




## 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 versatile class of statistical models for time series that are driven by hidden states. In financial applications, the hidden states can often be linked to market regimes such as bearish and bullish markets or recessions and periods of economic growth. To give an example, when the market is in a nervous state, corresponding stock returns often follow some distribution with relatively high variance, whereas calm periods are often characterized by a different distribution with relatively smaller variance. Hidden Markov models can be used to explicitly model the distribution of the observations conditional on the hidden states and the transitions between states, and thus help us to draw a comprehensive picture of market behavior. While various implementations of hidden Markov models are available, a comprehensive R package that is tailored to financial applications is still lacking. In this paper, we introduce the R package **fHMM**, which provides various tools for applying hidden Markov models to financial time series. It contains functions for fitting hidden Markov models to data, conducting simulation experiments, and decoding the hidden state sequence. Furthermore, functions for model checking, model selection, and state prediction are provided. In addition to basic hidden Markov models, hierarchical hidden Markov models are implemented, which can be used to jointly model multiple data streams that were observed at different temporal resolutions. The aim of the **fHMM** package is to give R users with an interest in financial applications access to hidden Markov models and their extensions.

*Keywords:* hidden Markov models, hierarchical hidden Markov models, regime switching, financial time series, decoding market behavior, 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 hidden Markov chain can often be linked to market regimes such as bearish and bullish markets or recessions and periods of economic growths. To give an example, when the market is in a nervous state, corresponding stock returns often follow some distribution with relatively high variance, whereas when the market is in a calm state, another distribution with relatively smaller variance is active. By their dependence structure, HMMs naturally account for such disparate patterns and thus allow us to infer hidden market regimes and their underlying dynamics from financial time series.

Over the last decades, HMMs have become increasingly popular in finance. In various studies, they have been applied to model business cycles (Kim and Nelson 1998; Gregoir and Lengart 2000), to derive stylized facts of stock returns (Bulla and Bulla 2006; Nystrup, Madsen, and Lindström 2015), and to model energy prices conditional on market regimes (Langrock, Adam, Leos-Barajas, Mews, Miller, and Papastamatiou 2018; Adam, Langrock, and Kneib 2019b; Adam, Mayr, and Kneib 2022), to name but a few examples. Lihn (2017) used HMMs to model volatility in the Standard and Poor's 500 index to investigate the conjecture that stock returns exhibit negative correlation with volatility. Nguyen (2018) used HMMs to predict closing prices to derive an optimal trading strategy, which was shown to outperform the conventional buy-and-hold strategy, whereas Bulla, Mergner, Bulla, Sesboüe, and Chesneau (2011); Nystrup *et al.* (2015); Nystrup, Madsen, and Lindström (2018) have shown that HMMs prove useful in asset allocation and portfolio optimization applications. All these examples demonstrate that HMMs constitute a versatile class of statistical models for time series that naturally accounts for the state-switching patterns often found in financial data.

In R (R Core Team 2024), various implementations of HMMs are available. For general purposes, the packages **hmm** (Himmelmann 2022), **depmixS4** (Visser and Speekenbrink 2010), and **msm** (Jackson 2011) are frequently used. In addition, a wide range of special-purpose packages is available, for example **moveHMM** (Michelot, Langrock, and Patterson 2016) and **momentuHMM** (McClintock and Michelot 2018) for modeling ecological time series, **hsmm** (Bulla, Bulla, and Nenadić 2010) and **mhsmm** (O'Connell and Højsgaard 2011) for hidden semi-Markov models, **hmm.discnp** (Turner 2022) and **countHMM** (Adam 2019) for modeling count data, and **LMest** (Bartolucci, Pandolfi, and Pennoni 2017) for longitudinal data. In Python (Van Rossum *et al.* 2011), the library **hmmlearn** (Lebedex 2022) can be used. For MATLAB (The MathWorks Inc. 2021), the **Hidden Markov Model** toolbox (Chen 2022) is available. In Stata (StataCorp 2019), HMMs are implemented in the **mswitch()** function. For Julia (Bezanson, Edelman, Karpinski, and Shah 2017), the packages **HiddenMarkovModels.jl** (Dalle 2024) and **MarkovModels.jl** (Ondel, Lam-Yee-Mui, Kocour, Filippo, and Lukás Burget 2021) are available. While **ldhmm** (Lihn 2019) implements a symmetric lambda distribution framework for the study of share returns, a comprehensive R package tailored to financial applications is still lacking.

In this paper, we introduce the R package **fHMM** (Oelschläger, Adam, and Michels 2024), available from the Comprehensive R Archive Network (CRAN) at <https://CRAN.R-project.org/package=fHMM>. The package aims at complementing the above mentioned collection of implementations by making HMMs accessible to R users with an interest in financial time series. The package functionality can be classified into functions for data preparation, model estimation, and model evaluation, which are illustrated in the flowchart displayed in Figure 1. Functions for data preparation include a convenient interface to Yahoo Finance

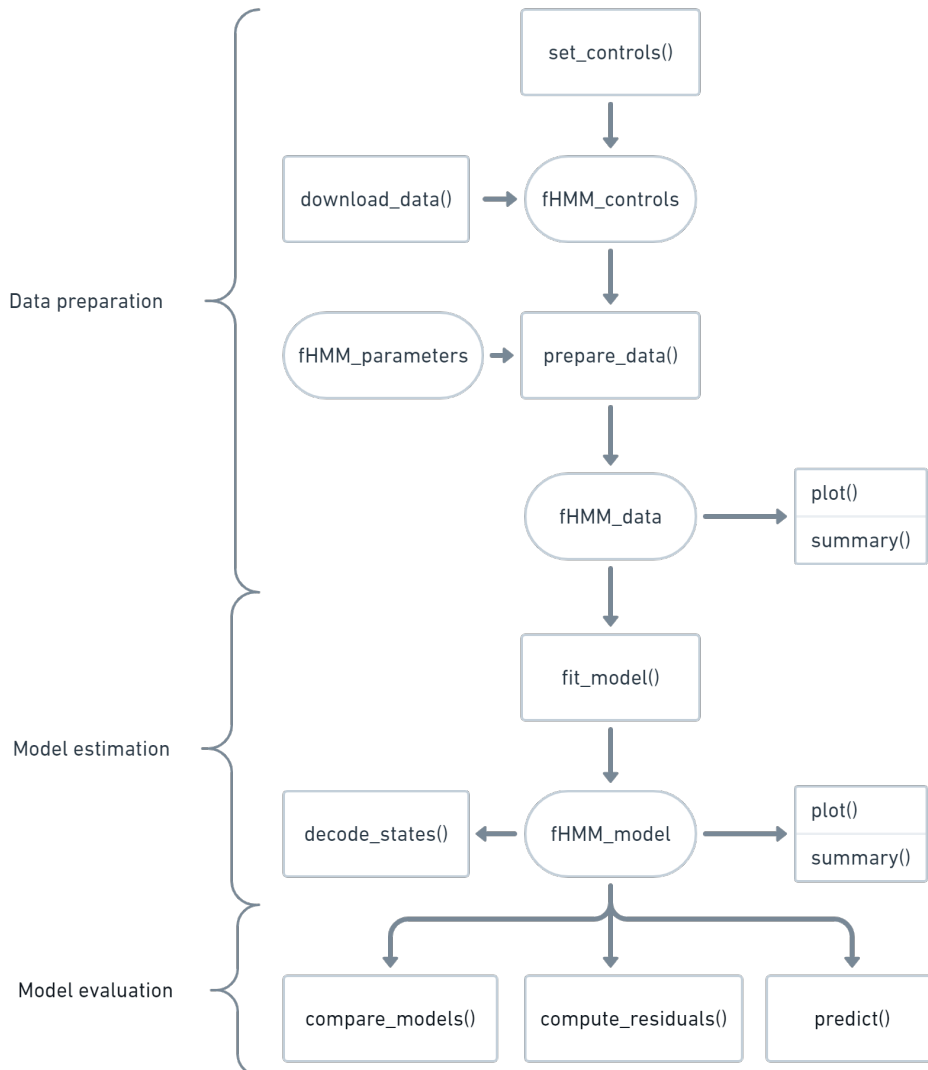


Figure 1: Flowchart. The main functions of the **fHMM** package are visualized using rectangles, while objects are illustrated as ovals.

(<https://www.finance.yahoo.com>) that allows users to download stock market 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 and efficient computation. Functions for model evaluation include pseudo-residual analyses and the computation of model selection criteria. In addition to basic HMMs, the package also implements hierarchical HMMs (HHMMs). HHMMs (Oelschläger and Adam 2021) constitute an extension that improves the model’s capability of distinguishing between short- and long-term trends in the data and allows us to jointly model multiple data streams that were collected at different temporal resolutions, such as monthly trade volumes and daily stock returns (Adam and Oelschläger 2020).

