Longitudinal data

Jan-Ole Koslik

In real-data applications, one will often be faced by a data set consisting of several measurement tracks, that can reasonably be assumed to be mutually independent. Examples for such a longitudinal structure include GPS tracks of several individuals (or several tracks (e.g. days) of one individual), or when analyzing sports data, one will often be faced by time series for separate games. In such settings, the researcher of course has to decide whether to pool parameters across tracks or not. Here, we will provide brief examples for complete and partial pooling.

In the situations above, the likelihood function will look slightly different. In case of K independent tracks, we have

$$L(\theta) = \prod_{k=1}^{K} L_k(\theta),$$

where $L_k(\theta)$ is the usual HMM likelihood for the k-th track. Thus the log-likelihood becomes a sum over K tracks, which we can calculate in a loop. When K is even moderately large, performing this loop in `R` already leads to severe slowdowns in likelihood evaluation times. Thus, the forward algorithms in `LaMa` allow for the likelihood formulation above, when the indices at which separate tracks begin are specified. Here, we shortly demonstrate how to use this option.

Complete pooling

Generating data

We generate K separate tracks, all from the exact same model:

```
K = 100 # number of individuals, for example different animals

# parameters are shared across individuals
mu = c(15, 60)
sigma = c(10, 40)
Gamma = matrix(c(0.95, 0.05, 0.15, 0.85), nrow = 2, byrow = TRUE)
delta = stationary(Gamma) # stationary HMM

# simulation of all tracks
set.seed(123)
n = 30 # observations per animal only (but many animals)
s = x = rep(NA, n*K)
for(k in 1:K){
  sk = xk = rep(NA, n)
  sk[1] = sample(1:2, 1, prob = delta)
  xk[1] = rnorm(1, mu[sk[1]], sigma[sk[1]])
  for(t in 2:n){
    sk[t] = sample(1:2, 1, prob = Gamma[sk[t-1],])
    xk[t] = rnorm(1, mu[sk[t]], sigma[sk[t]])
  }
```

```
  s[(k-1)*n + 1:n] = sk
  x[(k-1)*n + 1:n] = xk
}
```

Writing the negative log-likelihood function

To calculate the joint log-likelihood of the independent tracks, we slightly modify the standard negative log-likelihood function by adding the additional argument `trackInd`. This is a vector containing all the indices at which a new track begins. `forward()` now calculates the sum of indivual likelihood contributions, each starting in the respective initial distribution (which we pool here).

```
# fast version using trackInd in forward()
mllk_pool = function(theta.star, x, trackInd){
  Gamma = tpm(theta.star[1:2])
  delta = stationary(Gamma)
  mu = theta.star[3:4]
  sigma = exp(theta.star[5:6])
  allprobs = matrix(1, length(x), 2)
  for(j in 1:2){ allprobs[,j] = dnorm(x, mu[j], sigma[j]) }

  # here we add trackInd as an argument to forward()
  -forward(delta, Gamma, allprobs, trackInd)
}

# slow alternative looping over individuals in R
mllk_pool_slow = function(theta.star, x, K){
  n = length(x)/K
  Gamma = tpm(theta.star[1:2])
  delta = stationary(Gamma)
  mu = theta.star[3:4]
  sigma = exp(theta.star[5:6])
  allprobs = matrix(1, length(x), 2)
  for(j in 1:2){ allprobs[,j] = dnorm(x, mu[j], sigma[j]) }

  # here we just loop over individuals in R
  l = 0
  for(k in 1:K){
    l = l + forward(delta, Gamma, allprobs[(k-1)*n + 1:n,])
  }
  -l
}
```

