



fHMM: Hidden Markov Models for Financial Time Series in R

Lennart Oelschläger
Bielefeld University

Timo Adam
University of St Andrews

Rouven Michels
Bielefeld University

Abstract

Hidden Markov models constitute a popular class of statistical models for time series that are driven by hidden states. In finance, these hidden states can often be linked to market regimes such as bearish and bullish markets or recessions and periods of economics growths, to name but a few examples. Hidden Markov models account for these state-switching patterns by... In this paper, we introduce the R package **fHMM** that provides various tools for modeling financial time series using hidden Markov models. Its key features include...

Keywords: decoding market behavior, hidden Markov models, time series modeling, R.

1. Introduction

In recent years, hidden Markov models (HMMs) have emerged as a popular tool for modeling time series that are subject to state-switching over time (Zucchini, MacDonald, and Langrock 2016). In their basic form, HMMs comprise an observed state-dependent process that is driven by a hidden state process, the latter of which is typically modeled using a discrete-time, finite-state Markov chain. In financial applications, the states of the underlying Markov chain can often be linked to market regimes such as bearish and bullish markets or recessions and periods of economics growths: when the market is in a calm state, then the observations follow some distribution, whereas when the market is in a nervous state, then another distribution is active to generate the observations. By their dependence structure, HMMs naturally account for such state-switching patterns and thus allow us to infer hidden market regimes and their dynamics from financial time series.

Over the last decades, HMMs have been applied to various financial problems, including modeling business cycles Gregoir and Lengart (2000), deriving stylized facts of stock returns Bulla and Bulla (2006); Nystrup, Madsen, and Lindström (2015), and modeling energy prices con-

ditional on market regimes Langrock, Adam, Leos-Barajas, Mews, Miller, and Papastamatiou (2018); Adam, Mayr, and Kneib (2022), to name but a few examples. More recently, Lihn (2017) used HMMs to identify different volatility levels in the Standard and Poor’s 500 index, aiming at providing evidence for the conjecture that returns exhibit negative correlation with volatility. Nguyen (2018), to give another example, used HMMs to predict monthly closing prices to derive an optimal trading strategy, which was shown to outperform the conventional buy-and-hold strategy. Further financial applications include HMMs for asset allocation and portfolio optimization (Bulla, Mergner, Bulla, Sesboüe, and Chesneau 2011; Nystrup *et al.* 2015; Nystrup, Madsen, and Lindström 2018). All these examples demonstrate that HMMs constitute a versatile class of time series models that naturally accounts for the state-switching patterns that are often found in financial time series.

The **fHMM** package Oelschläger, Adam, and Michels (2022) aims at making HMMs accessible to users interested in modeling financial time series. Its functionality can be classified into functions for data preparation, model estimation, and model evaluation, which is illustrated in Figure 1. Functions for data preparation include an interface to Yahoo Finance for downloading stock data. The model is estimated in a maximum likelihood framework, where the likelihood is evaluated using the forward algorithm, which is implemented in C++ and parallelized for fast computation. Functions for model evaluation include... It also implements hierarchical HMMs; an extension of basic HMMs proposed in Oelschläger and Adam (2021) that improves the model’s capability of distinguishing between short- and long-term trends and allows us to model data collected at different time scales. interpret market dynamics at multiple time scales.

This paper is structured as follows: in Section 2, ... In Section 3, ...

2. Model definition

Hidden Markov models (HMMs) are a modeling framework for time series data where a sequence of observation is assumed to depend on a latent state process. The peculiarity is that, instead of the observation process, the state process cannot be directly observed. However, the latent states comprise information about the environment the model is applied on.

The connection between hidden state process and observed state-dependent process arises by the following: Let N be the number of possible states. We assume that for each point in time $t = 1, \dots, T$, an underlying process $(S_t)_{t=1, \dots, T}$ is in one of those N states. Then, depending on the active state $S_t \in \{1, \dots, N\}$, the observation X_t from the state-dependent process $(X_t)_{t=1, \dots, T}$ is assumed to be generated by the corresponding distribution of the N distributions $f^{(1)}, \dots, f^{(N)}$.

Furthermore, we assume $(S_t)_t$ to be Markovian, i.e. we assume that the actual state only depends on the previous state. Henceforth, we can identify the process by its initial distribution δ and its transition probability matrix (t.p.m.) Γ . Moreover, by construction, we force the process $(X_t)_{t=1, \dots, T}$ to satisfy the conditional independence assumption, i.e. the actual observation X_t depends on the current state S_t , but does not depend on previous observations or states at all.

Referring to financial data, the different states can serve as proxies for the actual market situation, e.g. calm or nervous. Even though these moods cannot be observed directly, price