The paper is structured as follows: In Section 2, we introduce HMMs and HHMMs, where we focus on their dependence structure and the underlying model assumptions. In Sections 3–8,

we illustrate a typical **fHMM** workflow, explaining how to specify a model, how to download, prepare, and simulate data, how to fit a model, how to decode the hidden states, how to use a fitted model for state forecasting, how to check the goodness of fit, and how to perform model selection. Each section begins with some theoretical background, which is followed by illustrating examples using stock market data from the Deutscher Aktienindex (DAX), the Standard & Poor's 500 (SPX) and US unemployment rates, as well as simulated data. Each of these sections is complemented by code chunks, which cannot only be used to replicate the examples given in this paper but also serve as a starting point for R users with an interest in financial time series who want to apply HMMs to their own data. Section 9 concludes and gives an outlook of anticipated, future extensions of the **fHMM** package.

## 2. Model definition

Hidden Markov models (HMMs) constitute a modeling framework for time series where a sequence of observations is assumed to depend on a hidden state process. The peculiarity is that, instead of the observation process, the state process cannot be directly observed. However, the hidden states comprise information about the environment the model is applied on. The hidden state process and the observed state-dependent process are connected as follows: we assume that for each point in time  $t = 1, \dots, T$ , a hidden process  $(S_t)_{t=1, \dots, T}$  is in one of  $N$  possible 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 state-dependent distribution  $f^{(S_t)}$ . We assume  $(S_t)_t$  to be Markovian, i.e., the active state at time  $t$  only depends on the previous state at time  $t - 1$ . Henceforth, the state process is identified by its initial distribution  $\delta = (\delta_i)$ ,  $\delta_i = \Pr(S_1 = i)$ ,  $i = 1, \dots, N$ , and its transition probability matrix (t.p.m.)  $\Gamma = (\gamma_{ij})$ ,  $\gamma_{ij} = \Pr(S_t = j \mid S_{t-1} = i)$ ,  $i, j = 1, \dots, N$ ,  $t = 2, \dots, T$ . Furthermore,  $(X_t)_{t=1, \dots, T}$  satisfies the conditional independence assumption, i.e., the observation  $X_t$  depends on  $S_t$ , but is independent from previous observations or states.

When modeling financial time series, the different states can serve as proxies for the current market situation, e.g., calm and nervous. Even though these moods cannot be directly observed, price changes or trading volumes (which can be assumed to depend on the current mood of the market) are observable. Thereby, using a hidden Markov process, we can infer 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 explicitly model price changes at time  $t$  by different normal distributions whose mean and variance depend on the current state,  $S_t$ .

Following [Zucchini \*et al.\* \(2016\)](#), we assume that the initial distribution  $\delta$  equals the stationary distribution  $\pi$ , where  $\pi = \pi\Gamma$ , i.e., the stationary and henceforth the initial distribution is determined by  $\Gamma$ .<sup>1</sup> 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.

---

<sup>1</sup>A note on the existence of a stationary distribution: If the Markov process is irreducible, it has a unique distribution, which is the solution to the equation system  $\pi = \pi\Gamma$ . If, additionally, the Markov process is aperiodic, its state distribution converges to the stationary distribution, see [Norris \(1997\)](#). Irreducibility and aperiodicity are usually satisfied in practice.

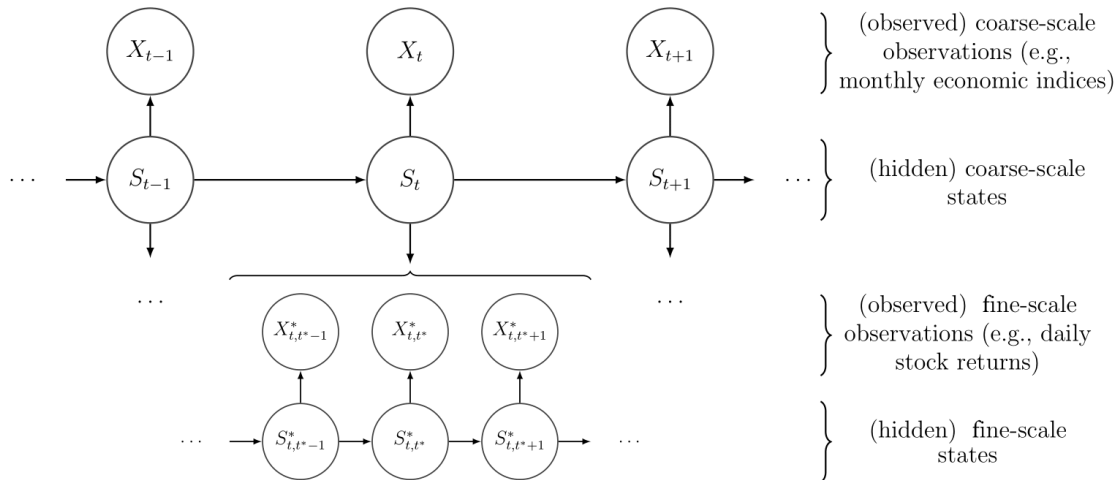


Figure 2: Dependence structure of an HHMM. The coarse-scale observations,  $X_t$ , depend on the state of the coarse-scale Markov chain,  $S_t$ . The fine-scale observations,  $X_{t,t^*}^*$ , depend on the state of fine-scale Markov chain,  $S_{t,t^*}^*$ , where the fine-scale HMM that is active depends on the state of the coarse-scale Markov chain.

HHMMs constitute a flexible extension of basic HMMs that can be used to jointly model multiple data observed on two different time scales (Oelschläger and Adam 2021). 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 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  $\{1, \dots, T\}$ , a hidden process  $(S_t)_t$  selects one state from the coarse-scale state space  $\{1, \dots, N\}$ . We refer to  $(S_t)_t$  as the hidden coarse-scale state process. Depending on which state is active at time  $t$ , one of  $N$  possible 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. This dependence structure is visualized in the upper part of Figure 2.

For the fine-scale time series, the observed data are split into  $T$  distinct chunks, each of them having a correspondence to the  $t$ -th coarse-scale time point. The hierarchical structure arises naturally: For each chunk, we model the corresponding data points via the fine-scale HMM that is selected by the hidden coarse-scale state  $S_t = i$ . Thus, each fine-scale HMM consists of two stochastic processes: The hidden fine-scale process  $(S_{t,t^*}^*)_{t^*}$  selecting states from  $\{1, \dots, N^*\}$ , the fine-scale state space, for each time point  $t^*$  in  $\{1, \dots, T^*\}$ , the fine-scale time space, and the observed fine-scale process  $(X_{t,t^*}^*)_{t^*}$ , whose observations are assumed to depend on one of  $N^*$  possible distributions  $f^{*(i,1)}, \dots, f^{*(i,N^*)}$ , chosen depending on the active fine-scale state. By construction, each fine-scale HMM contains an own t.p.m.  $\Gamma^{*(i)}$ , initial distribution  $\delta^{*(i)}$ , stationary distribution  $\pi^{*(i)}$ , and state-dependent distributions  $f^{*(i,1)}, \dots, f^{*(i,N^*)}$ . Similar to the coarse-scale HMM, the hidden fine-scale process is Markovian and satisfies the conditional independence assumption. In contrast, the observed

fine-scale process has exclusive dependence on the active state of the hidden fine-scale process. This dependence structure is visualized in Figure 2.

### 3. Model specification

In the `fHMM` package, models are specified by a named list of controls that is passed to the `set_controls()` function. This usually constitutes the first step when using the package, see Figure 1. The function checks the specifications and returns an ‘`fHMM_controls`’ object, which stores all settings and thereby provides the information required for other functionalities. In the following, we demonstrate three example specifications that should help the user to tailor an (hierarchical) HMM to their need. The examples are continued in the following sections. All possible specifications are documented in detail on the function’s help page, which can be accessed from the R console via `help(set_controls, package = "fHMM")`.

**Example 1: DAX.** We fit a 3-state HMM to the closing prices of the DAX (Janßen and Rudolph 1992). Assume that the data are available in the global environment as a `data.frame` called `dax`. Such data can be obtained directly from Yahoo Finance via the convenience function `download_data()`, see Section 4.

