20236 Time Series Analysis: Lab 2

Sonia Petrone

This LAB is an introduction to Hidden Markov Models (HMMs) with R.

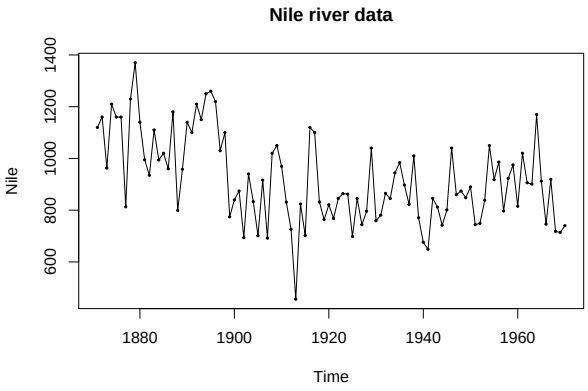
The first question is: What is available in R for HMMs? Explore that yourself! You'll find several packages, including packages "depmixS4" and "HiddenMarkov". Here we use package "depmixS4" . A reference is: Visser, I. and Speekenbrink, M. (2010) "depmixS4: An R Package for Hidden Markov Models", *Journal of Statistical Software*, 36.

Load packages depmixS4 in the Rchunk at the beginning of the file.Rmd. (you may also want to install and load the auxiliary packages tidyverse, ggplott2, magritte,... for fancier stuff).

EXAMPLE 1 : Nile river data

Let's look again at the Nile river data

```
plot(Nile, type="o", pch=19, cex=.3, main="Nile river data")
```



We could think of the simplest model for these data

$$Y_t = \mu + \epsilon_t, \quad \epsilon_t \overset{iid}{\sim} N(0, \sigma^2)$$

However, the data clearly show a change point in the river's level: a model that allows μ to change over time is more appropriate. We fit a HMM for the Nile river data, using package depmixS4.

```
?depmixS4
```

In the **Details** in the help page, we read: "Models are specified through the depmix function, which uses standard *glm* style arguments to specify the observed distributions"

Thus, let us first give some preliminaries: R functions **lm** (linear models) and **glm** (generalized linear models)

```
?lm
# example:
x <- c(20, 12, 30, 60, 39, 90, 45, 10, 120, 150)
```

```
y <- 2 * x + rnorm(10)
out <- lm(y ~ x)
out
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Coefficients:
## (Intercept)            x
##     -0.4036       2.0031
```

```
# to remove the intercept
out <- lm(y ~ x - 1)
# only the intercept
out <- lm(y ~ 1)
?glm
```

We want to specify a Gaussian HMM for the Nile river data, where

$$\begin{cases} Y_t = \mu_1 + \epsilon_t, & \epsilon_t \overset{iid}{\sim} N(0, \sigma_1^2) \quad \text{if the state } S_t = 1 \\ Y_t = \mu_2 + \epsilon_t, & \epsilon_t \overset{iid}{\sim} N(0, \sigma_2^2) \quad \text{if the state } S_t = 2. \end{cases}$$

Thus, the *lm* will only have the intercept.

Let's now fit the above HMM for the Nile data.

1. **STEP 1: specify the model.**

In package depmix, the model is specified by the function

```
?depmix
```

The data should be provided as a data frame.

```
is.ts(Nile)
```

```
## [1] TRUE
```

```
y <- as.numeric(Nile)
model <- depmix(y ~ 1, data=data.frame(y), nstates=2)
model
```

```
## Initial state probabilities model
## pr1 pr2
## 0.5 0.5
##
## Transition matrix
##        toS1 toS2
## fromS1  0.5  0.5
## fromS2  0.5  0.5
##
## Response parameters
## Resp 1 : gaussian
##     Re1.(Intercept) Re1.sd
## St1               0      1
## St2               0      1
```

Notice, in the model so specified, the choice of the starting values of the unknown parameters, namely of

- the initial probabilities for S_0;
- the transition matrix;
- the parameters $\theta_1 = (\mu_1, \sigma_1)$; $\theta_2 = (\mu_2, \sigma_2)$.