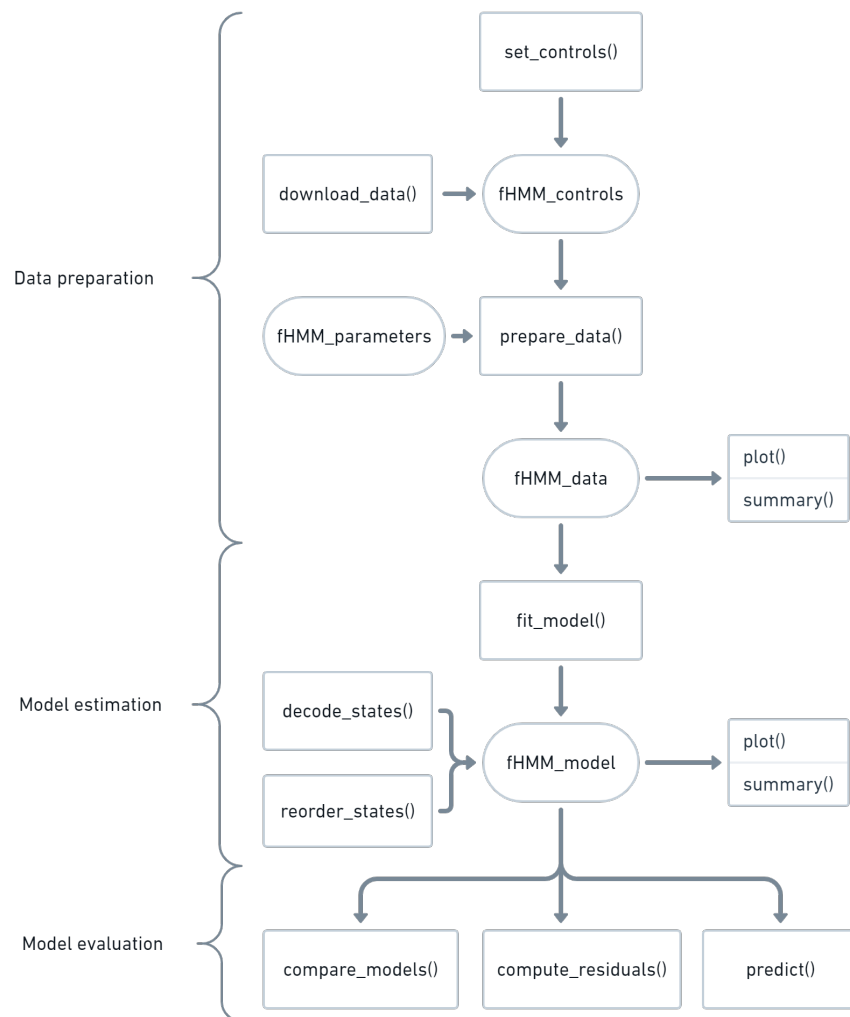


Figure 1: Flowchart of the package functionality.

changes or trading volumes, which clearly depend on the current mood of the market, can be observed. Thereby, using an underlying Markov process, we can detect which mood is active at any point in time and how the different moods alternate. Depending on the current mood, a price change is generated by a different distribution. These distributions characterize the moods in terms of expected return and volatility. For example, we can model price changes at time point t to be generated by different normal distributions whose mean and volatility depend on S_t .

Following [Zucchini *et al.* \(2016\)](#), we assume that the initial distribution δ equals the stationary distribution π , where $\pi = \pi\Gamma$, i.e. the stationary and henceforth the initial distribution is determined by Γ . If the Markov process is irreducible, it has a unique distribution, which solves $\pi = \pi\Gamma$. If additionally the Markov process is aperiodic, its state distribution converges to the stationary distribution, see [Norris \(1997\)](#). Irreducibility and aperiodicity are usually satisfied assumptions in reality. This is reasonable from a practical point of view: On the one hand, the hidden state process has been evolving for some time before we start to observe it and hence can be assumed to be stationary. On the other hand, setting $\delta = \pi$ reduces the number of parameters that need to be estimated, which is convenient from a computational perspective.

The hierarchical hidden Markov model (HMMM) is a flexible extension of the HMM that can jointly model data observed on two different time scales. The two time series, one on a coarser and one on a finer scale, differ in the number of observations, e.g. monthly observations on the coarser scale and daily or weekly observations on the finer scale.

Following the concept of HMMs, we can model both state-dependent time series jointly. First, we treat the time series on the coarser scale as stemming from an ordinary HMM, which we refer to as the coarse-scale HMM: At each time point t of the coarse-scale time space $\{1, \dots, T\}$, an underlying process $(S_t)_t$ selects one state from the coarse-scale state space $\{1, \dots, N\}$. We call $(S_t)_t$ the hidden coarse-scale state process. Depending on which state is active at t , one of N distributions $f^{(1)}, \dots, f^{(N)}$ realizes the observation X_t . The process $(X_t)_t$ is called the observed coarse-scale state-dependent process. The processes $(S_t)_t$ and $(X_t)_t$ have the same properties as before, namely $(S_t)_t$ is a first-order Markov process and $(X_t)_t$ satisfies the conditional independence assumption.

Subsequently, we segment the observations of the fine-scale time series into T distinct chunks, each of which contains all data points that correspond to the t -th coarse-scale time point. Assuming that we have T^* fine-scale observations on every coarse-scale time point, we face T chunks comprising of T^* fine-scale observations each.

The hierarchical structure now evinces itself as we model each of the chunks by one of N possible fine-scale HMMs. Each of the fine-scale HMMs has its own t.p.m. $\Gamma^{*(i)}$, initial distribution $\delta^{*(i)}$, stationary distribution $\pi^{*(i)}$, and state-dependent distributions $f^{*(i,1)}, \dots, f^{*(i,N^*)}$. Which fine-scale HMM is selected to explain the t -th chunk of fine-scale observations depends on the hidden coarse-scale state S_t . The i -th fine-scale HMM explaining the t -th chunk of fine-scale observations consists of the following two stochastic processes: At each time point t^* of the fine-scale time space $\{1, \dots, T^*\}$, the process $(S_{t,t^*}^*)_{t^*}$ selects one state from the fine-scale state space $\{1, \dots, N^*\}$. We call $(S_{t,t^*}^*)_{t^*}$ the hidden fine-scale state process. Depending on which state is active at t^* , one of N^* distributions $f^{*(i,1)}, \dots, f^{*(i,N^*)}$ realizes the observation X_{t,t^*}^* . The process $(X_{t,t^*}^*)_{t^*}$ is called the observed fine-scale state-dependent process.

The fine-scale processes $(S_{1,t^*}^*)_{t^*}, \dots, (S_{T,t^*}^*)_{t^*}$ and $(X_{1,t^*}^*)_{t^*}, \dots, (X_{T,t^*}^*)_{t^*}$ satisfy the Markov

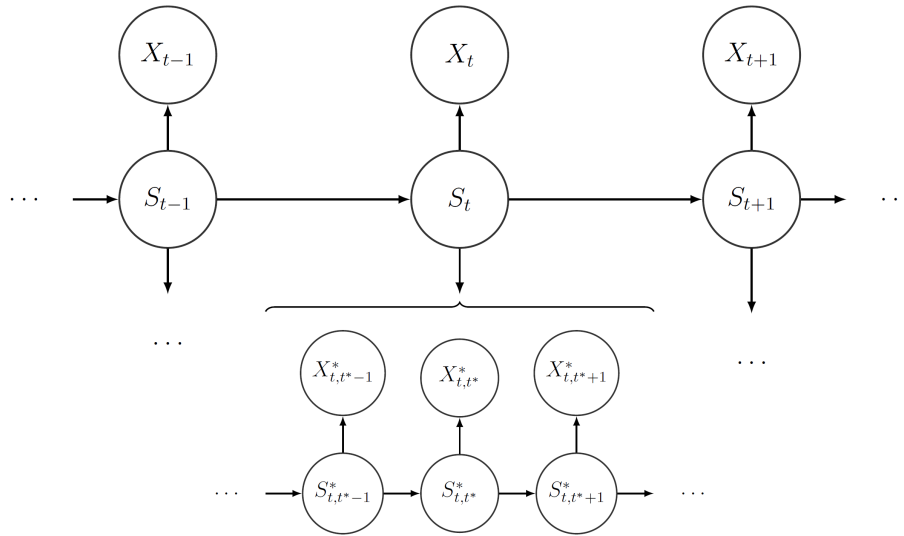


Figure 2: Dependence structure of an HHMM.

property and the conditional independence assumption, respectively, as well. Furthermore, it is assumed that the fine-scale HMM explaining $(X_{t,t^*}^*)_{t^*}$ only depends on S_t . This hierarchical structure is visualized in Figure 2.