The following code chunk sets the number `states = 3` of hidden states. Any number greater than or equal to 2 is possible. Next, `sdds = "t"` specifies state-dependent  $t$  distributions, which provide a popular choice for modeling log-returns (Platen and Rendek 2008). Alternatively, `sdds = "gamma"` specifies Gamma distributions, which are useful for modeling trading volumes, see Adam and Oelschläger (2020). Additionally, normal, log-normal, and Poisson distributions are available via `sdds = "normal"`, `sdds = "lognormal"`, and `sdds = "poisson"`, respectively. The `data` entry sets the `data.frame` (or, alternatively, the path to a `.csv`-data file), the file’s column that contains the dates (`date_column = "Date"`), and the data (`date_column = "Close"`). The specification `logreturns = TRUE` transforms the data to log-returns. The entries `from` and `to` can be used to restrict the observation period. Finally, details of the fitting process can be specified via the `fit` entry, see Section 5.

```
R> contr_dax <- list(states = 3, sdds = "t",
+   data = list(file = dax, date_column = "Date", data_column = "Close",
+   logreturns = TRUE, from = "2000-01-03", to = "2022-12-31"),
+   fit = list(runs = 100, iterlim = 300, gradtol = 1e-6, steptol = 1e-6))
```

Passing this list to the `set_controls()` function returns an ‘`fHMM_controls`’ object, which contains the specifications. Any control that is not specified by the user is set to default settings, which are documented inside the `set_controls()` function.

```
R> contr_dax <- set_controls(contr_dax)
R> class(contr_dax)
```

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

**Example 2: Simulation.** If the `data` entry is not specified, data are simulated according to the model specification. Simulation typically serves to assess the properties of estimation

algorithms either for research or in a bootstrap-like fashion, as can be seen for example in [Oelschläger \(2019\)](#). The following code chunk specifies a 2-state HMM with state-dependent Gamma distributions, where the expected values<sup>2</sup> for state 1 and 2 are fixed to 1 and 2, respectively. The model is fitted to 200 data points (`horizon = 200`) simulated according to this specification based on `runs = 50` randomly initialized numerical optimization runs of the model's likelihood function.

```
R> contr_sim <- list(states = 2, sdds = "gamma(mu = 1|2)", horizon = 200,
+   fit = list(runs = 50))
R> contr_sim <- set_controls(contr_sim)
```

Printing the 'fHMM\_controls' object summarizes the model specification:

```
R> print(contr_sim)
```

```
fHMM controls:
* hierarchy: FALSE
* data type: simulated
* number of states: 2
* sdds: gamma(mu = 1|2)
* number of runs: 50
```

**Example 3: Multiple data streams.** Hierarchical HMMs enable joint modeling of multiple data streams that were collected at different temporal resolutions. As an illustration, we apply the model to 50 years of monthly US unemployment rates ([OECD 2023](#)) and S&P 500 closing prices. Both data sets (`unemp` and `spx`) are contained in the `fHMM` package; the data are visualized in Section 4. The following is an example specification:

```
R> contr_hhmm <- list(hierarchy = TRUE, states = c(3, 2),
+   sdds = c("t", "t"), period = "m",
+   data = list(file = list(unemp, spx),
+     data_column = c("rate_diff", "Close"),
+     date_column = c("date", "Date"),
+     from = "1970-01-01", to = "2020-01-01",
+     logreturns = c(FALSE, TRUE)),
+   fit = list(runs = 50, iterlim = 1000, gradtol = 1e-6, steptol = 1e-6))
R> contr_hhmm <- set_controls(contr_hhmm)
```

When setting `hierarchy = TRUE`, specifications for both layers must be made. In particular, `states = c(3, 2)` specifies 3 coarse-scale and 2 fine-scale states, respectively, while `sdds = c("t", "t")` selects state-dependent  $t$  distributions for both layers. Via `period = "m"`, we specify a monthly fine-scale time horizon. Here, we can choose between "w", "q", and "y" for weekly, quarterly, and yearly time horizons, respectively.<sup>3</sup> As coarse-scale observations, we

<sup>2</sup>Expected value and standard deviation of a Gamma distribution are obtained by means of a parameter transformation.

<sup>3</sup>If the coarse-scale data had a finer temporal resolution than defined by `period`, the data can be merged by specifying a function via the `merge` entry.

select column `rate_diff` of `data.frame unemp`, which contains relative differences in unemployment rate. As fine-scale observations, we select daily closing prices of the S&P 500 index. The data streams are synchronized via the date columns specified in `date_column`. The observation period is restricted to 50 years via `from = "1970-01-01"` and `to = "2020-01-01"`. With `logreturns = c(FALSE, TRUE)`, we ensure that only the fine-scale data are transformed to log-returns.

## 4. Data management

Empirical data for modeling must be provided as a `data.frame` in `set_controls()`, see the previous section. Alternatively, the path to a comma-separated values (`.csv`) file can be specified. The package includes the convenience function `download_data()` for downloading daily stock market data directly from Yahoo Finance 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 Yahoo Finance,
- `from` and `to` define the desired time interval (in the format "YYYY-MM-DD"),
- and `file` optionally specifies a file path to save the data.

For example, the 21st century data of the DAX can be downloaded via the following line:

```
R> dax <- download_data(symbol = "^GDAXI", from = "2000-01-01",
+   to = "2022-12-31")
R> head(dax)
```

	Date	Open	High	Low	Close	Adj.Close	Volume
1	2000-01-03	6961.72	7159.33	6720.87	6750.76	6750.76	43072500
2	2000-01-04	6747.24	6755.36	6510.46	6586.95	6586.95	46678400
3	2000-01-05	6585.85	6585.85	6388.91	6502.07	6502.07	52682800
4	2000-01-06	6501.45	6539.31	6402.63	6474.92	6474.92	41180600
5	2000-01-07	6489.94	6791.53	6470.14	6780.96	6780.96	56058900
6	2000-01-10	6785.47	6975.26	6785.47	6925.52	6925.52	42006200

**Example 1: DAX (continued).** Based on the specification from the previous section, the `prepare_data()` function prepares and returns the model data as an 'fHMM\_data' object. This object can then be passed to the `fit_model()` function for model fitting in the next step, see Section 5. The `summary()` method provides an overview of the data.

```
R> data_dax <- prepare_data(contr_dax)
R> summary(data_dax)
```

```
Summary of fHMM empirical data
* number of observations: 5882
* data source: data.frame
* date column: Date
* log returns: TRUE
```



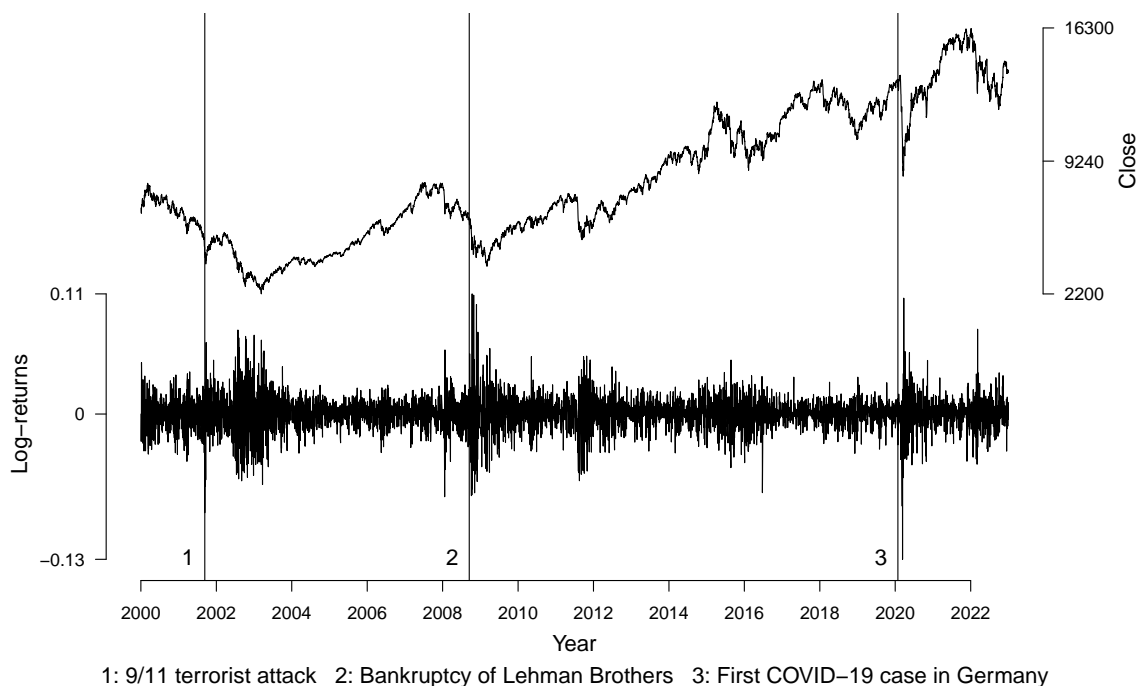


Figure 3: Time series of daily DAX prices and log-returns marked with historical events with a potential influence.

The data can be visualized in Figure 3 using the `plot()` method. To facilitate interpretation, historical events with a potential influence on the time series can be highlighted as follows:

```
R> events_dax <- 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_dax, events = events_dax)
```

We see that the considered events were followed by periods of dropping prices (top chart) and increased volatility in log-returns (bottom chart). Section 6 complements this plot by the decoded market regimes based on the fitted HMM.