2. **STEP 2: Fit the model, computing the MLEs of the unknown parameters.**

```
fmodel <- fit(model)
```

```
## converged at iteration 23 with logLik: -629.8045
```

```
fmodel # logLik and optimization information
```

```
## Convergence info: Log likelihood converged to within tol. (relative change)
## 'log Lik.' -629.8045 (df=7)
## AIC:  1273.609
## BIC:  1291.845
```

```
summary(fmodel) # MLEs of the unknown parameters.
```

```
## Initial state probabilities model
## pr1 pr2
##   1   0
##
## Transition matrix
##         toS1  toS2
## fromS1 0.964 0.036
## fromS2 0.000 1.000
##
## Response parameters
## Resp 1 : gaussian
##     Re1.(Intercept)  Re1.sd
## St1        1097.153 133.748
## St2         850.757 124.446
```

You may extract, for instance, the MLE for the mean and standard deviation corresponding to the second hidden state, as follows

```
fmodel@response[[2]][[1]]@parameters$coefficients
```

```
## (Intercept)
##    850.7565
```

```
fmodel@response[[2]][[1]]@parameters$sd
```

```
##       sd
## 124.4464
```

Remark. Whenever we provide an estimate, we should also provide the associated standard error.
Note that standard errors were not included in the original release of package depMixS4. See the paper by Visser and Speekenbrink (Journal of Statistical Software, 2010), posted on BBoard. And for example read at `https://stat.ethz.ch/pipermail/r-packages/2019/001651.html` (2019): *"The new 1.4 version of depmixS4 has an important (and much requested!) new feature: the possibility to request standard errors of estimated parameters through the use of a finite differences approximation of the hessian. As this is a critical feature we appreciate your comments and feedback..[....]"*

```
MLEse=standardError(fmodel)
```

What are the MLEs?

```
# str(MLEse)
MLEse$par
```

What are their standard errors? Below are the standard errors of the MLEs of the parameters in the emission distribution in the two states:

```
round(MLEse$par, 3)
```

```
##  [1]    1.000    0.000    0.964    0.036    0.000    1.000 1097.153  133.748
##  [9]  850.757  124.446
```

```
round(MLEse$se[7:10], 3)
```

```
## [1] 25.557 18.194 14.749 10.424
```

3. **STEP 3: decoding**

```
? posterior
# Get the estimated state for each timestep
estStates <- posterior(fmodel)
```

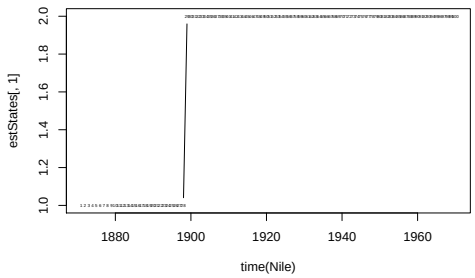
The first column of the output has the Viterbi states, the other columns have the delta probabilities (see Rabiner, 1989) :

```
estStates[1:5,]
```

```
##   state        S1          S2
## 1     1 1.0000000 0.000000000
## 2     1 0.9979643 0.002035736
## 3     1 0.9577703 0.042229730
## 4     1 0.9989133 0.001086733
## 5     1 0.9979643 0.002035736
```

Let us plot the data and the estimated "most likely states"

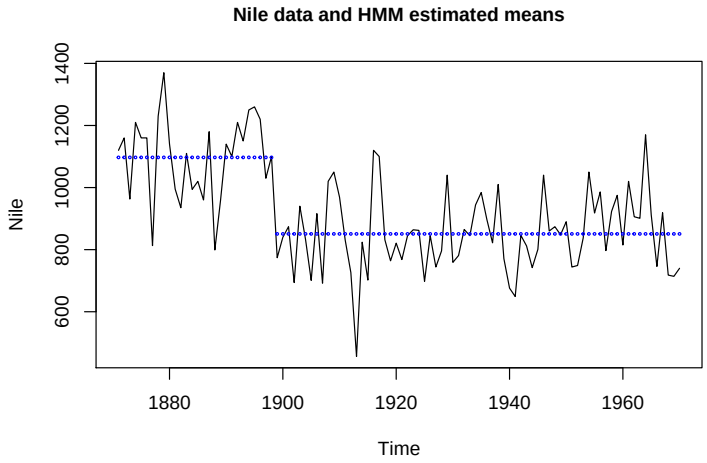
```
plot(time(Nile), estStates[,1],  cex=.3)
```