3. Model specification

Model specification in the **fHMM** is done by specifying a named list of controls and passing it to the `set_controls()` function. The function checks the specifications and returns an ‘**fHMM_controls**’ object which stores all specifications and thereby provides required information for other **fHMM** functionalities. In the following, we demonstrate example specifications that should help the user to tailor a hidden Markov model to their need. All possible specifications are documented in detail on the help page `help(set_controls, package = "fHMM")`. Say that we want to fit an HMM to the closing prices of the Deutscher Aktienindex DAX Janssen and Rudolph (1992). The data can be obtained from <https://finance.yahoo.com/> or directly via the `download_data()` function. The next section on data management introduces this function and lists all data requirements. For now, assume that our working directory contains the file `"dax.csv"`, which is in the correct format. Consider the following lines:

```
R> controls <- list(
+   states = 3,
+   sdds   = "t",
+   data   = list(file      = "dax.csv",
+                   date_column = "Date",
+                   data_column = "Close",
+                   logreturns  = TRUE)
+ )
R> controls <- set_controls(controls)
```

```
R> class(controls)
```

```
[1] "fHMM_controls"
```

We specified a 3-state HMM (`states = 3`) with state-dependent t-distributions (`sdds = "t"`). The `data` element sets the path to the file (`file = "dax.csv"`), the file's column that contains the dates (`date_column = "Date"`) and the data (`date_column = "Close"`), and the command `logreturns = TRUE` that transforms the data to log-returns. The output of `set_controls()` can then be passed to the estimation routine introduced in section 5.

If the `data` element is not set, data will be simulated from the model specification. For example, the following specifies a 2-state HMM with state-dependent Gamma distributions, where the expectation values for state 1 and 2 are fixed to 0.5 and 2, respectively. The model will be fitted to 500 data points (`horizon = 500`) simulated from this specification based on `runs = 50` randomly initialized numerical optimization of the model's log-likelihood function:

```
R> controls <- list(
+   states = 2,
+   sdds    = "gamma(mu = 0.5|2)",
+   horizon = 500,
+   fit      = list(runs = 50)
+ )
R> set_controls(controls)
```

```
fHMM controls:
* hierarchy: FALSE
* data type: simulated
* number of states: 2
* sdds: gamma(mu = 0.5|2)
* number of runs: 50
```

The print method of the 'fHMM_controls' object summarizes the specification. Per default, the model has no hierarchy. To specify a hierarchical HMM, add `hierarchy = TRUE`. The rest is analogue, except that some parameters now can be specified for both hierarchies, where the first entry is for the coarse- and the second for the fine-scale, respectively:

```
R> controls <- list(
+   hierarchy = TRUE,
+   states    = c(3, 2),
+   sdds       = c("t(df = 1)", "t(df = Inf)"),
+   horizon    = c(100, 10)
+ )
R> set_controls(controls)
```

```
fHMM controls:
* hierarchy: TRUE
* data type: simulated
```

```
* number of states: 3 2
* sdds: t(df = 1) t(df = Inf)
* number of runs: 100
```

We specified an hierarchical HMM with 3 coarse-scale and 2 fine-scale states `states = c(3, 2)` and state-dependent t-distributions. The degrees of freedom are fixed to `-1` on the coarse-scale and `Inf` on the fine-scale (which results in a normal distribution).

4. Data management

Empirical data for modeling must be provided as a comma-separated values (CSV) file and its path must be specified in `set_controls()`, see the previous section. We recommend to use the data provided by <https://finance.yahoo.com/>. The **fHMM** package includes the convenience function `download_data()` for downloading daily stock data directly from the website in the required format. The function call is `download_data(symbol, from, to, file)`, where

- `symbol` is the stock's symbol that has to match the official symbol on <https://finance.yahoo.com>,
- `from` and `to` define the time interval (in format "YYYY-MM-DD"),
- `file` is the name of the file where the .csv-file is saved. Per default, it is saved in the current working directory under the name `<symbol>.csv`.

For example, the call

```
R> download_data(symbol = "^GDAXI", from = "2000-01-01", to = Sys.Date())
```

Download successful.

```
* symbol: ^GDAXI
* from: 2000-01-03
* to: 2022-04-04
* path: C:\Users\Lennart\Projekte\fHMM\jss\^GDAXI.csv
```

downloads the 21st century daily data of the DAX into the current working directory.

The **fHMM** package already includes two data sets of the Deutscher Aktienindex DAX and the VW stock for demonstration purpose that can be accessed as follows:

```
R> system.file("extdata", "dax.csv", package = "fHMM")
R> system.file("extdata", "vw.csv", package = "fHMM")
```

The `prepare_data` function prepares the data based on the 'fHMM_controls' specifications and returns an 'fHMM_data' object that can be passed to the `fit_model()` function for model fitting, see the next section:

```
R> controls <- list(
+   states = 3,
+   sdds   = "t",
```

```

+   data   = list(file       = system.file("extdata", "dax.csv", package = "fHMM"),
+                 date_column = "Date",
+                 data_column = "Close",
+                 from        = "2000-01-01",
+                 to          = "2021-12-31",
+                 logreturns  = TRUE),
+   fit     = list(runs      = 100)
+ )
R> controls <- set_controls(controls)
R> data <- prepare_data(controls)
R> class(data)