**Example 2: Simulation (continued).** As mentioned in the previous section, if the `data` argument in the model's controls is unspecified, data are simulated according to the model specification. True model parameters can be selected by defining an `'fhmm_parameters'` object and passing it to `prepare_data()`, for example:

```
R> pars <- fhmm_parameters(controls = contr_sim,
+   Gamma = matrix(c(0.9, 0.2, 0.1, 0.8), nrow = 2), sigma = c(0.1, 0.5))
R> data_sim <- prepare_data(contr_sim, true_parameters = pars, seed = 1)
```

The plot of the simulated time series is shown in Figure 4 and is produced by the code below. It shows the state persistence (induced by  $\gamma_{11} = 0.9$  and  $\gamma_{22} = 0.8$ ) and the different standard

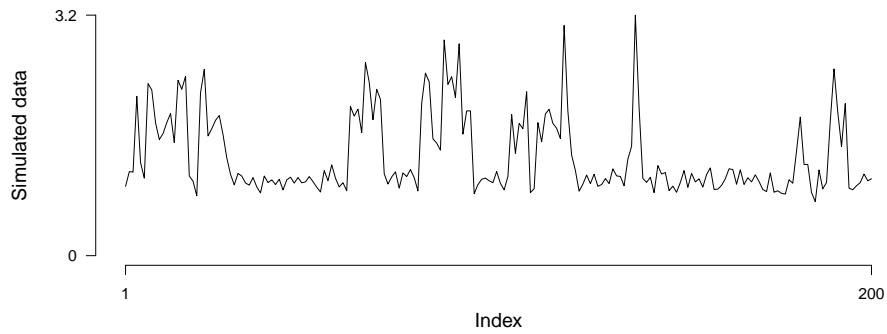


Figure 4: Simulated time series from Example 2.

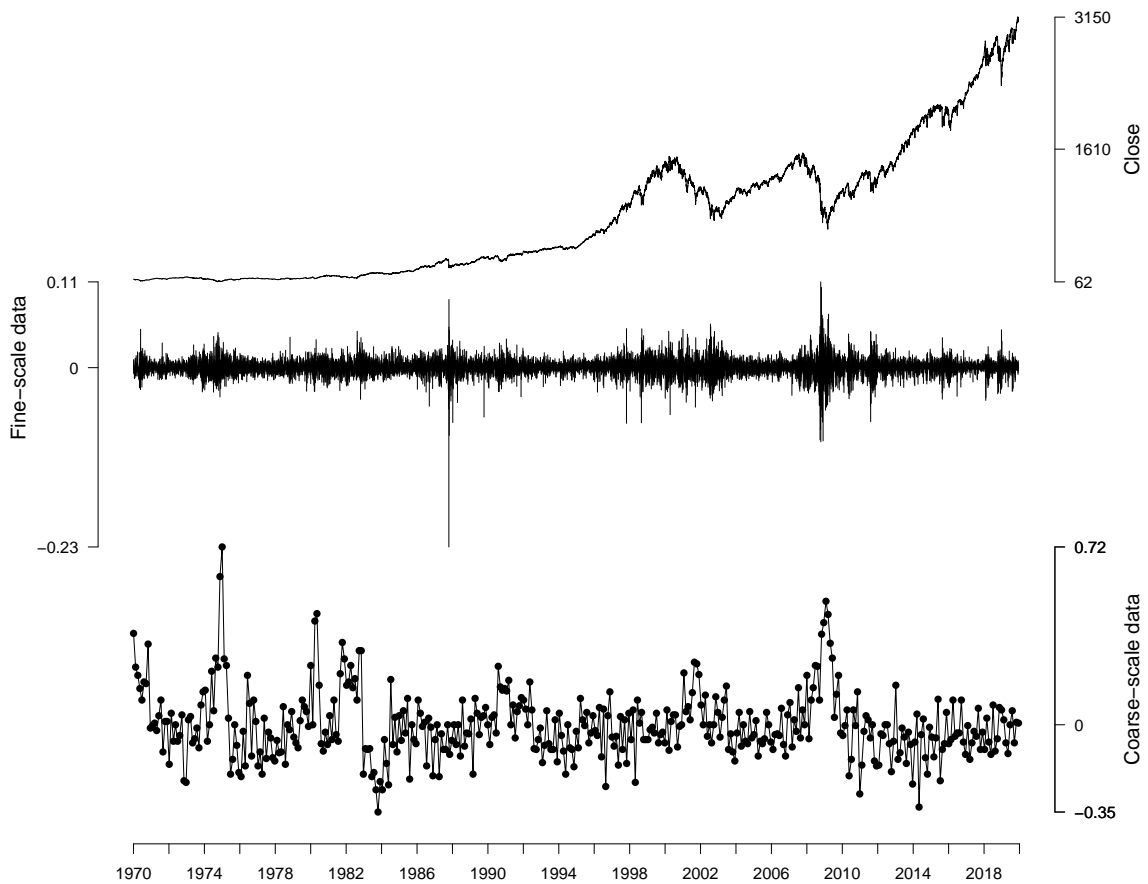


Figure 5: S&P 500 daily closing prices from 1970 to 2020 (top panel) and their corresponding log-returns (middle panel). The bottom panel shows the monthly differences in the US unemployment rate.

deviations ( $\sigma_1 = 0.1$  and  $\sigma_2 = 0.5$ ) of the state-dependent Gamma distributions. Remember that the expected values were fixed to  $\mu_1 = 1$  and  $\mu_2 = 2$ .

```
R> plot(data_sim)
```

**Example 3: Multiple data streams (continued).** Data preparation for the HHMM application can be done analogously:

```
R> data_hhmm <- prepare_data(contr_hhmm)
```

The following code produces Figure 5, which displays the S&P 500 closing prices from 1970 to 2020 in the top chart and the corresponding log-returns in the middle chart. The bottom chart shows the monthly changes in the US unemployment rate.

```
R> plot(data_hhmm)
```

In the following section, we will utilize the hierarchical HMM to investigate the relationship between these two economic indicators.

## 5. Model estimation

The **fhmm** package implements the maximum likelihood method for model estimation. In the following, the likelihood function of an HHMM is derived, the non-hierarchical case can be deduced. We also discuss challenges related to the numerical maximization and subsequently estimate the three running example models.

Note that 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 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:

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

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  $\Gamma$  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.

Two more technical difficulties arise: First, 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 [Oelschläger and Adam \(2021\)](#) for the details. Second, 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 ([Zucchini et al. 2016](#)). For efficient initialization, the **fHMM** package uses data clustering in combination with the first and second data moments as a basis for the initial guesses.

**Example 1: DAX (continued).** In Section 4, we defined the ‘fHMM\_data’ object `data_dax`. This object can be directly passed to the `fit_model()` function that numerically maximizes the model’s (log-) likelihood function as described above:<sup>4</sup>

```
R> dax_model_3t <- fit_model(data_dax, seed = 2, ncluster = 10)
```

The `coef()` method returns a `data.frame` of the estimated model coefficients along with 1–alpha confidence intervals (`alpha = 0.05` being the default) obtained via the inverse Fisher information (`lb` stands for lower- and `ub` for upper-bound of the intervals, respectively):

---

<sup>4</sup>For faster computation, the optimization runs can be parallelized by setting the function’s `ncluster` argument to a value greater than 1. The `seed` argument controls for the randomly generated initial values.

```
R> coef(dax_model_3t, alpha = 0.05)
```

	lb	estimate	ub
Gamma_2.1	2.717964e-03	4.997848e-03	9.133754e-03
Gamma_3.1	9.152749e-13	9.066110e-13	8.928059e-13
Gamma_1.2	1.011260e-02	1.850867e-02	3.363877e-02
Gamma_3.2	1.508443e-02	2.440749e-02	3.926302e-02
Gamma_1.3	2.510224e-13	2.488932e-13	2.450564e-13
Gamma_2.3	1.189853e-02	1.894514e-02	2.997985e-02
mu_1	-3.866569e-03	-1.786859e-03	2.928515e-04
mu_2	-7.968575e-04	-2.621635e-04	2.725305e-04
mu_3	9.636588e-04	1.271135e-03	1.578610e-03
sigma_1	2.354319e-02	2.588088e-02	2.845070e-02
sigma_2	1.225935e-02	1.300551e-02	1.379708e-02
sigma_3	5.392043e-03	5.835334e-03	6.315069e-03
df_1	5.539083e+00	1.083040e+01	2.117634e+01
df_2	6.985181e+00	4.762837e+01	3.247534e+02
df_3	3.974572e+00	5.250206e+00	6.935252e+00

The t.p.m. associated with the hidden state process was estimated as

$$\hat{\Gamma} = \begin{pmatrix} 0.981 & 0.019 & 0.000 \\ 0.005 & 0.976 & 0.019 \\ 0.000 & 0.024 & 0.976 \end{pmatrix},$$

which implies the stationary distribution  $(0.132, 0.489, 0.379)$ . The stationary state probabilities can be regarded as the long-term proportion of time that the state process spends in the different states. State 1 corresponds to the highest marginal volatility ( $\hat{\sigma}_1 = 25.9 \cdot 10^{-3}$ ) and lowest marginal expected return ( $\hat{\mu}_1 = -1.8 \cdot 10^{-3}$ ), while state 3 corresponds to the lowest marginal volatility ( $\hat{\sigma}_3 = 5.8 \cdot 10^{-3}$ ) and highest marginal expected return ( $\hat{\mu}_3 = 1.3 \cdot 10^{-3}$ ). The estimated degrees of freedom for state 2 are found to be very large, leading to the conclusion that the state-dependent  $t$  distribution approaches a normal distribution. The distributions for states 1 and 3 appear to require heavier tails. Adding `plot_type = "sdds"` to the `plot()` method visualizes the estimated distributions for the different states.

