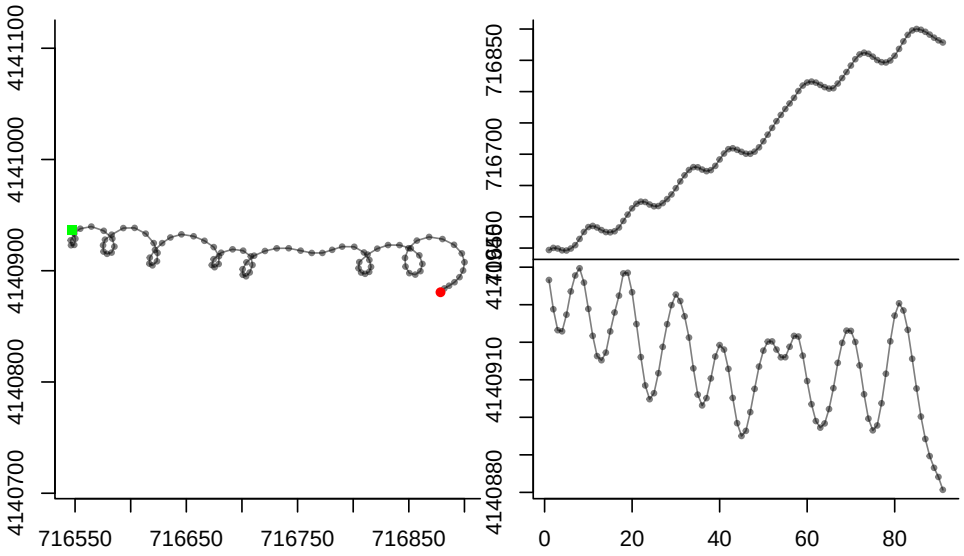
In the paper, we analyze a phenomenal data set of a single lesser kestrel (*Falco naumanni*) flight in southern Spain from Hernández-Pliego et al (2015a, 2015b). One kestrel's flight is in the package:

```
data(Kestrel); plot_track(Kestrel[,c("X","Y")], cex=0.3)
```



We can analyze one portions of the kestrel's flight, mirroring the analysis in Gurarie (in review). The following 90 second snippet is clearly advective and rotational:

```
K1 <- Kestrel[3360:3450,]
with(K1, scan_track(x=X, y=Y))
```



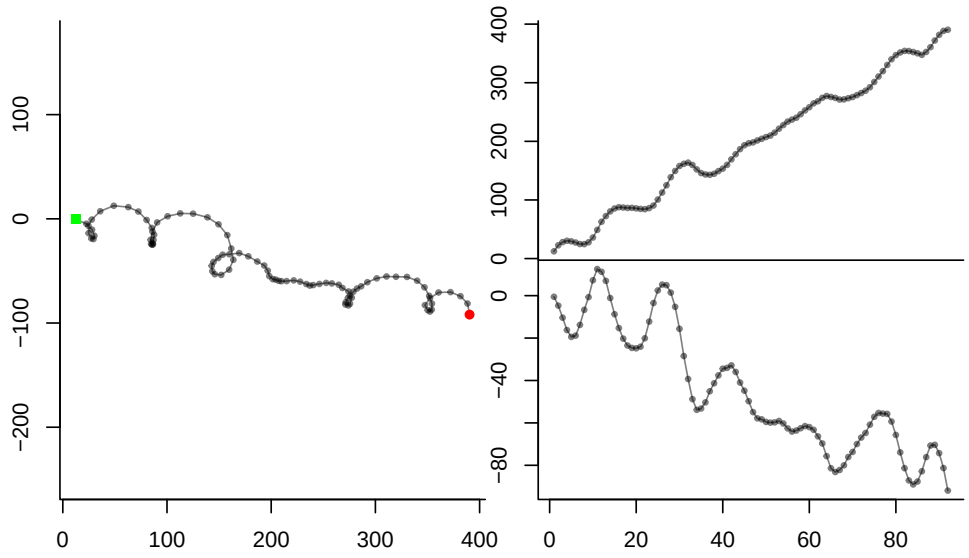
The model selection confirms this (note that we specify the time unit of analysis - in this case, seconds).

```
(fit1 <- with(K1, estimateRACVM(XY = cbind(X,Y), T = timestamp, spline=TRUE,
                                model = "RACVM", time.units = "sec")))
```

```
## $results
##                eta     omega     mu.x       mu.y        tau       rms
## Estimate  7.588318 0.5088868 3.867659 -0.3625973  31.508582  8.605233
## CI.low    3.044545 0.4702505 3.299105 -0.9311530   9.331182  4.606687
## CI.high  12.132090 0.5475230 4.436212  0.2059585 106.394962 12.603779
##
## $LL
## [1] -303.429
##
## $CompareTable
##         logLike k      AIC deltaAIC      BIC  deltaBIC
## UCVM  -439.6961 2 883.3922 266.5342 888.3918 259.03478
## ACVM  -438.7136 4 885.4271 268.5691 895.4263 266.06930
## RCVM  -357.2247 3 720.4495 103.5915 727.9489  98.59185
## RACVM -303.4290 5 616.8580   0.0000 629.3570   0.00000
```

The track is very regularly but very slightly coarsely sampled. We therefore use the spline correction (though its impact is probably minimal). The model selection strongly prefers an RACVM model. The rotational speed is about 0.5 radians per second, the advective speed is around 4 m/sec. We feed these estimates back into the simulation algorithm to simulate a trajectory:

```
p.fit1 <- with(fit1$results, list(eta=eta[1], tau = tau[1],
                                  mu = mu.x[1] + 1i*mu.y[1],
                                  omega = omega[1], v0 = diff(K1$Z)[1]))
K1.sim <- with(p.fit1, simulateRACVM(eta=eta, tau=tau, mu = mu, omega=omega,
                                     Tmax = nrow(K1), v0 = v0, dt = 1))
with(K1.sim, scan_track(z=Z))
```



Visually, the simulation looks rather similar to the data snippet with respect to radii and frequency of rotations, randomness and advective tendency.