```

```
[1] "fHMM_data"
```

The ‘fHMM_data’ object has a `summary()` method for a quick data overview:

```
R> summary(data)
```

```

Summary of fHMM empirical data
* number of observations: 5626
* data source: dax.csv
* date column: Date
* log returns: TRUE

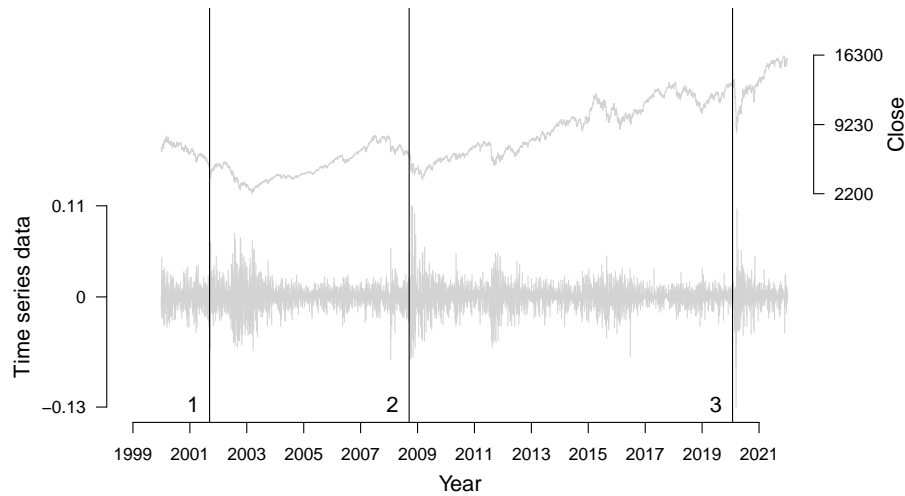
```

Additionally, the data can be visualized via the `plot()` method, where historical events can be highlighted by specifying a named list `events` with elements `dates` (a vector of dates) and `labels` (a vector of labels for the events) and passing it to the plot method, for example:

```

R> events <- fHMM_events(
+   list(
+     dates = c("2001-09-11", "2008-09-15", "2020-01-27"),
+     labels = c("9/11 terrorist attack", "Bankruptcy of Lehman Brothers",
+               "First COVID-19 case in Germany")
+   )
+ )
R> plot(data, events = events)

```

1: 9/11 terrorist attack 2: Bankruptcy of Lehman Brothers 3: First COVID-19 case in Germany

As mentioned in the previous section, if the `data` parameter in the model's `controls` is unspecified, the model is fitted to simulated data from the model specification. This can be useful for testing the functionality or conducting simulation experiments. True model parameters can be specified by defining an `'fHMM_parameters'`-object via the `fHMM_parameters()` function and passing it to `prepare_data()`.

An example: In the previous section, we specified the following `controls`:

```
R> controls <- list(
+   states = 2,
+   sdds   = "gamma(mu = 0.5/2)",
+   horizon = 500,
+   fit     = list(runs = 50)
+ )
R> controls <- set_controls(controls)
```

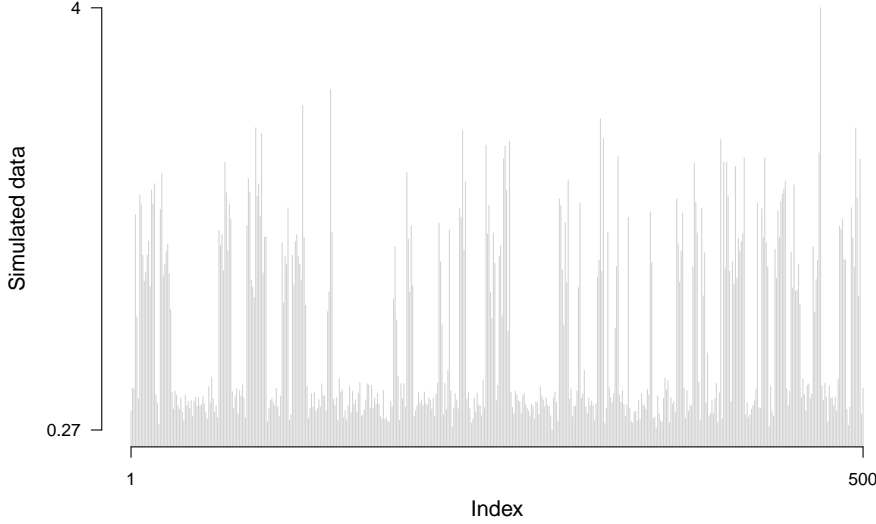
The means of the Gamma distributions in the two states are fixed to 0.5 and 2. With the following lines, we specify the transition probability matrix `Gamma` and the standard deviations `sigmas`:

```
R> pars <- fHMM_parameters(
+   controls = controls, Gamma = matrix(c(0.9,0.2,0.1,0.8), nrow = 2),
+   sigmas = c(0.1,0.5)
+ )
R> class(pars)
```

```
[1] "fHMM_parameters"
```

Submitting the `'fHMM_parameters'` object together with `controls` to `prepare_data()` creates the (simulated) `'fHMM_data'` object:

```
R> data <- prepare_data(controls, true_parameters = pars, seed = 1)
R> plot(data)
```



The ‘fHMM_data’ object can now be passed to the `fit_model()` function for model estimation, which is explained in the next section.

5. Model estimation

The **fHMM** package fits a hidden Markov model via the maximum-likelihood method, i.e. by numerically maximizing the likelihood function. Deriving the likelihood function of an hidden Markov model is part of the hierarchical case, hence the following only discusses the general case.

An HHMM can be treated as an HMM with two conditionally independent data streams; the coarse-scale observations on the one hand and the corresponding chunk of fine-scale observations connected to a fine-scale HMM on the other hand. To derive the likelihood of an HHMM, we start by computing the likelihood of each chunk of fine-scale observations being generated by each fine-scale HMM.

To fit the i -th fine-scale HMM, with model parameters $\theta^{*(i)} = (\delta^{*(i)}, \Gamma^{*(i)}, (f^{*(i,k)})_k)$ to the t -th chunk of fine-scale observations, which is denoted by $(X_{t,t^*})_{t^*}$, we consider the fine-scale forward probabilities

$$\alpha_{k,t^*}^{*(i)} = f^{*(i)}(X_{t,1}^*, \dots, X_{t,t^*}^*, S_{t,t^*}^* = k),$$

where $t^* = 1, \dots, T^*$ and $k = 1, \dots, N^*$. Using the fine-scale forward probabilities, the fine-scale likelihoods can be obtained from the law of total probability as

$$\mathcal{L}^{\text{HMM}}(\theta^{*(i)} \mid (X_{t,t^*}^*)_{t^*}) = \sum_{k=1}^{N^*} \alpha_{k,T^*}^{*(i)}.$$

The forward probabilities can be calculated in a recursively as

$$\begin{aligned}\alpha_{k,1}^{*(i)} &= \delta_k^{*(i)} f^{*(i,k)}(X_{t,1}^*), \\ \alpha_{k,t^*}^{*(i)} &= f^{*(i,k)}(X_{t,t^*}^*) \sum_{j=1}^{N^*} \gamma_{jk}^{*(i)} \alpha_{j,t^*-1}^{*(i)}, \quad t^* = 2, \dots, T^*.\end{aligned}$$