It is well known that likelihood optimization of mixture models is highly sensitive to the initial values supplied to the numerical optimizer (Shireman, Steinley, and Brusco 2017). To demonstrate the effect of local optima in the HMM likelihood, we can visualize the optimization results by adding `plot_type = "ll"`, which plots the differences in log-likelihood value over multiple optimization runs (the best run is marked in red). The following code produces the output in Figure 6.

```
R> par(mfrow = c(1, 2))
R> plot(dax_model_3t, plot_type = c("ll", "sdds"))
```

**Example 2: Simulation (continued).** Fitting an HMM to the simulated data is analogue. The `summary()` method gives an overview of the estimated model.

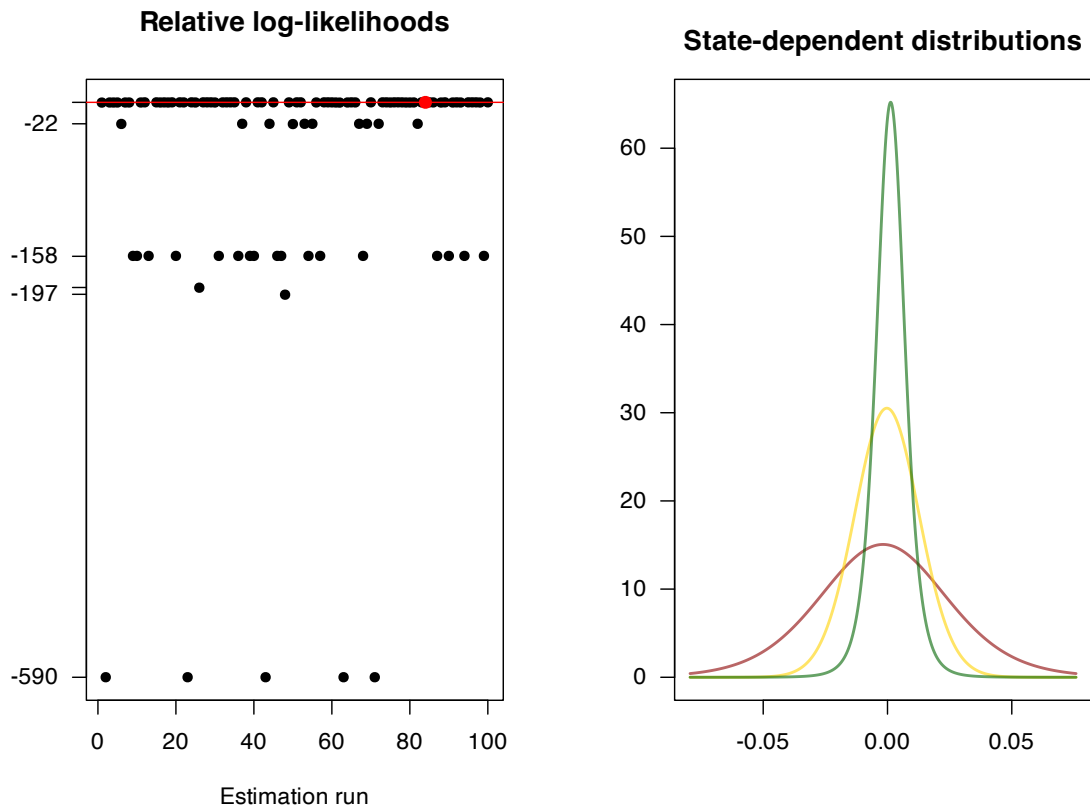


Figure 6: Differences in log-likelihood value over multiple optimization runs (left panel) and estimated distributions for the different states (right panel) for the DAX example.

```
R> sim_model_2gamma <- fit_model(data_sim, seed = 1)
R> summary(sim_model_2gamma)
```

Summary of fHMM model

	simulated hierarchy	LL	AIC	BIC
1	TRUE FALSE	13.04795	-18.09589	-4.902621

State-dependent distributions:

gamma(mu = 1|2)

Estimates:

	lb	estimate	ub	true
Gamma_2.1	0.07855	0.14334	0.2472	0.2
Gamma_1.2	0.03909	0.07325	0.1331	0.1
sigma_1	0.09163	0.10423	0.1186	0.1
sigma_2	0.35833	0.43239	0.5217	0.5

In simulated settings, we can assess the accuracy of the estimates by comparing them to the true model coefficients (column `true`), providing an evaluation of the model's ability to

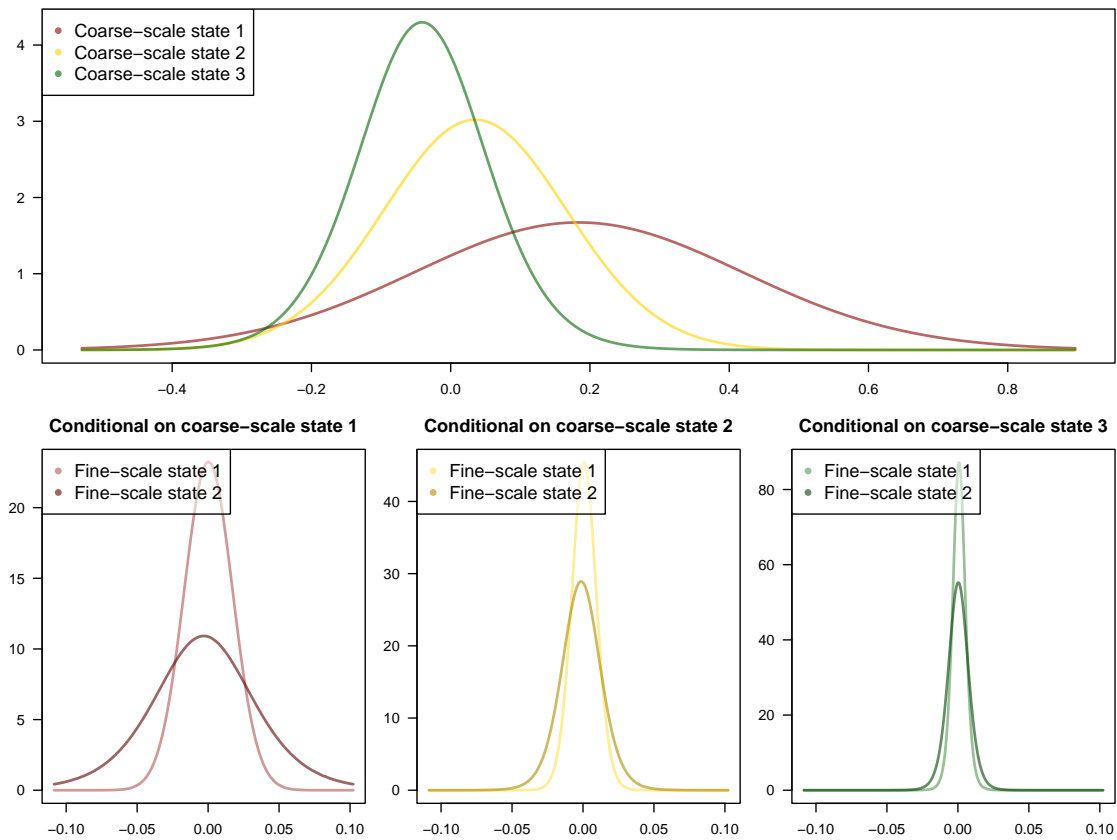


Figure 7: Estimated state-dependent distributions at both the coarse-scale and fine-scale layer for the multiple data streams example.

reproduce the true parameters. We see that in this example, the true parameters are relatively close to the estimates and lie within the 95% confidence intervals.