Estimating the model

Now we estimate the model with complete pooling. We compare the fast version using `forward()` with `trackInd` with the slow version also using `forward()` but looping over individuals in `R`. Before we begin, we define the indices where a new track starts:

```
# defining the indices where a new track begins
trackInd = n*(0:(K-1)) + 1

# In a real data scenario, we would typically have an ID column in the data set.
```

```
# We can then use the function calc_trackInd() to calculate the trackInd vector:
ID = rep(1:K, each = n)
trackInd = calc_trackInd(ID)
# the ID variable does not have to be numeric, it can also be a character vector.

# initial parameter vector
theta.star = c(-1,-1, # off-diagonals of Gamma (on logit scale)
               15,60,log(10),log(40)) # mu and sigma

# fast version:
s = Sys.time()
mod = nlm(mllk_pool, theta.star, x = x, trackInd = trackInd)
Sys.time()-s
#> Time difference of 0.04609299 secs

# slow version
s = Sys.time()
mod = nlm(mllk_pool_slow, theta.star, x = x, K = K)
Sys.time()-s
#> Time difference of 0.107379 secs
```

In this example, looping over individuals in R doubles the evaluation time, but this can be much more severe for more complicated models.

Partial pooling

If some parameters of our model are individual-specific, while the rest is shared, we speak of partial pooling. We demonstrate this here for 5 individuals with their own transition probability matrices. We could estimate a separate transition probability matrix for each individual, but here we opt for a more parsimonious approach, where the transition probabilities depend on an external, individual-specific covariate. We will estimate the effect of this covariate on the transition probabilities.

Generating data

```
K = 5 # number of individuals, for example different animals

# state-dependent parameters are shared across individuals
mu = c(15, 60)
sigma = c(10, 40)

# but we define a tpm for each individual depending on covariates
set.seed(123)
z = rnorm(K) # covariate (e.g. age)
beta = matrix(c(-2,-2, 1, -1), nrow = 2)
# we calculate 5 tpms depending on individual-specific covariates:
Gamma = tpm_g(z, beta)
# each individual starts in its stationary distribution:
Delta = matrix(NA, K, 2)
for(k in 1:K){ Delta[k,] = stationary(Gamma[,,k]) }

# simulation of all tracks
```

```
set.seed(123)
n = 200 # observations per animal only (but many animals)
s = x = rep(NA, n*K)
for(k in 1:K){
  sk = xk = rep(NA, n)
  sk[1] = sample(1:2, 1, prob = Delta[k, ])
  xk[1] = rnorm(1, mu[sk[1]], sigma[sk[1]])
  for(t in 2:n){
    sk[t] = sample(1:2, 1, prob = Gamma[sk[t-1],,k])
    xk[t] = rnorm(1, mu[sk[t]], sigma[sk[t]])
  }
  s[(k-1)*n + 1:n] = sk
  x[(k-1)*n + 1:n] = xk
}
```

Writing the negative log-likelihood function

Now we write the corresponding negative log-likehood function that incorporates the above structure:

```
# fast version using trackInd in forward()
mllk_partial = function(theta.star, x, z, trackInd){
  # individual-specific tpms
  beta = matrix(theta.star[1:4], nrow = 2)
  Gamma = tpm_g(z, beta)
  Delta = matrix(NA, length(z), 2)
  for(k in 1:length(z)) Delta[k,] = stationary(Gamma[,,k])
  mu = theta.star[5:6]
  sigma = exp(theta.star[7:8])
  allprobs = matrix(1, length(x), 2)
  for(j in 1:2){ allprobs[,j] = dnorm(x, mu[j], sigma[j]) }

  # just handing a Delta matrix and Gamma array for all individuals to forward()
  -forward(Delta, Gamma, allprobs, trackInd)
}
```

Estimating the model

```
# again defining all the indices where a new track begins
ID = rep(1:K, each = n)
trackInd = calc_trackInd(ID)

# initial parameter vector
theta.star = c(-2,-2,0,0, # beta
               15,60,log(10),log(40)) # mu and sigma

s = Sys.time()
mod_partial = nlm(mllk_partial, theta.star, x = x, z = z, trackInd = trackInd)
Sys.time()-s
#> Time difference of 0.05137396 secs
```