The transition from the likelihood function of an HMM to the likelihood function of an HHMM is straightforward: Consider the coarse-scale forward probabilities

$$\alpha_{i,t} = f(X_1, \dots, X_t, (X_{1,t^*}^*)_{t^*}, \dots, (X_{t,t^*}^*)_{t^*}, S_t = i),$$

where $t = 1, \dots, T$ and $i = 1, \dots, N$. The likelihood function of the HHMM results as

$$\mathcal{L}^{\text{HHMM}}(\theta, (\theta^{*(i)})_i \mid (X_t)_t, ((X_{t,t^*}^*)_{t^*})_t) = \sum_{i=1}^N \alpha_{i,T}.$$

The coarse-scale forward probabilities can be calculated similarly by applying the recursive scheme

$$\begin{aligned}\alpha_{i,1} &= \delta_i \mathcal{L}^{\text{HMM}}(\theta^{*(i)} \mid (X_{1,t^*}^*)_{t^*}) f^{(i)}(X_1), \\ \alpha_{i,t} &= \mathcal{L}^{\text{HMM}}(\theta^{*(i)} \mid (X_{t,t^*}^*)_{t^*}) f^{(i)}(X_t) \sum_{j=1}^N \gamma_{ji} \alpha_{j,t-1}, \quad t = 2, \dots, T.\end{aligned}$$

To account for parameter constraints associated with the transition probabilities (and potentially the parameters of the state-dependent distributions), we use parameter transformations. To ensure that the entries of the t.p.m.s fulfill non-negativity and the unity condition, we estimate unconstrained values $(\eta_{ij})_{i \neq j}$ for the non-diagonal entries of Γ and derive the probabilities using the multinomial logit link

$$\gamma_{ij} = \frac{\exp[\eta_{ij}]}{1 + \sum_{k \neq i} \exp[\eta_{ik}]}, \quad i \neq j$$

rather than estimating the probabilities $(\gamma_{ij})_{i,j}$ directly. The diagonal entries result from the unity condition as

$$\gamma_{ii} = 1 - \sum_{j \neq i} \gamma_{ij}.$$

Furthermore, variances are strictly positive, which can be achieved by applying an exponential transformation to the unconstrained estimator.

When numerically maximizing the likelihood using some Newton-Raphson-type method, we often face numerical under- or overflow, which can be addressed by maximizing the logarithm of the likelihood and incorporating constants in a conducive way (see [Zucchini *et al.* \(2016\)](#) and [Oelschläger and Adam \(2021\)](#) for details).

As the likelihood is maximized with respect to a relatively large number of parameters, the obtained maximum can be a local rather than the global one. To avoid this problem, it is recommended to run the maximization multiple times from different, possibly randomly

selected starting points, and to choose the model that corresponds to the highest likelihood (see [Zucchini *et al.* \(2016\)](#) and [Oelschläger and Adam \(2021\)](#) for details).

For illustration, we fit a 3-state HMM with state-dependent t-distributions to the DAX data. We use the same specification as in the previous section:

```
R> controls <- list(
+   states = 3,
+   sdds   = "t",
+   data   = list(file      = system.file("extdata", "dax.csv", package = "fHMM"),
+                 date_column = "Date",
+                 data_column = "Close",
+                 from        = "2000-01-01",
+                 to          = "2021-12-31",
+                 logreturns  = TRUE),
+   fit     = list("runs" = 100)
+ )
R> controls <- set_controls(controls)
R> data <- prepare_data(controls)
```

The ‘data’ object can be directly passed to the `fit_model()` function that numerically maximizes the model’s (log-) likelihood function `runs = 100` times. For convenience, the ‘controls’ object is saved in ‘data’ and does not need to be submitted to `fit_model()`. The numerical maximization runs can be parallelized by setting the `ncluster` argument. Setting `ncluster = 1` avoids any dependency on packages that provide clustering.

```
R> dax_model_3t <- fit_model(data, seed = 1, verbose = FALSE)
```

The estimated model is saved in the **fHMM** package and can be accessed as follows:

```
R> data(dax_model_3t)
```

The `coef()` method returns a data frame of the estimated model coefficients:

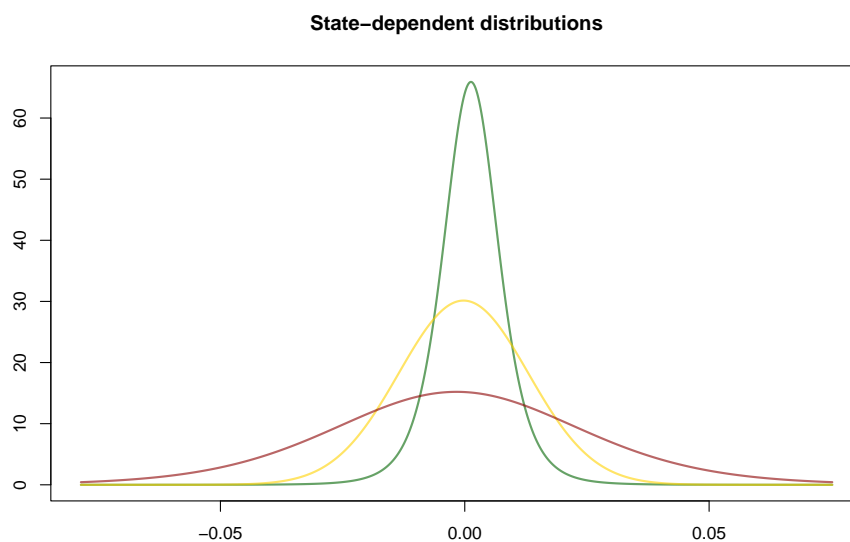
```
R> coef(dax_model_3t)
```

	lb	estimate	ub
Gamma_2.1	1.393745e-02	2.170056e-02	3.357222e-02
Gamma_3.1	1.710420e-06	1.696596e-06	1.671225e-06
Gamma_1.2	1.659146e-02	2.641008e-02	4.179228e-02
Gamma_3.2	9.359401e-03	1.740175e-02	3.213059e-02
Gamma_1.3	1.258535e-08	1.246159e-08	1.226657e-08
Gamma_2.3	2.789877e-03	5.145960e-03	9.431251e-03
mu_1	9.610235e-04	1.268712e-03	1.576400e-03
mu_2	-7.955275e-04	-2.517499e-04	2.920278e-04
mu_3	-3.723796e-03	-1.673649e-03	3.764982e-04
sigma_1	5.332119e-03	5.774343e-03	6.253244e-03

sigma_2	1.269075e-02	1.323801e-02	1.380887e-02
sigma_3	2.329677e-02	2.562622e-02	2.818860e-02
df_1	3.964154e+00	5.272157e+00	7.011747e+00
df_2	7.592272e+04	7.592547e+04	7.592821e+04
df_3	5.501298e+00	1.064107e+01	2.058285e+01