**Example 3: Multiple data streams (continued).** The HHMM is fitted via:

```
R> unemp_spx_model_3_2 <- fit_model(data_hhmm, seed = 1, ncluster = 25)
```

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

```
R> data("unemp_spx_model_3_2", package = "fHMM")
```

The estimates can be obtained through the `coef()` method as previously demonstrated. To maintain brevity, the output is omitted in this instance. We can visualize in Figure 7 the estimated state-dependent distributions at both the coarse-scale and fine-scale layer as follows:

```
R> plot(unemp_spx_model_3_2, plot_type = "sdds")
```

Coarse-scale state 1 (which is colored in red) captures periods of rising unemployment, which are linked to increased market volatility (as indicated by the conditional fine-scale distributions that are characterized by increased standard deviations). In contrast, when coarse-scale

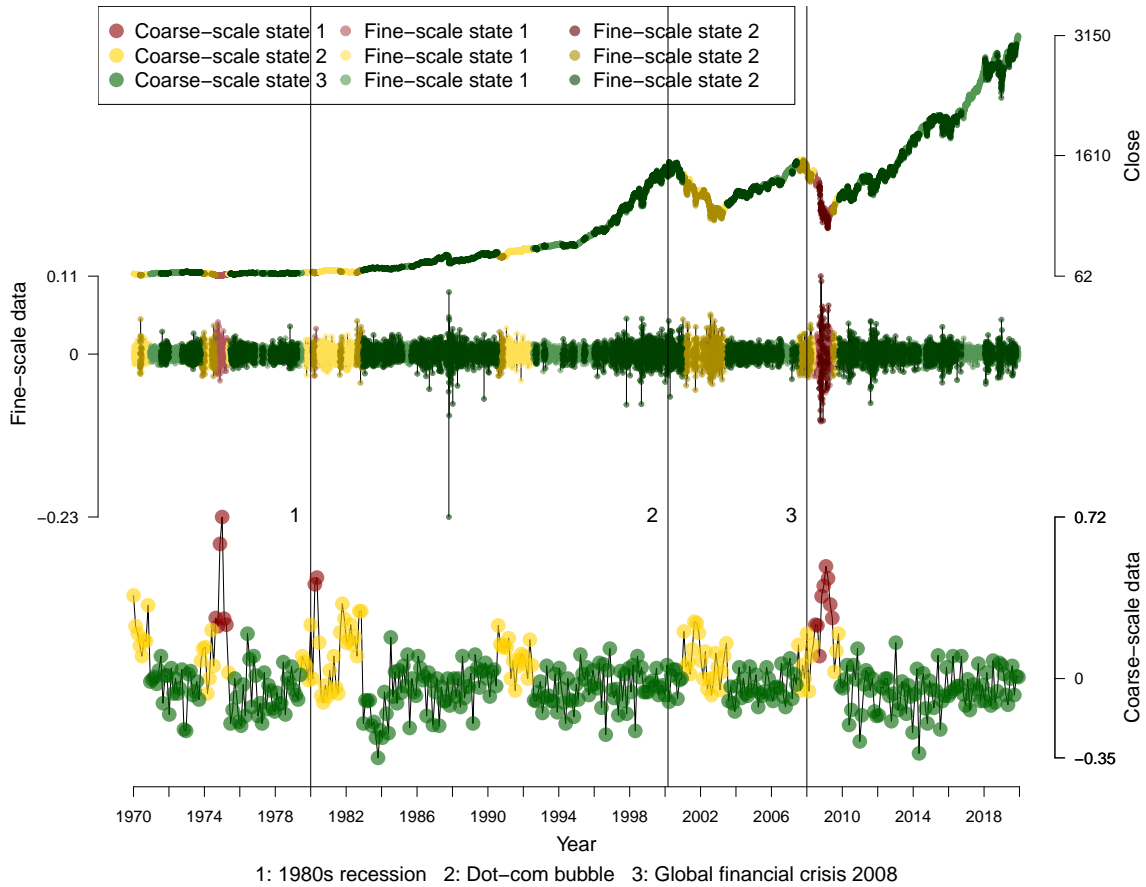


Figure 8: Decoded time series of S&P 500 daily closing prices from 1970 to 2020 (top panel) and their corresponding log-returns (middle panel). The bottom panel shows the decoded monthly changes in the US unemployment rate.

state 3 (which is colored in green) is active, then the unemployment rate tends to decrease, and the S&P 500 exhibits less volatility. These conclusions are confirmed by the decoded time series of daily closing prices in Figure 8, which are displayed in the top chart. The mathematical details behind decoding the hidden states are presented in the following section.

```
R> unemp_spx_model_3_2 <- decode_states(unemp_spx_model_3_2)
R> events_spx <- fHMM_events(list(
+   dates = c("1980-01-01", "2000-03-01", "2008-01-01"),
+   labels = c("1980s recession", "Dot-com bubble",
+   "Global financial crisis 2008")))
R> plot(unemp_spx_model_3_2, events = events_spx)
```

The 1980s recession impacted the S&P 500 and US unemployment rate, causing a decline in the index and a rise in unemployment. The late 1990s saw the S&P 500 rise due to tech growth, but eventually declined from 2000 to 2002 with relatively stable unemployment. The 2008 financial crisis resulted in a significant decline in the stock market and a peak in US unemployment in late 2009.



## 6. State decoding and prediction

For financial markets, it is of special interest to infer the hidden states in order to gain insight about the actual market situation and for prediction. The Viterbi algorithm (Forney 1973) is a recursive scheme that enables to find the most likely trajectory of hidden states under the estimated HMM. To this end, we follow Zucchini *et al.* (2016) and define

$$\begin{aligned}\zeta_{1i} &= \Pr(S_1 = i, X_1 = x_1) = \delta_i p_i(x_1) \\ \zeta_{ti} &= \max_{s_1, \dots, s_{t-1}} \Pr(S_{t-1} = s_{t-1}, S_t = i, X_t = x_t)\end{aligned}$$

for  $i = 1, \dots, N$  (the index for the states) and  $t = 2, \dots, T$  (the index of time). Then, the trajectory of most likely states  $i_1, \dots, i_T$  can be calculated recursively backwards from

$$\begin{aligned}i_T &= \operatorname{argmax}_{i=1, \dots, N} \zeta_{Ti} \\ i_t &= \operatorname{argmax}_{i=1, \dots, N} (\zeta_{ti} \gamma_{i, i_{t+1}}), \quad t = T - 1, \dots, 1.\end{aligned}$$

Transferring the state decoding to HHMMs is straightforward via decoding the coarse-scale states first and afterwards, by using this information, decoding the fine-scale state process, see Adam *et al.* (2019a).

In the following, we introduce the `decode_states()` function for state decoding and the `predict()` method for forecasting. As all of the functionalities of the **fHMM** package presented in the remainder of this paper are completely analogue for the hierarchical and the simulated case, respectively, we will focus our attention on example 1 and invite the reader to apply the methods to example 2 and 3 on their own.

**Example 1: DAX (continued).** The hidden states of the 3-state HMM for the DAX can be decoded via the `decode_states()` function, which updates an ‘`fHMM_model`’ object. The state sequence is saved as argument `dax_model_3t$decoding`:

```
R> dax_model_3t <- decode_states(dax_model_3t)
R> table(dax_model_3t$decoding)
```

```
 1    2    3
704 2926 2252
```

The decoded time series can then be visualized (see Figure 9) by using the `plot()` method:<sup>5</sup>

```
R> plot(dax_model_3t, events = events_dax)
```

Figure 9 displays that the first state can be interpreted as a bearish market, indicated by negative log-returns on average and a high volatility. In contrast, the third state is a proxy for a bull market, characterized by mostly positive log-returns and low volatility. The intermediate, second state serves as a transition state with balanced log-returns and moderate volatility (see, e.g., Lihn 2017).

<sup>5</sup>Notice that the model is invariant to permutations of the state labels. The **fHMM** package provides the option to switch labels after state decoding via the `reorder_states()` function. For example, `reorder_states(dax_model_3t, state_order = 3:1)` reverses the order.

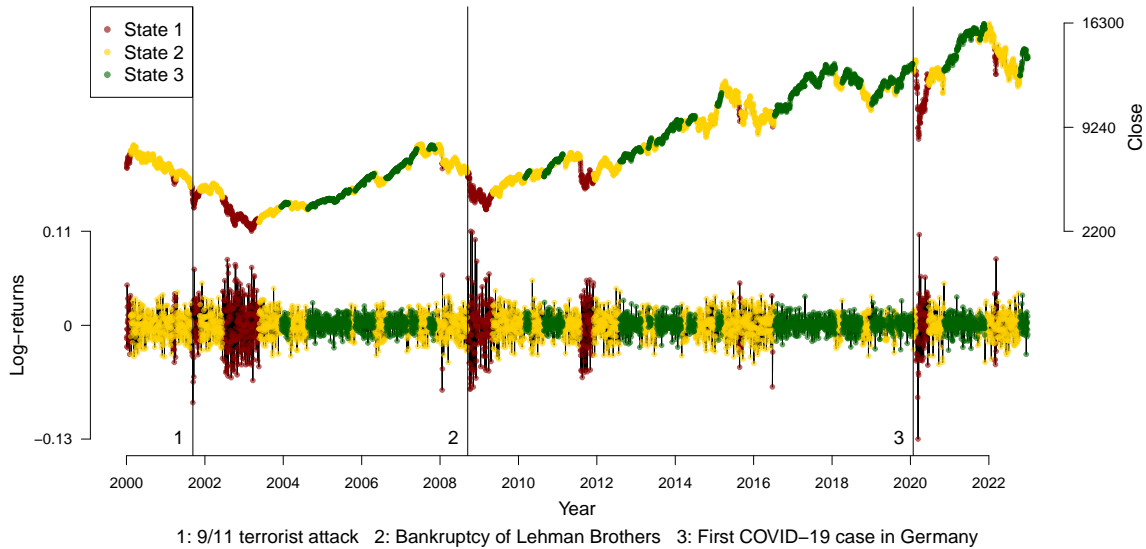


Figure 9: Decoded time series of the daily DAX prices and of the corresponding log-returns.