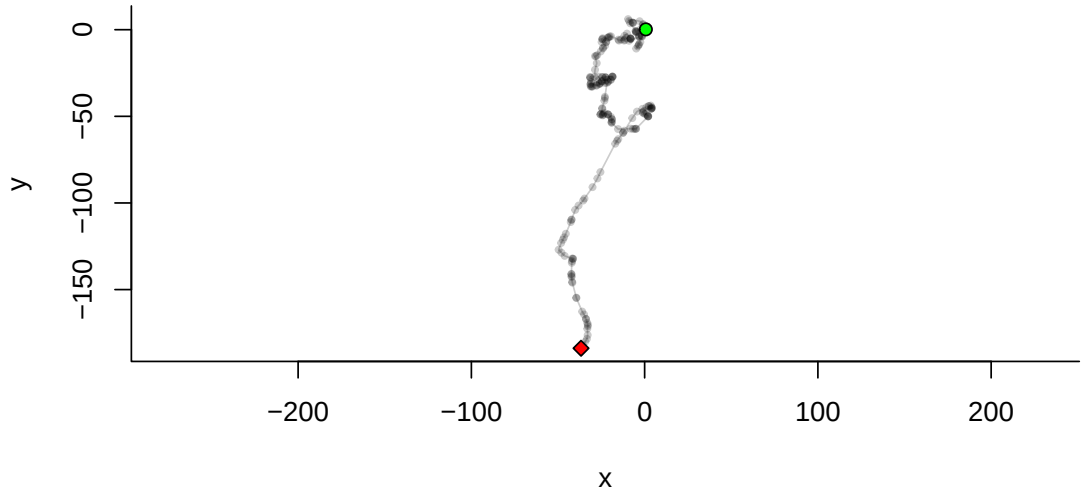
5 Behavioral change point analyses

Change point analysis (like most of ecological analysis) is not an exact science. However, the existence of likelihoods and model selection criteria makes it possible to develop an heuristic that can propose and assess candidate change points, select "significant" ones, and estimate the movement model and parameters between those change points.

5.1 Identifying a single change point

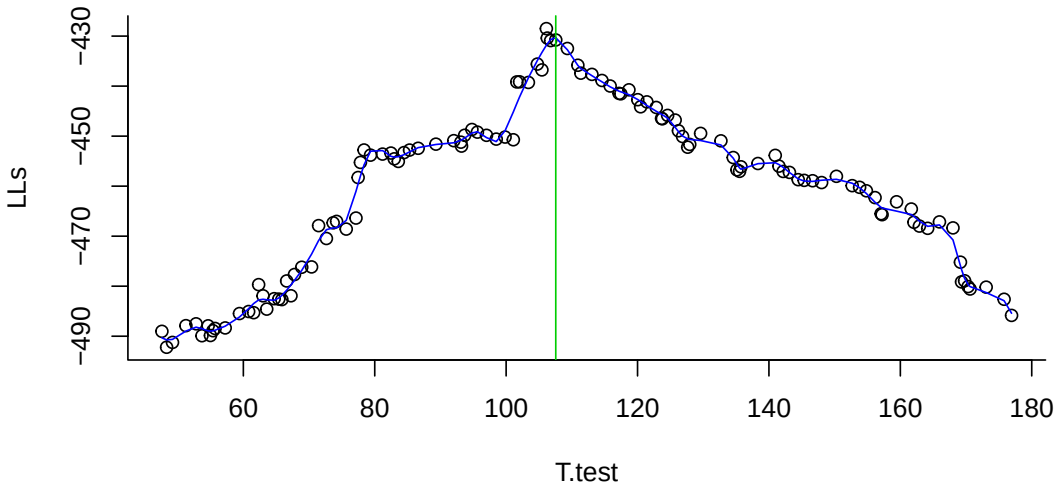
Simulate a two-phase UCVM:

```
ucvm1 <- simulateUCVM(T=cumsum(rexp(100)), nu=2, tau=1, method="exact")
ucvm2 <- simulateUCVM(T=cumsum(rexp(100)), nu=2, tau=10, v0 = ucvm1$V[100], method="exact")
T <- c(ucvm1$T, ucvm1$T[100] + ucvm2$T)
Z <- c(ucvm1$Z, ucvm1$Z[100] + ucvm2$Z)
plot_track(Z)
```



The only parameter that changed in this simulation is the time scale - it is not necessarily easy to identify by eye where the model switched. The likelihood of all candidate change points within this track is found using the `findSingleBreakPoint` as follows:

```
findSingleBreakPoint(Z,T, method = "sweep")
```




```
## [1] 107.554
```

This is a good estimate of the true change point, which occurred at time 106.3219321.