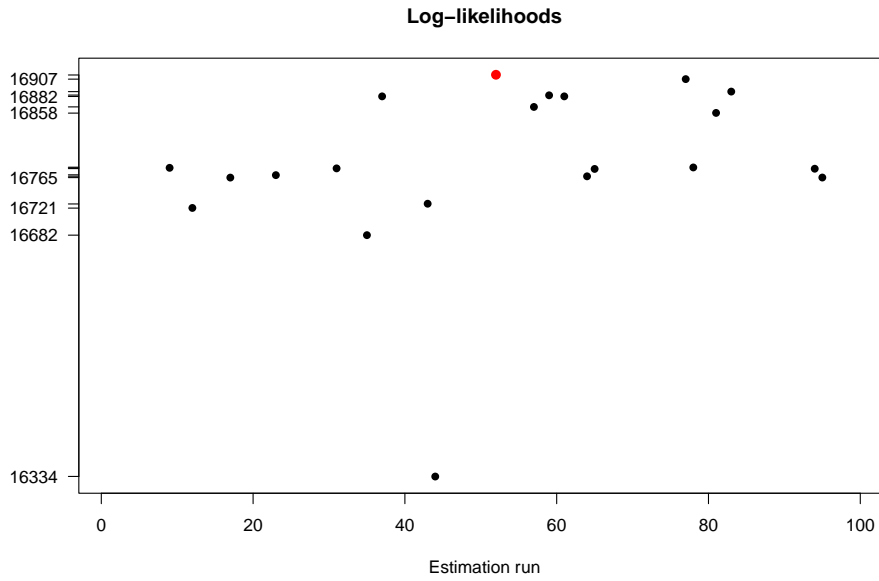
The estimated state-dependent distributions can be plotted as follows:

```
R> plot(dax_model_3t, plot_type = "sdds")
```



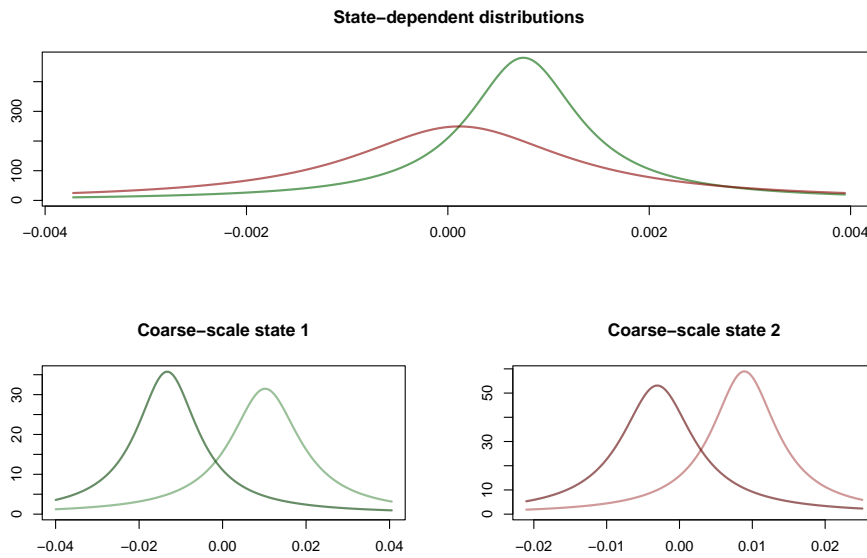
As mentioned above, the HMM likelihood function is prone to local optima. This effect can be visualized by plotting the log-likelihood value in the different optimization runs, where the best run is marked in red:

```
R> plot(dax_model_3t, plot_type = "ll")
```



The package also includes an hierarchical HMM with the following specification:

```
R> data(dax_vw_model)
R> plot(dax_vw_model, plot_type = "sdds")
```



6. State decoding and predicition

For financial markets, it is of special interest to infer the underlying (hidden) states in order to gain insight about the actual market situation. Decoding a full time series S_1, \dots, S_T is called **global decoding**. Hereby, we aim to find the most likely trajectory of hidden states

under the estimated model. Global decoding can be accomplished by using the so-called Viterbi algorithm which is a recursive scheme enabling to find the global maximum without being confronted with huge computational costs. To this end, we follow [Zucchini *et al.* \(2016\)](#) and define

$$\zeta_{1i} = \Pr(S_1 = i, X_1 = x_1) = \delta_i p_i(x_1)$$

for $i = 1, \dots, N$ and for the following $t = 2, \dots, T$

$$\zeta_{ti} = \max_{s_1, \dots, s_{t-1}} \Pr(S_{t-1} = s_{t-1}, S_t = i, X_t = x_t).$$

Then, the trajectory of most likely states i_1, \dots, i_T can be calculated recursively from

$$i_T = \operatorname{argmax}_{i=1, \dots, N} \zeta_{Ti}$$

and for the following $t = T - 1, \dots, 1$ from

$$i_t = \operatorname{argmax}_{i=1, \dots, N} (\zeta_{ti} \gamma_{i, i_{t+1}}).$$

Transferring the state decoding to HHMMs is straightforward: at first the coarse-scale state process must be decoded. Afterwards, by using this information the fine-scale state process can be decoded, see [Adam, Griffiths, Leos-Barajas, Meese, Lowe, Blackwell, Righton, and Langrock \(2019\)](#).