After decoded the hidden states, we use the estimated transition probabilities to compute the state probabilities of the subsequent observations. Based on these probabilities and in combination with the estimated state-dependent distributions, the subsequent observations can be predicted, compare [Zucchini \*et al.\* \(2016\)](#):

```
R> predict(dax_model_3t, ahead = 5)
```

	state_1	state_2	state_3	lb	estimate	ub
1	0.00000	0.02441	0.97559	-0.01065	0.00123	0.01312
2	0.00012	0.04773	0.95224	-0.01093	0.00120	0.01332
3	0.00036	0.06974	0.92990	-0.01120	0.00116	0.01352
4	0.00070	0.09077	0.90835	-0.01145	0.00113	0.01371
5	0.00114	0.11079	0.88807	-0.01171	0.00110	0.01390

Columns 1 to 3 show the state probabilities for the next `ahead = 5` trading days. The values in columns 4 to 6 (the bounds of the 95% confidence intervals and the point predictions) are log-returns, which obviously can be transformed to index values or relative returns.

## 7. Model checking

Checking whether a fitted model describes the data well is an essential part of any modeling process. In the HMM setting, this is typically done by analyzing the so-called pseudo-residuals ([Zucchini \*et al.\* 2016](#)). Since the observations are modeled by different distributions (depending on the active state), they have to be transformed on a common scale, which can be done as follows: If the observation  $X_t$  has the invertible distribution function  $F_{X_t}$ , then  $Z_t = \Phi^{-1}(F_{X_t}(X_t))$  is standard normally distributed, where  $\Phi$  denotes the cumulative distribution function of the standard normal distribution. The observations are modeled well if

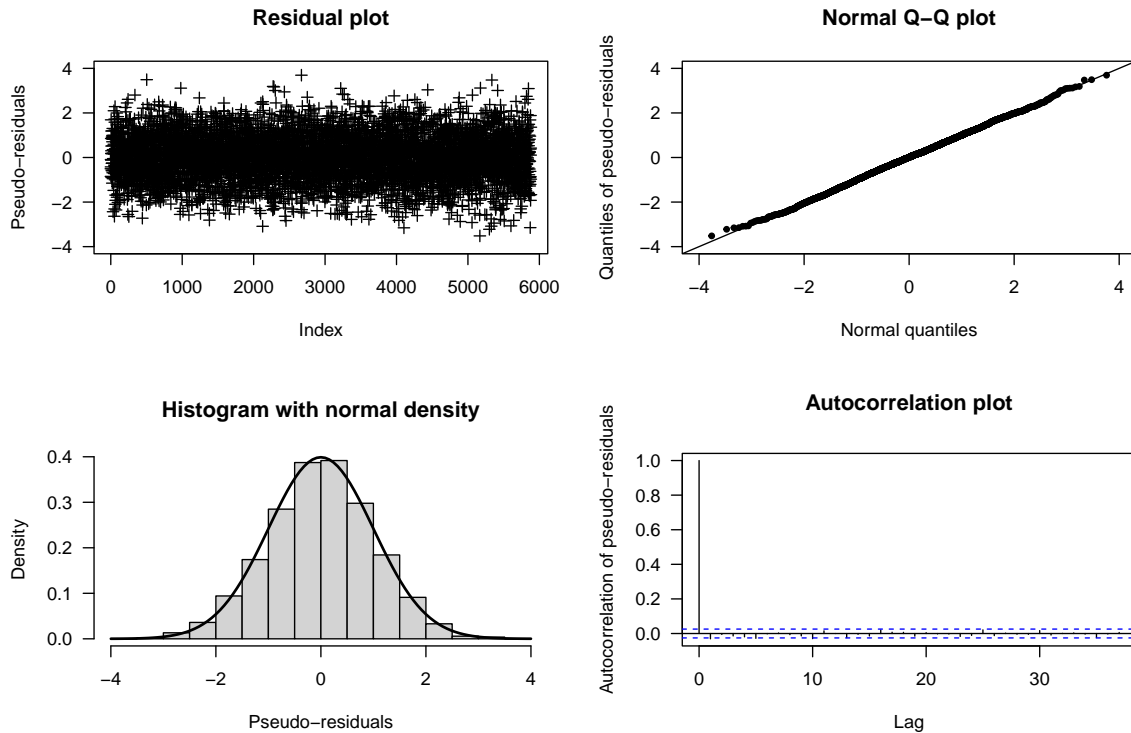


Figure 10: Diagnostic plots for the residuals in the DAX example.

the pseudo-residuals  $(Z_t)_t$  are approximately standard normally distributed and exhibit no autocorrelation. In the hierarchical case, 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.

**Example 1: DAX (continued).** Via the `compute_residuals()` function, pseudo-residuals can be computed (provided that the states have been decoded beforehand). The function updates the `dax_model_3t` object in the following line:

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

The normality and independence of the pseudo-residuals can be verified visually (see Figure 10):

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

Alternatively, the residuals can be extracted from the model object via the `residuals()` method for normality tests, for example a Jarque-Bera test (Jarque and Bera 1987). Here, the test is unable to reject the null hypothesis that the data are normally distributed.

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

## Jarque Bera Test

```
data: res
X-squared = 2.2705, df = 2, p-value = 0.3213
```

## 8. Model selection

Model selection involves the choice of a family for the state-dependent distributions and the selection of the number of states. Common model selection tools are information criteria, such as the Akaike information criterion (AIC, [Akaike 1974](#)) or the Bayesian information criterion (BIC, [Schwarz 1978](#)). They are defined as

$$\begin{aligned} \text{AIC} &= -2 \log \mathcal{L}^{(\text{H})\text{HMM}} + 2p; \\ \text{BIC} &= -2 \log \mathcal{L}^{(\text{H})\text{HMM}} + \log(T)p, \end{aligned}$$

where  $p$  denotes the number of model parameters and  $T$  is the number of observations.<sup>6</sup> Both criteria aim at finding a compromise between model fit and model complexity, where a model with a lower value is to be preferred. For an in-depth discussion of pitfalls, practical challenges, and pragmatic solutions regarding model selection, we refer to [Pohle, Langrock, Van Beest, and Schmidt \(2017\)](#).

**Example 1: DAX (continued).** We compare our 3-state HMM with state-dependent  $t$  distributions fitted to the DAX data to an HMM with 2 states and normal state-dependent distributions. We assume that the competing model was estimated using the same data and can be accessed as object `dax_model_2n`. The `compare_models()` function takes (arbitrarily many) ‘`fHMM_model`’ objects as input and returns the number of parameters, the log-likelihood value, the AIC, and the BIC. In this example, both AIC and BIC clearly prefer the more complex model:

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

	parameters	loglikelihood	AIC	BIC
<code>dax_model_2n</code>	6	17403.86	-34795.71	-34755.64
<code>dax_model_3t</code>	15	17650.30	-35270.61	-35170.41

## 9. Conclusions

The **fHMM** package aims at making HMMs accessible to R users with an interest in financial time series. It contains functions to download, prepare, and simulate data, to fit models, to decode the hidden states, to use a fitted model for state forecasting, to check the goodness of fit, and to perform model selection. The **fHMM** package has a user-friendly design: All model specifications are centralized in a list of controls, four different package objects can be

<sup>6</sup>In the hierarchical case,  $T$  equals the sum of coarse-scale and fine-scale observations.

seamlessly passed between functions, and its usage follows a clear workflow (see Figure 1). In this paper, we illustrated a typical workflow using three illustrating examples (applications to stock market data from the DAX and S&P 500 as well as a model fitted to simulated data) that serves as a starting point for R users who want to apply HMMs and their extensions to their own data. Current limitations of the **fHMM** package include (1) the confined set of available state-dependent distributions for the observations, (2) the restriction to a discrete state space, (3) the restriction to a discretized time dimension, (4) the limitation of a maximum number of two hierarchies in HHMMs, and (5) the need to specify the number of hidden states in advance of the model estimation. We aim to address these limitations in future package versions and welcome suggestions from the community for additional features to be implemented.

## Computational details

The results presented in this paper were obtained using R 4.4.0 with the **fHMM** 1.3.1 package. R itself and all packages used are available from the Comprehensive R Archive Network (CRAN) at <https://CRAN.R-project.org/>.

## Acknowledgments

We appreciate the editor and anonymous reviewer’s insightful feedback on our software implementation and article, and we are particularly thankful to the reviewer who suggested the application in Example 3.

## References

- Adam T (2019). **countHMM**: *Penalized Estimation of Flexible Hidden Markov Models for Time Series of Counts*. R package version 0.1.0, URL <https://CRAN.R-project.org/package=countHMM>.
- Adam T, Griffiths CA, Leos-Barajas V, Meese EN, Lowe CG, Blackwell PG, Righton D, Langrock R (2019a). “Joint Modelling of Multi-Scale Animal Movement Data Using Hierarchical Hidden Markov Models.” *Methods in Ecology and Evolution*, **10**(9), 1536–1550. doi:10.1111/2041-210x.13241.
- Adam T, Langrock R, Kneib T (2019b). “Model-Based Clustering of Time Series Data: A Flexible Approach Using Nonparametric State-Switching Quantile Regression Models.” In *Book of Short Papers of the 12th Scientific Meeting on Classification and Data Analysis*, pp. 8–11.
- 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.
- Adam T, Oelschläger L (2020). “Hidden Markov Models For Multi-Scale Time Series: An Application to Stock Market Data.” In *Proceedings of the 35th International Workshop on Statistical Modelling*, volume 1, pp. 2–7.