Note, this function only works on the UCVM model - i.e. it is useful for identifying sudden changes in time scales τ and rms speeds η for unbiased movements only.

5.2 Multiple change points

To find multiple change points where the number of changes is unknown *a priori* AND the underlying model changes, we follow these steps:

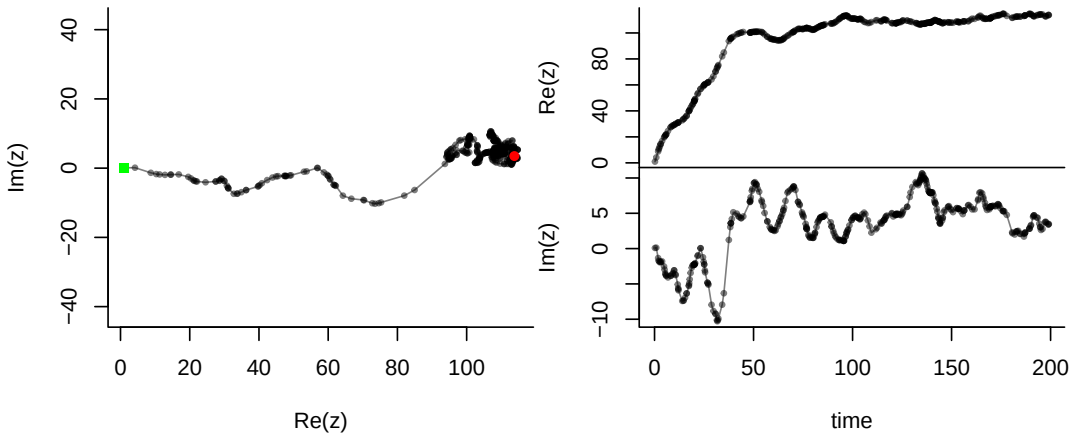
1. sweep a change point analysis window across the entire time series to obtain *candidate change points*.
2. separately assess the "significance" of each of the change points with respect to both parameters and movement model selection, iteratively removing non-significant changepoints and re-assessing
3. estimate the final models and parameter values within the final selected set of phases.

We perform an example of this analysis on a simulated trajectory with 3 behavioral phases, switching from an advective, fairly high speed movement, to an unbiased, moderately fast movement, to a highly uncorrelated slower movement. Roughly, this might approximate the shifts between commuting to a foraging site, switching to a searching behavior, and switching to an intensive foraging behavior.

```
taus <- c(3, 3, 1)
mus <- c(2, 0, 0)
etas <- c(2, 1, 1)
durations <- c(40,60,100)

Z.raw <- 0
T.raw <- 0
mycvm <- list()

for(i in 1:length(taus)){
  if(i > 1)  v0 <- mycvm$V[length(mycvm)]  else v0 = mus[1]
  mycvm <- simulateRACVM(tau = taus[i], eta = etas[i], mu = mus[i], v0 = v0,
                         Tmax = durations[i], dt = 0.01)
  Z.raw <- c(Z.raw, mycvm$Z + Z.raw[length(Z.raw)])
  T.raw <- c(T.raw, mycvm$T + T.raw[length(T.raw)])
}
```

For the analysis, we will randomly sample 400 observations from the complete data:

```
par(bty = "l", mar = c(0,4,0,0), oma = c(4,0,2,2), xpd = NA)
multicvm <- data.frame(Z = Z.raw, T = T.raw)[sample(1:length(Z.raw), 400),] %>% arrange(T)
with(multicvm, scan_track(z = Z, time = T))
```



The actual change points occur at times 40 and 100.

5.2.1 Step I - the Window Sweep

To perform a window sweep, we need to set two variables: the *windowsize* of analysis and the *windowstep*. Both are set in units of time. The window size is the temporal interval of analysis, the step is the increment by which the analysis windows are moved forward.

Setting the analysis window size is totally the user's choice. The size should be driven by biological considerations. Ideally, it will be large enough to encompass one, but not two or more change points. The smaller the window step, the more "thorough" the analysis, but ultimately its size is not very important as long as it is much smaller than the analysis window. Ultimately, for strong changes, the analysis will not be very sensitive to reasonable values of either of these parameters.

Note that if there are gaps in the data or they are irregular, the data extent of the window will be smaller, and if there are fewer than 30 data points in the window, the window will be skipped. Note also, that the window sweep is only working with a single model which should be specified. It should be the most complex model you are interested (in our case, the ACVM).

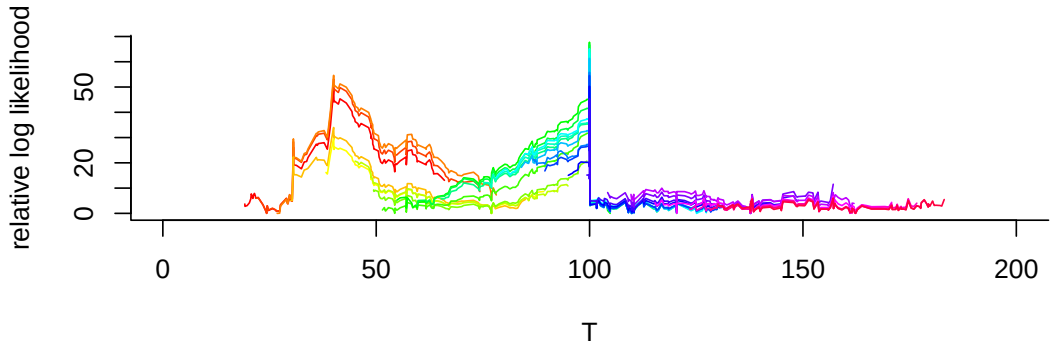
```
Z <- multicvm$Z
T <- multicvm$T
simSweep <- sweepRACVM(Z=Z, T=T, windowsize = 80, windowstep = 5, model = "ACVM", progress=FALSE)
```