A simple plot of the data with the estimated state-dependent means

```
i= estStates[1,1]
ii= if(i==1){i+1} else {i-1}
estMean1=fmodel@response[[i]][[1]]@parameters$coefficients
estMean2=fmodel@response[[ii]][[1]]@parameters$coefficients
```

```
estMeans=rep(estMean1, length(Nile))
estMeans[estStates[,1]==ii]=estMean2
plot(Nile)
title(main="Nile data and HMM estimated means", cex.main=1)
points(time(Nile), estMeans,  col="blue", cex=.3)
```

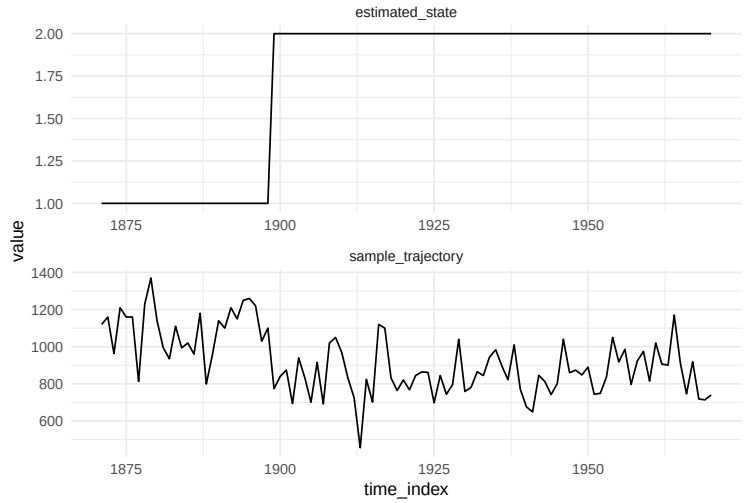


For other fancier plots (require additional packages)

```
results_df <- data.frame(time_index=as.numeric(time(Nile)),
                    sample_trajectory=y,
                    estimated_state=estStates[,1])%>%
                    gather("variable", "value", -time_index)

plotobj <- ggplot(results_df, aes(time_index, value)) +
  geom_line() + facet_wrap(variable ~ ., scales="free", ncol=1) +
  theme_minimal()

plot(plotobj)
```



EXAMPLE 2: Poisson HMM.

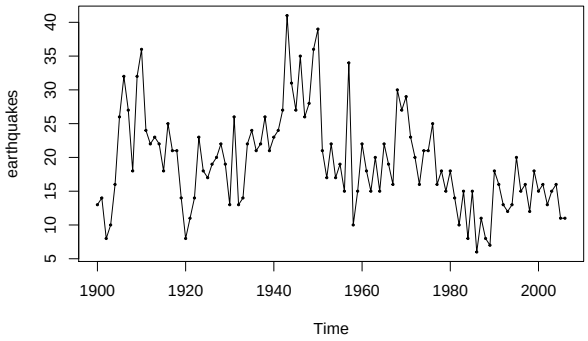
This is an example from

Zucchini, Mac Donald, Langrock (2016) *Hidden Markov Models for Time Series: An introduction using R*, CRC Press.

We study the series of annual counts of major earthquakes (i.e. magnitude 7 and above), from 1900 to 2006.

```
earthquakes <- read.table("http://www.hmms-for-time-series.de/second/data/earthquakes.txt")

earthquakes <- ts(earthquakes[,2], start=1900)
plot(earthquakes, type="o", pch=19, cex=.3)
```



When dealing with unbounded counts, a possible choice is a Poisson distribution. Remember

$$\text{if } Y \sim Poisson(\lambda), \quad \text{then } E(Y) = V(Y) = \lambda.$$

However, in the timeplot of the data, we notice periods when earthquakes are relatively more frequent than in other periods, suggesting that there may be different values of the intersity λ, corresponding to different latent states.

We can also infer from the sample mean and variance that the series displays considerable overdispersion relative to the Poisson distribution; therefore, the Poisson distribution (with a constant λ) is not appropriate to model the earthquake data.

```
mean(earthquakes)
```

```
## [1] 19.36449
```

```
var(earthquakes)
```

```
## [1] 51.57344
```