- Akaike H (1974). “A New Look at the Statistical Model Identification.” *IEEE Transactions on Automatic Control*, **19**. doi:10.1109/tac.1974.1100705.
- Bartolucci F, Pandolfi S, Pennoni F (2017). “**LMest**: An R Package for Latent Markov Models for Longitudinal Categorical Data.” *Journal of Statistical Software*, **81**(4), 1–38. doi:10.18637/jss.v081.i04.
- Bezanson J, Edelman A, Karpinski S, Shah VB (2017). “Julia: A Fresh Approach to Numerical Computing.” *SIAM Review*, **59**(1), 65–98. doi:10.1137/141000671.
- Bulla J, Bulla I (2006). “Stylized Facts of Financial Time Series and Hidden Semi-Markov Models.” *Computational Statistics & Data Analysis*, **51**(4), 2192–2209. doi:10.1016/j.csda.2006.07.021.
- Bulla J, Bulla I, Nenadić O (2010). “**hsmm** – An R Package for Analyzing Hidden Semi-Markov Models.” *Computational Statistics & Data Analysis*, **54**(3), 611–619. doi:10.1016/j.csda.2008.08.025.
- 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.
- Chen M (2022). *Hidden Markov Model Toolbox (HMM)*. MATLAB Central File Exchange. Retrieved July 6, 2022., URL <https://www.mathworks.com/matlabcentral/fileexchange/55866-hidden-markov-model-toolbox-hmm>.
- Dalle G (2024). “**HiddenMarkovModels.jl**: Generic, Fast and Reliable State Space Modeling.” *Journal of Open Source Software*, **9**(96). doi:10.21105/joss.06436.
- Forney GD (1973). “The Viterbi Algorithm.” *Proceedings of the IEEE*, **61**(3), 268–278. doi:10.1109/proc.1973.9030.
- Gregoir S, Lenglard 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. doi:10.1002/(sici)1099-131x(200003)19:2<81::aid-for734>3.0.co;2-1.
- Himmelmann L (2022). *HMM: Hidden Markov Models*. R package version 1.0.1, URL <http://CRAN.R-project.org/package=HMM>.
- Jackson C (2011). “Multi-State Models for Panel Data: The **msm** Package for R.” *Journal of Statistical Software*, **38**(8), 1–28. doi:10.18637/jss.v038.i08.
- Janßen B, Rudolph B (1992). *Der Deutsche Aktienindex DAX*. Knapp Verlag.
- Jarque CM, Bera AK (1987). “A Test For Normality of Observations and Regression Residuals.” *International Statistical Review*, **55**, 163–172. doi:10.2307/1403192.
- Kim CJ, Nelson CR (1998). “Business Cycle Turning Points, a New Coincident Index, and Tests of Duration Dependence Based on a Dynamic Factor Model with Regime Switching.” *Review of Economics and Statistics*, **80**(2), 188–201. doi:10.1162/003465398557447.

- Langrock R, Adam T, Leos-Barajas V, Mews S, Miller DL, Papastamatiou YP (2018). “Spline-Based Nonparametric Inference in General State-Switching Models.” *Statistica Neerlandica*, **72**(3), 179–200. doi:10.1111/stan.12133.
- Lebedex S (2022). *hmmlearn: Hidden Markov Models in Python, with scikit-Learn Like API*. Python library, version 0.2.7, URL <https://hmmlearn.readthedocs.io/>.
- Lihn SHT (2017). “Hidden Markov Model for Financial Time Series and Its Application to S&P 500 Index.” *Paper 2979516*, SSRN. URL [https://papers.ssrn.com/sol3/papers.cfm?abstract\\_id=2979516](https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2979516).
- Lihn SHT (2019). *ldhmm: Hidden Markov Model for Financial Time-Series Based on Lambda Distribution*. R package version 0.5.1, URL <https://CRAN.R-project.org/package=ldhmm>.
- McClintock BT, Michelot T (2018). “**momentuHMM**: R Package for Generalized Hidden Markov Models of Animal Movement.” *Methods in Ecology and Evolution*, **9**(6), 1518–1530. doi:10.1111/2041-210x.12995.
- Michelot T, Langrock R, Patterson TA (2016). “**moveHMM**: An R Package for the Statistical Modelling of Animal Movement Data Using Hidden Markov Models.” *Methods in Ecology and Evolution*, **7**(11), 1308–1315. doi:10.1111/2041-210x.12578.
- Nguyen N (2018). “Hidden Markov Model for Stock Trading.” *International Journal of Financial Studies*, **6**(2). doi:10.3390/ijfs6020036.
- Norris JR (1997). *Markov Chains*. Cambridge University Press. doi:10.1017/cbo9780511810633.
- 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.
- 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.
- O’Connell J, Højsgaard S (2011). “Hidden Semi Markov Models for Multiple Observation Sequences: The **mhsmm** Package for R.” *Journal of Statistical Software*, **39**, 1–22. doi:10.18637/jss.v039.i04.
- OECD (2023). “Unemployment Rate.” doi:10.1787/52570002-en. Accessed on 2023-01-18.
- Oelschläger L (2019). *Detection of Bearish and Bullish Markets in the DAX Using Hierarchical Hidden Markov Models*. Master’s thesis, Bielefeld University.
- Oelschläger L, Adam T (2021). “Detecting Bearish and Bullish Markets in Financial Time Series Using Hierarchical Hidden Markov Models.” *Statistical Modelling*, **23**(2), 107–126. doi:10.1177/1471082x211034048.
- Oelschläger L, Adam T, Michels R (2024). *fHMM: Fitting Hidden Markov Models to Financial Data*. R package version 1.3.1, URL <https://CRAN.R-project.org/package=fHMM>.

- Ondel L, Lam-Yee-Mui LM, Kocour M, Filippo C, Lukás Burget C (2021). “GPU-Accelerated Forward-Backward Algorithm with Application to Lattice-Free MMI.” In *IEEE International Conference on Acoustics, Speech, and Signal Processing*. Singapore. URL <https://hal.archives-ouvertes.fr/hal-03434552>.
- Platen E, Rendek R (2008). “Empirical Evidence on Student  $t$  Log-Returns of Diversified World Stock Indices.” *Journal of Statistical Theory and Practice*, **2**. doi:10.1080/15598608.2008.10411873.
- Pohle J, Langrock R, Van Beest FM, Schmidt NM (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. doi:10.1007/s13253-017-0283-8.
- R Core Team (2024). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL <https://www.R-project.org/>.
- Schwarz G (1978). “Estimating the Dimension of a Model.” *The Annals of Statistics*, **6**. doi:10.1214/aos/1176344136.
- Shireman E, Steinley D, Brusco MJ (2017). “Examining the Effect of Initialization Strategies on the Performance of Gaussian Mixture Modeling.” *Behavior Research Methods*, **49**(1), 282–293. doi:10.3758/s13428-015-0697-6.
- StataCorp (2019). *Stata: Statistical Software: Release 16*. StataCorp LLC, College Station. URL <https://www.stata.com/>.
- The MathWorks Inc (2021). *MATLAB – The Language of Technical Computing, Version R2021a*. Natick. URL <https://www.mathworks.com/products/matlab/>.
- Turner R (2022). **hmm.discnp**: *Hidden Markov Models with Discrete Non-Parametric Observation Distributions*. R package version 3.0-9, URL <http://CRAN.R-project.org/package=hmm.discnp>.
- Van Rossum G, et al. (2011). *Python Programming Language*. URL <https://www.python.org/>.
- Visser I, Speekenbrink M (2010). “**depmixS4**: An R Package for Hidden Markov Models.” *Journal of Statistical Software*, **36**, 1–21. doi:10.18637/jss.v036.i07.
- Zucchini W, MacDonald IL, Langrock R (2016). *Hidden Markov Models for Time Series: An Introduction Using R*. 2nd edition. Chapman & Hall/CRC. doi:10.1201/b20790.

### Affiliation:

Lennart Oelschläger, Rouven Michels  
 Department of Business Administration and Economics  
 Bielefeld University  
 Postfach 10 01 31, Germany  
 E-mail: [lennart.oelschlaeger@uni-bielefeld.de](mailto:lennart.oelschlaeger@uni-bielefeld.de), [r.michels@uni-bielefeld.de](mailto:r.michels@uni-bielefeld.de)



Timo Adam  
School of Mathematics and Statistics  
University of St Andrews  
The Observatory, Buchanan Gardens, St Andrews KY16 9LZ, United Kingdom  
E-mail: [ta59@st-andrews.ac.uk](mailto:ta59@st-andrews.ac.uk)