Toggling the `progress` to TRUE will provide reports on the progress of the analysis. There is also an option to parallelize the analysis using the `doParallel` and `foreach` R packages. This which can speed things up considerably (roughly in proportion to the number of cores on your processor). An example of implementation:

```
require(foreach); require(doParallel)
cl <- makeCluster(detectCores())
registerDoParallel(cl)
simSweep <- sweepRACVM(Z=Z, T=T, windowsize = 80, windowstep = 5, model = "ACVM", .parallel = TRUE)
```

This function outputs a jagged matrix of likelihood estimates, which can be visualized:

```
plotWindowSweep(simSweep)
```



In the image above, each color represents the relative log-likelihood profile for a single window. There are two very distinct peaks, the remainder of the algorithm boils these down to "significant" change points.

5.2.2 Step II - Obtain Candidate Change Points

We obtain candidate change points by taking all of the most likely change points (MLCP) in each of the windows, listing all of the MLCP's. The function that does this is `findCandidateChangePoints`. Here is a complete list of candidates:

```
CP.all <- findCandidateChangePoints(windowsweep = simSweep, clusterwidth = 0)

## Note:  clustering candidate change points at 0 time units collapsed 6 candidate change
points to 6 change points.
## Warning in findCandidateChangePoints(windowsweep = simSweep, clusterwidth = 0):
##  Some of your partitions are very small - probably too small.  You might consider re-clustering
the change points with a threshold of at least 0.61.

str(CP.all)

##  atomic [1:6] 40 45.1 99.4 100 116.9 ...
##  - attr(*, "time.unit")= chr "auto"
##  - attr(*, "time")= num [1:400] 0.48 0.69 1.12 1.68 2.26 3 3.57 4.04 4.13 4.91 ...
```

The warning lets us know that some of the candidate change points are rather too close (in time) to each other, and it might be a good idea to cluster them, i.e. assign multiple points within a given time interval to a single cluster. If we take a rather generous cluster width below:

```
CP.clustered <- findCandidateChangePoints(windowsweep = simSweep, clusterwidth = 4)

## Note:  clustering candidate change points at 4 time units collapsed 6 candidate change
points to 5 change points.

str(CP.clustered)

##  atomic [1:5] 40 45.1 99.7 116.9 157.1
```

```
##  - attr(*, "time.unit")= chr "auto"
##  - attr(*, "time")= num [1:400] 0.48 0.69 1.12 1.68 2.26 3 3.57 4.04 4.13 4.91 ...
```

We now have a somewhat smaller set of change points to assess.

5.2.3 Step III - Selecting Significant Change Points

We use the set of candidate change points to separate the track into phases and for each change point (i.e. pair of neighboring phases) we assess the "significance" of the change point by comparing the BIC (or AIC) of a two-model versus one-model fit. The function that performs this is `getCPtable`

```
getCPtable(CPs = CP.clustered, Z = Z, T = T, modelset = c("UCVM", "ACVM"), tidy = NULL)
```

```
## Warning in min(i1):  no non-missing arguments to min; returning Inf
## Warning in max(i2):  no non-missing arguments to max; returning -Inf
## Error in min(i1):max(i2):  result would be too long a vector
```

In the output of `selectRACVM`, the `extremes` column indicates which parameter showed significant changes in parameter values defined by no overlap in the 95% confidence interval, and the `dAIC` and `dBIC` compare the change point model to the no change point model. We can ask this function to tidy the selection table according to any combination of several criteria: - by the differences, or by negative dAIC or dBIC, or by a mismatch in the models selected (see the help file).

```
getCPtable(CPs = CP.clustered, Z = Z, T = T, modelset = c("UCVM", "ACVM"),  iterate = TRUE)
```

```
## Warning in min(i1):  no non-missing arguments to min; returning Inf
## Warning in max(i2):  no non-missing arguments to max; returning -Inf
## Error in min(i1):max(i2):  result would be too long a vector
```

We now have a final selected set of change points and, perhaps more usefully, a summary of which parameters precisely changed. Note that thanks to the magic of `magrittr` piping, the process of finding and selecting change points can be reduced to a single line of code, which is convenient for comparing the robustness of the different settings. Replicating the above analysis for several cluster widths yield the same result:

```
simSweep %>% findCandidateChangePoints(clusterwidth = 2) %>%
  getCPtable(Z = Z, T = T, modelset = c("UCVM", "ACVM"))
```

```
## Note:  clustering candidate change points at 2 time units collapsed 6 candidate change
points to 5 change points.
## Warning in min(i1):  no non-missing arguments to min; returning Inf
## Warning in max(i2):  no non-missing arguments to max; returning -Inf
## Error in min(i1):max(i2):  result would be too long a vector
```

```
simSweep %>% findCandidateChangePoints(clusterwidth = 4) %>%
  getCPtable(Z = Z, T = T, modelset = c("UCVM", "ACVM"))
```

```
## Note:  clustering candidate change points at 4 time units collapsed 6 candidate change
points to 5 change points.
## Warning in min(i1):  no non-missing arguments to min; returning Inf
## Warning in min(i1):  no non-missing arguments to max; returning -Inf
```

```
## Error in min(i1):max(i2):  result would be too long a vector
```

```
simSweep %>% findCandidateChangePoints(clusterwidth = 10) %>%
  getCPtable(Z = Z, T = T, modelset = c("UCVM", "ACVM"))
```

```
## Note:  clustering candidate change points at 10 time units collapsed 6 candidate change
points to 4 change points.
## Warning in min(i1):  no non-missing arguments to min; returning Inf
## Warning in min(i1):  no non-missing arguments to max; returning -Inf
## Error in min(i1):max(i2):  result would be too long a vector
```