Let's set up a Poisson HMM with 2 latent states. We need to specify *family = poisson()*

```
mod.phmm <- depmix(earthquakes ~ 1, nstates = 2, ntimes=107, family = poisson())

# Take a look at the default initialization
mod.phmm
```

```
## Initial state probabilities model
## pr1 pr2
## 0.5 0.5
##
```

```
## Transition matrix
##        toS1 toS2
## fromS1  0.5  0.5
## fromS2  0.5  0.5
##
## Response parameters
## Resp 1 : poisson
##     Re1.(Intercept)
## St1               0
## St2               0
```

```
# Fit the model, which is done by EM algorithm
f.phmm <- fit(mod.phmm)
```

```
## converged at iteration 25 with logLik: -341.8787
```

```
# The estimated parameters
summary(f.phmm)
```

```
## Initial state probabilities model
## pr1 pr2
##   1   0
##
## Transition matrix
##         toS1  toS2
## fromS1 0.928 0.072
## fromS2 0.119 0.881
##
## Response parameters
## Resp 1 : poisson
##     Re1.(Intercept)
## St1           2.736
## St2           3.259
```

```
# one state  has a smaller Poisson parameter compared to the other state,
# correponding to a state with lower rate of earthquakes
```

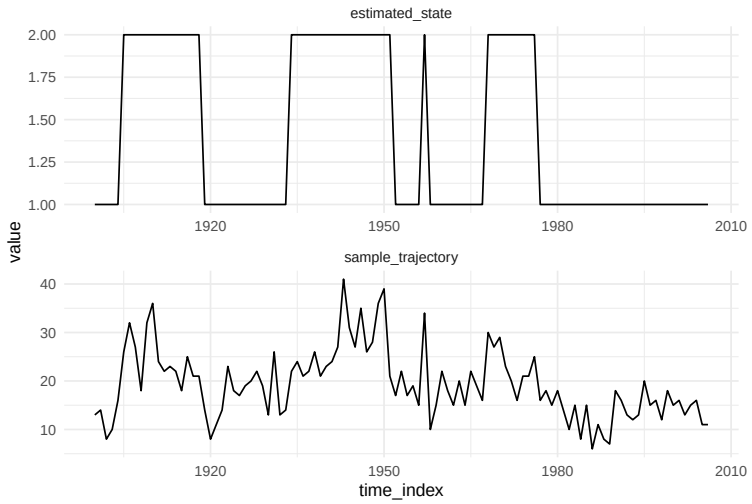
Let's now turn to deconding (the most likely state and posterior probabilities of each state at each year)

```
estStates=posterior(f.phmm)
```

To see the relationship between observations and most likely states in a clearer way, let's plot them together

```
results_df <- data.frame(time_index=time(earthquakes) %>% as.numeric(),
sample_trajectory=earthquakes %>% as.numeric(), estimated_state=posterior(f.phmm)$state) %>%
  gather("variable", "value", -time_index)

ggplot(results_df, aes(time_index, value)) + geom_line() +
  facet_wrap(variable ~ ., scales="free", ncol=1) + theme_minimal()
```



What about a HMM with 3 states?

```
mod.phmm <- depmix(earthquakes ~ 1, nstates = 3, ntimes=107, family = poisson())
mod.phmm
```

```
## Initial state probabilities model
##   pr1   pr2   pr3
## 0.333 0.333 0.333
##
## Transition matrix
##         toS1  toS2  toS3
## fromS1 0.333 0.333 0.333
## fromS2 0.333 0.333 0.333
## fromS3 0.333 0.333 0.333
##
## Response parameters
## Resp 1 : poisson
##     Re1.(Intercept)
## St1               0
## St2               0
## St3               0
```

```
f.phmm <- fit(mod.phmm)
```

```
## converged at iteration 26 with logLik: -328.5275
```

What is now the state underlying the highest Poisson intensity?

```
summary(f.phmm)
```

```
## Initial state probabilities model
## pr1 pr2 pr3
##   0   1   0
##
## Transition matrix
##         toS1  toS2  toS3
## fromS1 0.906 0.040 0.053
## fromS2 0.032 0.939 0.029
## fromS3 0.190 0.000 0.810
##
```

```
## Response parameters
## Resp 1 : poisson
##     Re1.(Intercept)
## St1           2.981
## St2           2.575
## St3           3.392
```

```
results_df <- data.frame(time_index=time(earthquakes) %>% as.numeric(),
sample_trajectory=earthquakes %>% as.numeric(),
estimated_state=posterior(f.phmm)$state) %>%
  gather("variable", "value", -time_index)

ggplot(results_df, aes(time_index, value)) + geom_line() +
  facet_wrap(variable ~ ., scales="free", ncol=1) + theme_minimal()
```