We revisit the DAX model of the vignette on model estimation:

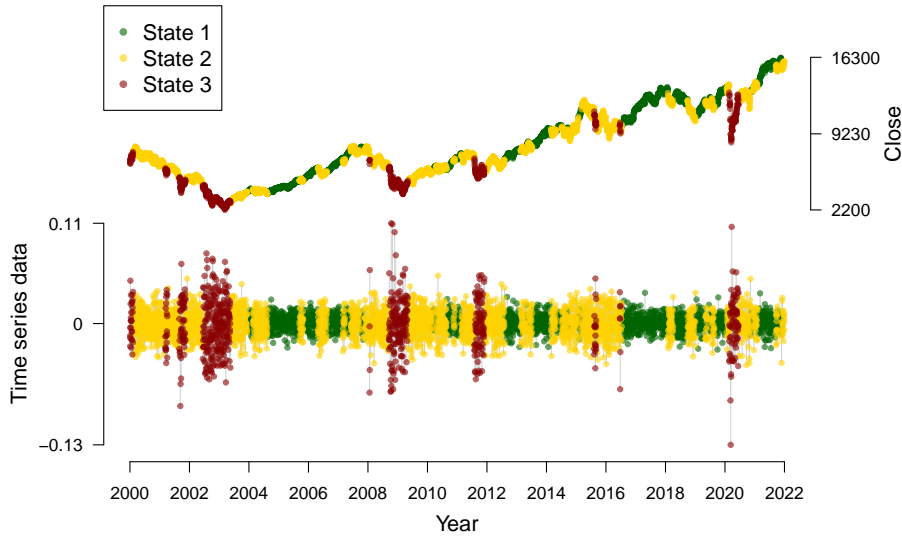
```
R> data(dax_model_3t)
```

The underlying states can be decoded via the `decode_states()` function:

```
R> dax_model_3t <- decode_states(dax_model_3t)
```

We now can visualize the decoded time series:

```
R> plot(dax_model_3t)
```



Mind that the model is invariant to permutations of the state labels. Therefore, **fHMM** provides the option to switch labels after decoding via the `reorder_states()` function, for example:

```
R> dax_model_3t <- reorder_states(dax_model_3t, 3:1)
```

Having decoded the underlying states, it is possible to compute the state probabilities of next observations. Based on these probabilities and in combination with the estimated state-dependent distributions, next observations can be predicted, compare [Zucchini *et al.* \(2016\)](#):

```
R> predict(dax_model_3t, ahead = 10)
```

	state_1	state_2	state_3	lb	estimate	ub
1	0.00515	0.97315	0.02170	-0.02190	-0.00023	0.02145
2	0.01006	0.94769	0.04225	-0.02179	-0.00020	0.02138
3	0.01477	0.92354	0.06170	-0.02168	-0.00018	0.02132
4	0.01926	0.90063	0.08011	-0.02158	-0.00016	0.02126
5	0.02356	0.87890	0.09754	-0.02148	-0.00014	0.02121
6	0.02767	0.85829	0.11403	-0.02140	-0.00012	0.02116
7	0.03161	0.83874	0.12965	-0.02131	-0.00010	0.02111
8	0.03537	0.82020	0.14442	-0.02124	-0.00008	0.02107
9	0.03898	0.80261	0.15841	-0.02116	-0.00007	0.02103
10	0.04243	0.78593	0.17164	-0.02110	-0.00005	0.02100

7. Model checking

Analyzing pseudo-residuals allows us to check whether the fitted model describes the data well. Since the observations are explained by different distributions (depending on the active state), this cannot be done by analyzing standard residuals. To transform all observations on a common scale, we proceed as follows: If X_t has the invertible distribution function F_{X_t} , then

$$Z_t = \Phi^{-1}(F_{X_t}(X_t))$$

is standard normally distributed, where Φ denotes the cumulative distribution function of the standard normal distribution. The observations, $(X_t)_t$, are modeled well if the so-called pseudo-residuals, $(Z_t)_t$, are approximately standard normally distributed, which can be visually assessed using quantile-quantile plots or further investigated using statistical tests such as the Jarque-Bera test [Zucchini *et al.* \(2016\)](#).

For HHMMs, we first decode the coarse-scale state process using the Viterbi algorithm. Subsequently, we assign each coarse-scale observation its distribution function under the fitted model and perform the transformation described above. Using the Viterbi-decoded coarse-scale states, we then treat the fine-scale observations analogously.

In **fHMM**, pseudo-residuals can be computed via the `compute_residuals()` function, provided that the states have been decoded beforehand.

We revisit the DAX example:

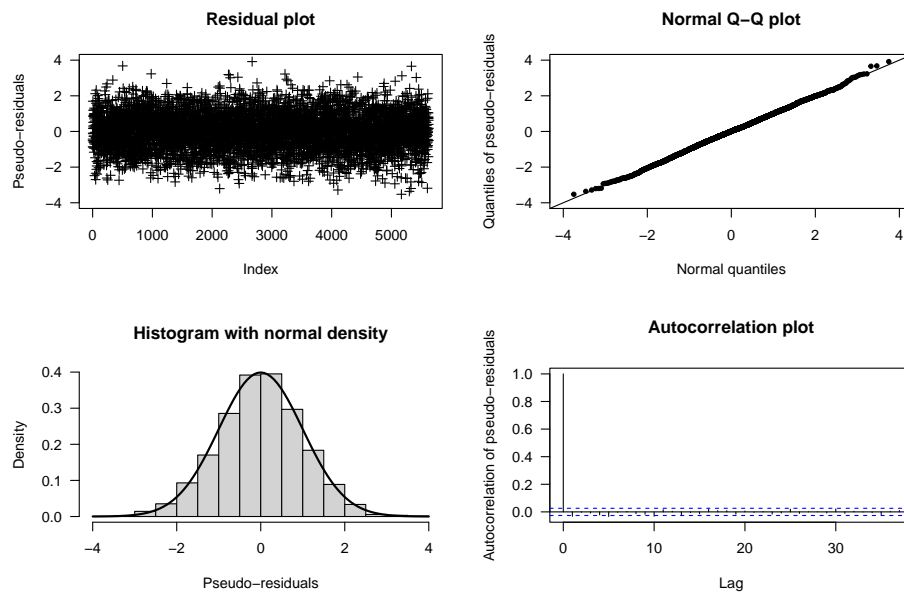

```
R> data(dax_model_3t)
```

The following line computes the residuals and saves them into the `model` object:

```
R> dax_model_3t <- compute_residuals(dax_model_3t)
```

The residuals can be visualized as follows:

```
R> plot(dax_model_3t, plot_type = "pr")
```



For additional normality tests, the residuals can be extracted from the `model` object. The following lines exemplarily perform a Jarque-Bera test [Jarque and Bera \(1987\)](#):

```
R> res <- dax_model_3t$residuals
R> tseries::jarque.bera.test(res)
```

Jarque Bera Test

```
data:  res
X-squared = 2.6542, df = 2, p-value = 0.2652
```

8. Model selection

Model selection involves the choice of a family for the state-dependent distribution and the selection of the number of states. Common model selection tools are information criteria, such as the Akaike information criterion (AIC) or the Bayesian information criterion (BIC), both of which aim at finding a compromise between model fit and model complexity.