Compared to the true values (40 and 100), these are excellent estimates of the change points themselves. However, the analysis didn't pick out the "true" advective model for the initial period. This is because the model selection step (by default) uses BIC, which tend to select simpler models. If we use AIC as the criterion instead:

```
simSweep %>% findCandidateChangePoints(clusterwidth = 4) %>%
  getCPtable(Z = Z, T = T, modelset = c("UCVM", "ACVM"), criterion = "AIC")
```

```
## Note:  clustering candidate change points at 4 time units collapsed 6 candidate change
points to 5 change points.
## Warning in min(i1):  no non-missing arguments to min; returning Inf
## Warning in max(i2):  no non-missing arguments to max; returning -Inf
## Error in min(i1):max(i2):  result would be too long a vector
```

we get the correct result. We can also expand our "model set" to include all four CVM model:

```
simSweep %>% findCandidateChangePoints(clusterwidth = 10, verbose = FALSE) %>% getCPtable(Z = Z, T = T, models
```

```
## Warning in min(i1):  no non-missing arguments to min; returning Inf
## Warning in max(i2):  no non-missing arguments to max; returning -Inf
## Error in min(i1):max(i2):  result would be too long a vector
```

Which (using AIC as the criterion) suggests that the second phase might, in fact, be rotational. Note that because this analyses is selecting from multiple models (of which, in this case, only one is the correct one), it is more likely to give a "spurious" (i.e. overdetermined) analysis.

5.2.4 Step IV - estimating the full model

The final step is to estimate the parameters for the model within each of the phases defined by the selected change points, and thereby obtain a fully parameterized model of the movement. This is done with the `getPhases` function, which takes a change point table (i.e. the output of `getCPtable`]) and returns a named list of phases and the estimated parameters:

```
simCP.table <- simSweep %>%
  findCandidateChangePoints(clusterwidth = 4, verbose = FALSE) %>%
  getCPtable(Z = Z, T = T, modelset = c("UCVM", "ACVM"), criterion = "AIC")
```

```
## Warning in min(i1):  no non-missing arguments to min; returning Inf
## Warning in max(i2):  no non-missing arguments to max; returning -Inf
## Error in min(i1):max(i2):  result would be too long a vector
```

```
simPhases <- getPhases(simCP.table, Z = Z, T = T)
```

```
## Error in getPhases(simCP.table, Z = Z, T = T): could not find function "getPhases"
```

The **simPhases** object itself contains more information about each phase, e.g.:

```
simPhases[[1]]
```

```
## Error in eval(expr, envir, enclos):  object 'simPhases' not found
```

You can see that the estimates for the three parameters we set (τ, η and μ) are within the confidence intervals.
The **summarizePhases**, **getPhaseParameter** and **plotPhaseParameter** functions are useful for obtaining the estimates (with confidence intervals) and plotting their values. In the code below, we plot the track, color it by partition, and illustrate the values of the parameters.

```
  layout(c(1,1,1,2:6))

  require(gplots)
  cols <- rich.colors(length(simPhases))
```

```
## Error in rich.colors(length(simPhases)):  object 'simPhases' not found
```

```
  T.cuts <- c(T[1], simCP.table$CP, T[length(T)])
```

```
## Error in eval(expr, envir, enclos):  object 'simCP.table' not found
```

```
  Z.cols <- cols[cut(T, T.cuts, include.lowest = TRUE)]
```

```
## Error in eval(expr, envir, enclos):  object 'cols' not found
```

```
  phaseTable <- summarizePhases(simPhases)
```

```
## Error in plyr::llply(phaselist, function(a) {:  object 'simPhases' not found
```

```
  plot(Z, asp=1, type="l", xpd=FALSE)
  points(Z, col=Z.cols, pch=21, bg = alpha(Z.cols, 0.5), cex=0.8)
```

```
## Error in plot.xy(xy.coords(x, y), type = type, ...):  object 'Z.cols' not found
```

```
  legend("top", legend = paste0(phaseTable$phase, ": ", phaseTable$model),
         fill=cols, ncol=3, bty="n", title = "Phase: model")
```

```
## Error in paste0(phaseTable$phase, ":  ", phaseTable$model):  object 'phaseTable' not found
```

```
  par(mar=c(0,0,1,0), xpd=NA)
  plotPhaseParameter("tau", simPhases, ylab="", xaxt="n", xlab="", col=cols, log="y")
```

```
## Error in inherits(.data, "split"):  object 'simPhases' not found
```

```
  plotPhaseParameter("eta", simPhases,  ylab="", xaxt="n", xlab="", col=cols)
```

```
## Error in inherits(.data, "split"):  object 'simPhases' not found
```

```
  plotPhaseParameter("mu.x", simPhases,  ylab= "", xaxt="n", xlab="", col=cols)
```

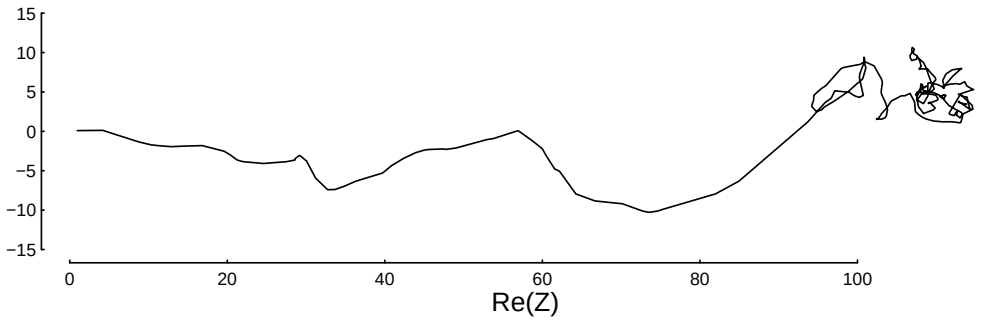
```
## Error in inherits(.data, "split"):  object 'simPhases' not found
```

```
  plotPhaseParameter("mu.y", simPhases,  ylab= "", xaxt="n", xlab="", col=cols)
```

```
## Error in inherits(.data, "split"):  object 'simPhases' not found
```

```
  plotPhaseParameter("rms", simPhases,  ylab= "", xlab="time", col=cols)
```

```
## Error in inherits(.data, "split"):  object 'simPhases' not found
```



The estimates are fairly accurate with respect to the true values:

```
data.frame(durations, taus, etas, mus)
```

```
##   durations taus etas mus
## 1        40    3    2   2
## 2        60    3    1   0
## 3       100    1    1   0
```

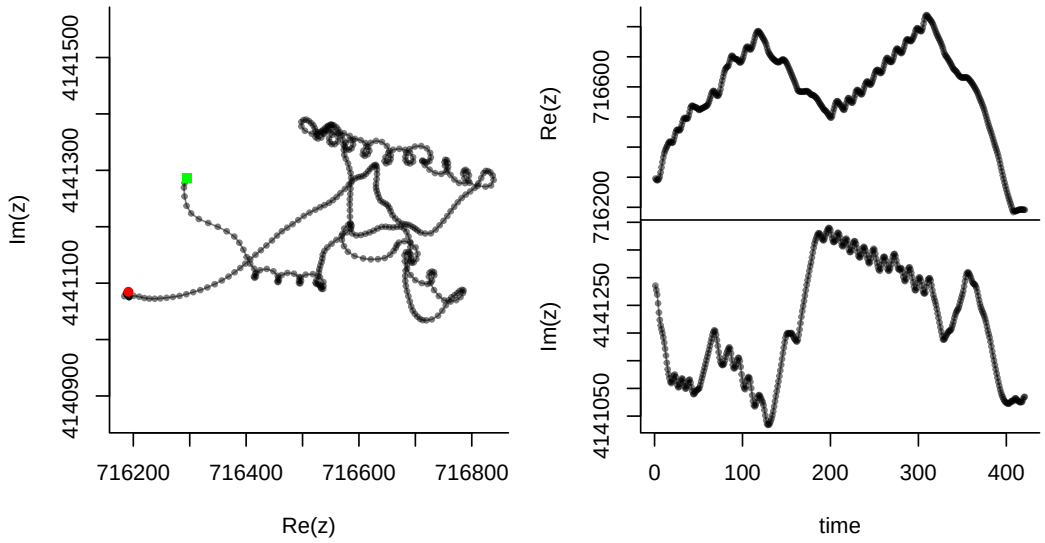
6 Kestrel data BCPA

We perform this analysis now on a portion of the kestrel data analyzed in the Gurarie et al. (in review) manuscript. The basic steps are the same as above, but the selection occurs across all four models. The portion of data we analyze is:

```
data(Kestrel)
k <- Kestrel[3730:4150,]
head(k)
```

```
##        event.id visible           timestamp longitude latitude
## 91352 274467264    true 2012-05-16 01:10:31 -6.556618 37.39312
## 91353 274467265    true 2012-05-16 01:10:32 -6.556670 37.39305
## 91354 274467266    true 2012-05-16 01:10:33 -6.556688 37.39297
## 91355 274467267    true 2012-05-16 01:10:34 -6.556680 37.39283
## 91356 274467268    true 2012-05-16 01:10:35 -6.556643 37.39268
## 91357 274467269    true 2012-05-16 01:10:36 -6.556557 37.39256
##       manually.marked.outlier sensor.type individual.taxon.canonical.name
## 91352                                 gps                  Falco naumanni
## 91353                                 gps                  Falco naumanni
## 91354                                 gps                  Falco naumanni
## 91355                                 gps                  Falco naumanni
## 91356                                 gps                  Falco naumanni
## 91357                                 gps                  Falco naumanni
##       tag.local.identifier     ID          study.name        X       Y
## 91352                  457 B[6.X] Lesser Kestrels EBD 716295.5 4141286
## 91353                  457 B[6.X] Lesser Kestrels EBD 716291.1 4141278
## 91354                  457 B[6.X] Lesser Kestrels EBD 716289.8 4141269
## 91355                  457 B[6.X] Lesser Kestrels EBD 716290.9 4141254
## 91356                  457 B[6.X] Lesser Kestrels EBD 716294.6 4141237
## 91357                  457 B[6.X] Lesser Kestrels EBD 716302.5 4141224
```

```
with(k, scan_track(x=X, y=Y))
```