The AIC is defined as

$$\text{AIC} = -2 \log(\mathcal{L}^{(\text{H})\text{HMM}}(\theta, (\theta^{*(i)})_i \mid (X_t)_t, ((X_{t,t^*}^*)_{t^*})_t)) + 2p,$$

where p denotes the number of parameters, while the BIC is defined as

$$\text{BIC} = -2 \log(\mathcal{L}^{(\text{H})\text{HMM}}(\theta, (\theta^{*(i)})_i \mid (X_t)_t, ((X_{t,t^*}^*)_{t^*})_t)) + \log(T)p,$$

where T is the number of observations.

In practice, however, information criteria often favor overly complex models. Real data typically exhibit more structure than can actually be captured by the model. This can be the case if the true state-dependent distributions are too complex to be fully modeled by some (rather simple) parametric distribution, or if certain temporal patterns are neglected in the model formulation. Additional states may be able to capture this structure, which can lead to an increased goodness of fit that outweighs the higher model complexity. However, as models with too many states are difficult to interpret and are therefore often not desired, information criteria should be treated with some caution and only considered as a rough guidance. For an in-depth discussion of pitfalls, practical challenges, and pragmatic solutions regarding model selection, see [Pohle, Langrock, van Beest, and Schmidt \(2017\)](#).

The **fHMM** package provides a convenient tool for comparing different models via the `compare_models()` function. The models (arbitrarily many) can be directly passed to the `compare_models()` function that returns an overview of the above model selection criteria. Below, we compare a 2-state HMM with normal state-dependent distributions with a 3-state HMM with state-dependent t-distributions for the DAX data, where the more complex model is clearly preferred:

```
R> data(dax_model_2n)
R> data(dax_model_3t)
R> compare_models(dax_model_2n, dax_model_3t)
```

	parameters	log-likelihood	AIC	BIC
dax_model_2n	6	16681.98	-33351.96	-33312.15
dax_model_3t	15	16913.33	-33796.65	-33697.13

9. Conclusions

The **fHMM** package intends to make the estimation of hidden Markov models for financial data available to practitioners in a user-friendly way. It contains functionality for data management, model fitting, state decoding, model checking, model selection, and methods for visualization. The hierarchical model extension is included for modeling trends on two temporal resolutions.

Computational details

The results in this paper were obtained using R 4.1.2 with the **fHMM** 1.0.0.9000 package. R itself and all packages used are available from the Comprehensive R Archive Network (CRAN) at <https://CRAN.R-project.org/>.

References

- Adam T, Griffiths C, Leos-Barajas V, Meese E, Lowe C, Blackwell P, Righton D, Langrock R (2019). “Joint modelling of multi-scale animal movement data using hierarchical hidden Markov models.” *Methods in Ecology and Evolution*, **10**(9), 1536–1550.
- Adam T, Mayr A, Kneib T (2022). “Gradient boosting in Markov-switching generalized additive models for location, scale, and shape.” *Econometrics and Statistics*, **22**, 3–16.
- Bulla J, Bulla I (2006). “Stylized facts of financial time series and hidden semi-Markov models.” *Computational Statistics and Data Analysis*, **51**(4), 2192–2209. ISSN 0167-9473. doi:[10.1016/j.csda.2006.07.021](https://doi.org/10.1016/j.csda.2006.07.021).
- Bulla J, Mergner S, Bulla I, Sesboüe A, Chesneau C (2011). “Markov-switching asset allocation: do profitable strategies exist?” *Journal of Asset Management*, **12**, 310–321. doi:[10.1057/jam.2010.27](https://doi.org/10.1057/jam.2010.27).
- Gregoir S, Lenglar F (2000). “Measuring the probability of a business cycle turning point by using a multivariate qualitative hidden Markov model.” *Journal of forecasting*, **19**(2), 81–102.
- Janssen B, Rudolph B (1992). “Der Deutsche Aktienindex DAX.” *Knapp Verlag*.
- Jarque C, Bera A (1987). “A Test for Normality of Observations and Regression Residuals.” *International Statistical Review / Revue Internationale de Statistique*, **55**, 163–172.
- Langrock R, Adam T, Leos-Barajas V, Mews S, Miller D, Papastamatiou Y (2018). “Spline-based nonparametric inference in general state-switching models.” *Statistica Neerlandica*, **72**(3), 179–200.
- Lihn S (2017). “Hidden Markov model for financial time series and its application to S&P 500 index.” *Quantitative Finance (forthcoming)*.
- Nguyen N (2018). “Hidden Markov model for stock trading.” *International Journal of Financial Studies*, **6**(2).
- Norris J (1997). “Markov Chains.” *Cambridge University Press*.
- Nystrup P, Madsen H, Lindström E (2015). “Stylised facts of financial time series and hidden Markov models in continuous time.” *Quantitative Finance*, **15**(9), 1531–1541. doi:[10.1080/14697688.2015.1004801](https://doi.org/10.1080/14697688.2015.1004801).
- Nystrup P, Madsen H, Lindström E (2018). “Dynamic portfolio optimization across hidden market regimes.” *Quantitative Finance*, **18**(1), 83–95. doi:[10.1080/14697688.2017.1342857](https://doi.org/10.1080/14697688.2017.1342857).
- Oelschläger L, Adam T (2021). “Detecting bearish and bullish markets in financial time series using hierarchical hidden Markov models.” *Statistical Modelling*. doi:[10.1177/1471082X211034048](https://doi.org/10.1177/1471082X211034048).
- Oelschläger L, Adam T, Michels R (2022). *fHMM: Fitting Hidden Markov Models to Financial Data*. R package version 1.0.1, URL <https://loelschlaeger.de/fHMM/>.

Pohle J, Langrock R, van Beest F, Schmidt N (2017). “Selecting the number of states in hidden Markov models: pragmatic solutions illustrated using animal movement.” *Journal of Agricultural, Biological and Environmental Statistics*, **22**(3), 270–293.

Zucchini W, MacDonald I, Langrock R (2016). “Hidden Markov models for time series: an introduction using R, 2nd Edition.” *Chapman and Hall/CRC*.

Affiliation:

Lennart Oelschläger
Department of Business Administration and Economics
Bielefeld University
Postfach 10 01 31
E-mail: lennart.oelschlaeger@uni-bielefeld.de