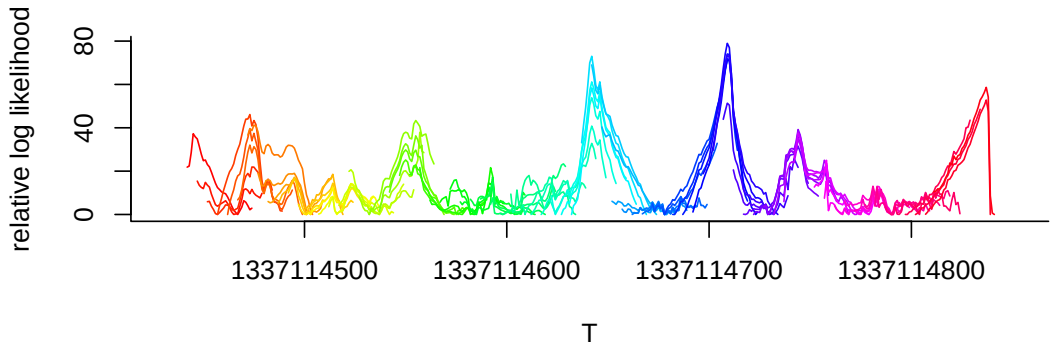
```
k$T <- as.numeric(k$timestamp - min(k$timestamp))
```

The window sweep can take some time. Note that we convert the time to seconds from start (in this case, exactly 420 seconds), which is somewhat more convenient for the analysis (though feeding POSIX objects will also work).

```
k.sweep <- with(k, sweepRACVM(XY=cbind(X,Y), T=T, windowsize = 50, windowstep = 5,
                              model = "RACVM", time.units = "sec", progress=FALSE,
                              .parallel = TRUE))
```

Note that the time data are POSIX objects, and it is important to specify the time units of the analysis (in this case, seconds). We plot the relative likelihoods:

```
plotWindowSweep(k.sweep)
```



There are several clear peaks corresponding to likely change points. We find candidate change points:

```
k.CPs <- findCandidateChangePoints(windowsweep = k.sweep, clusterwidth = 4)

## Note:  clustering candidate change points at 4 time units collapsed 38 candidate change
points to 26 change points.
```

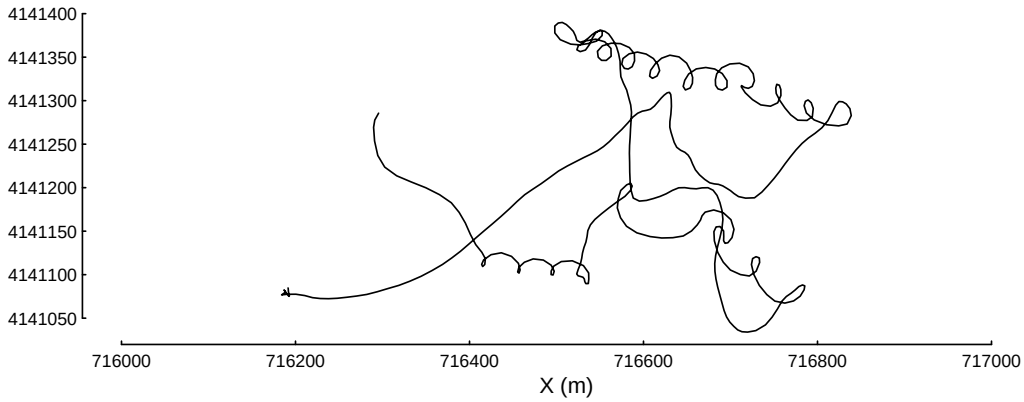
The selected cluster width (4) was the minimum one that did not produce a warning for clusters that were "too close". We now iteratively test all of the change points for parameter values and model changes. Specifying `modelset = 'all'` is a shortcut for `modelset = c('UCVM', 'ACVM', 'RCVM', 'RACVM')`:

```
XY <- cbind(k$X, k$Y)
k.CPtable <- getCPtable(CPs = k.CPs, XY = XY, T = k$T, modelset = "all", spline=TRUE)

## Warning in min(i1):  no non-missing arguments to min; returning Inf
## Warning in max(i2):  no non-missing arguments to max; returning -Inf
## Error in min(i1):max(i2):  result would be too long a vector
```

Note that in this implementation, we chose the option `spline=TRUE`, as the regularity and apparent precision of these data and the curvature of the track suggest that the spline approximation will give improved estimates of the parameters.

```
Z <- XY[,1] + 1i*XY[,2]
k.phases <- getPhases(k.CPtable, T=k$T, Z=Z, verbose=FALSE)
## Error in getPhases(k.CPtable, T = k$T, Z = Z, verbose = FALSE): could not find function
"getPhases"
summarizePhases(k.phases)
## Error in plyr::llply(phaselist, function(a) {:  object 'k.phases' not found
```

```
layout(c(1,1,1,1,2:6))

cols <- rich.colors(length(k.phases)/2)
## Error in rich.colors(length(k.phases)/2):  object 'k.phases' not found
T.cuts <- c(k$T[1], k.CPtable$CP, k$T[length(k$T)])
## Error in eval(expr, envir, enclos):  object 'k.CPtable' not found
Z.cols <- c(cols,cols)[cut(k$T, T.cuts, include.lowest = TRUE)]
## Error in eval(expr, envir, enclos):  object 'cols' not found
phaseTable <- summarizePhases(k.phases)
## Error in plyr::llply(phaselist, function(a) {:  object 'k.phases' not found
plot(Z, asp=1, type="l", xpd=FALSE, xlab="X (m)")
points(Z, col=Z.cols, pch=21, bg = alpha(Z.cols, 0.5), cex=0.8)
## Error in plot.xy(xy.coords(x, y), type = type, ...):  object 'Z.cols' not found
legend("topleft", legend = paste0(phaseTable$phase, ": ", phaseTable$model),
       fill=cols, ncol=2, bty="n", title = "Phase: model")
## Error in paste0(phaseTable$phase, ":  ", phaseTable$model):  object 'phaseTable' not found
par(mar=c(1,0,1,0), xpd=NA)
plotPhaseParameter("tau", k.phases, ylab="sec", xaxt="n", xlab="", col=cols, log="y")
## Error in inherits(.data, "split"):  object 'k.phases' not found
plotPhaseParameter("eta", k.phases,  ylab="m / sec", xaxt="n", xlab="",  col=cols)
## Error in inherits(.data, "split"):  object 'k.phases' not found
plotPhaseParameter("omega", k.phases,  ylab="rad / sec", xaxt="n", xlab="", col=cols)
## Error in inherits(.data, "split"):  object 'k.phases' not found
plotPhaseParameter("mu.x", k.phases,  ylab="m / sec", xaxt="n", xlab="", col=cols)
## Error in inherits(.data, "split"):  object 'k.phases' not found
plotPhaseParameter("mu.y", k.phases,  ylab="m / sec", xaxt="n", xlab="", col=cols)
## Error in inherits(.data, "split"):  object 'k.phases' not found
```



References

- Gurarie, E., C. Fleming, K. Laidre, W. Fagan, O. Ovaskainen (2017) Correlated velocity models as a fundamental unit of animal movement: synthesis and applications *Movement Ecology*.
- Johnson, D., J. London, M. -A. Lea, and J. Durban (2008) Continuous-time correlated random walk model for animal telemetry data. Ecology 89(5) 1208-1215.
- Hernández-Pliego, J., C. Rodríguez, J. Bustamante (2015) 'Data from: Why do kestrels soar?', Movebank Data Repository. https://www.datarepository.movebank.org/handle/10255/move.486
- Hernández-Pliego, J., C. Rodríguez, J. Bustamante (2015) 'Why do kestrels soar?', PLoS ONE 10(12).

Appendix: Properties of the rms estimate

Issues arise comparing speed parameters directly across model. In any of the advective models, the total speed is contained in the advective component $|\mu|$ and the random component η, whereas as in the unbiased (or purely rotating) models, the speed is entirely explained by η. Thus, to compare across model, the *root mean square* (rms) speed is the best measure, and we are interested in reporting and plotting point estimates and confidence intervals around the rms.

The formula for the rms is straightforward: $rms = \sqrt{\eta^2 + \mu_x^2 + \mu_y^2}$. However, each of those parameters is estimated (and therefore its own random variable) and propagating the error is a little bit tricky.

If we assume that the estimate of each of the three parameters has a normal distribution with unique means and variances, i.e. $\widehat{\eta} \sim \mathcal{N}(\mu_\eta, \sigma_\eta^2)$, etc. Then the variable $\sigma_x^2 X^2 \sim \chi^2_{n.c.}(1, \mu_x^2)$ where $\chi^2_{n.c.}(k, \lambda)$ is the non-central Chi-squared distribution with k degrees of freedom and λ non-centrality parameter (and X represents any of the speed estimates). The expectation and variance of this expression are:

$$\mathrm{E}(\sigma_x^2\, X) = \mu_x^2 + \sigma_x^2 \tag{3}$$

$$\mathrm{var}(\sigma_x^2\, X) = 2\sigma^2(\sigma^2 + 2\mu^2) \tag{4}$$

Thus, the estimate of the mean square speed (ms), which itself has a non-trivial distribution, has the following mean and variance:

$$\mathrm{E}(\widehat{ms}) = \sum_{i \in \{\mu_x, \mu_y, \eta\}} \mu_i^2 + \sigma_i^2 \tag{5}$$

$$\mathrm{var}(\widehat{ms}) = \sum_{i \in \{\mu_x, \mu_y, \eta\}} 2\sigma_i^2(\sigma_i^2 + 2\mu_i^2) \tag{6}$$

The expectation and variance of the *square root* of $\widehat{ms}$ is well-approximated by the Taylor expansion of the the square root function around the moments:

$$\mathrm{E}(\widehat{rms}) = \sqrt{\mathrm{E}(\widehat{ms})} - \frac{\mathrm{var}(\widehat{ms})}{8(\mathrm{E}(\widehat{ms}))^{3/2}} \tag{7}$$

$$\mathrm{var}(\widehat{rms}) = \frac{\mathrm{var}(\widehat{ms})}{4\mathrm{E}(\widehat{ms})} \tag{8}$$

Simulations confirm that these are excellent approximations.