# Written exercise 3

Assignment 3 – 02433 Hidden Markov Models – Anders Hørsted (s082382)

In this exercise a data set containing 10000 measurements of the power production at the danish wind farm "Klim" is analysed. Various time series models are fitted to the first 8000 measurements, and the predictive ability of the models are assessed using the remaining 2000 measurements.

The first 8000 measurements are called the training dataset and the remaining 2000 measurements are the test dataset. The summary output for the training dataset is

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      0    4242    9228    9615   15220   19690
```

And the training dataset is plotted in figure 1

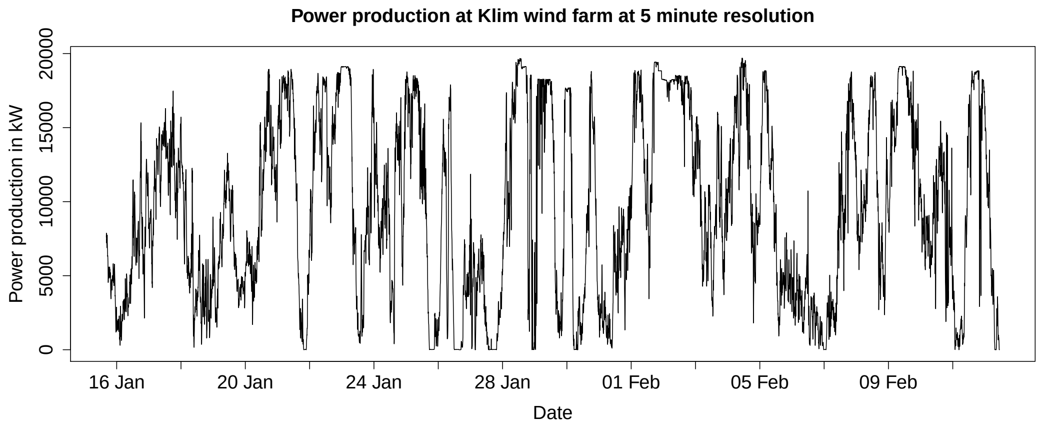


*Figure 1: Plot of the training dataset showing the power production at the danish wind farm "Klim". The measurement at time t is the total power production in the last 5 minutes up to time t*

A few things are noticed from the plot. First the minimum production of the wind farm is of cause 0 kW and the maximum production is told to be 21000 kW, and therefore the time series data is bounded. Also it is seen from the plot that the data seems to come from a few different time series with different moments, and that the shifts between the different time series happens over a short time period. As mentioned in the exercise description these shifts is due to changes in the meterological conditions. A natural way to model these shifts in conditions is by fitting a hidden markov model where each state dependent distribution is given by an ARIMA-process. These models are quite complex and before looking at these models a few simpler models are fitted to the data. First the exponential smoothing framework advocated in eg. [2] is used.

## The Exponential smoothing framework

Exponential smoothing procedures are procedures that gives point forecasts for time series. In [2] it is shown how various exponential smoothing procedures are obtained as the one-step prediction in a family of state space models. The R package `forecast` includes the function `ets` that given a time series finds the optimal exponential smoothing procedure and estimates the smoothing parameters. By running this function on the training data set the following result is obtained.

```
ETS(A,N,N)

Call:
 ets(y = power.production)

  Smoothing parameters:
    alpha = 0.9999

  Initial states:
    l = 7598.9915

  sigma:  900.7497

     AIC     AICc      BIC
180753.2 180753.2 180767.2
```

The line `ETS(A,N,N)` means that the best model have additive errors and no trend and seasonal component. The found procedure is therefore the simple exponential smoothing (see eg. [3]). From the result it is seen the smoothing constant is found as $\alpha = 0.9999 \approx 1$. As explained on page 41 in [2] the found model is given by

$$y_t = y_{t-1} + e_t$$

where $e_t$ is white noise. Therefore the found model is just the random walk process and the one-step prediction is then given by

$$\widehat{y}_t = y_{t-1}$$

.

The one-step prediction error for the test data set is easily found by

$$
\begin{aligned}
R &= \sqrt{\frac{1}{T-1}\sum_{t=1}^{T-1}(Y_{t+1}-\widehat{Y}_{t+1})^2} \\
&= \sqrt{\frac{1}{T-1}\sum_{t=1}^{T-1}(Y_{t+1}-Y_t)^2}
\end{aligned}
$$

with $T = 2000$. In R the one-step prediction error is found as

$$R_{\text{RandomWalk}} = 921.33$$

Using this as our baseline result ARIMA models for the dataset is found next.

## Fitting ARIMA models

In this section ARIMA models for the power production dataset is found. The ACF and PACF for the training dataset are shown in figure 2. The slow and almost linear decay of the ACF indicates that the time series is nonstationary (page 125 in [1]) and differencing is needed.

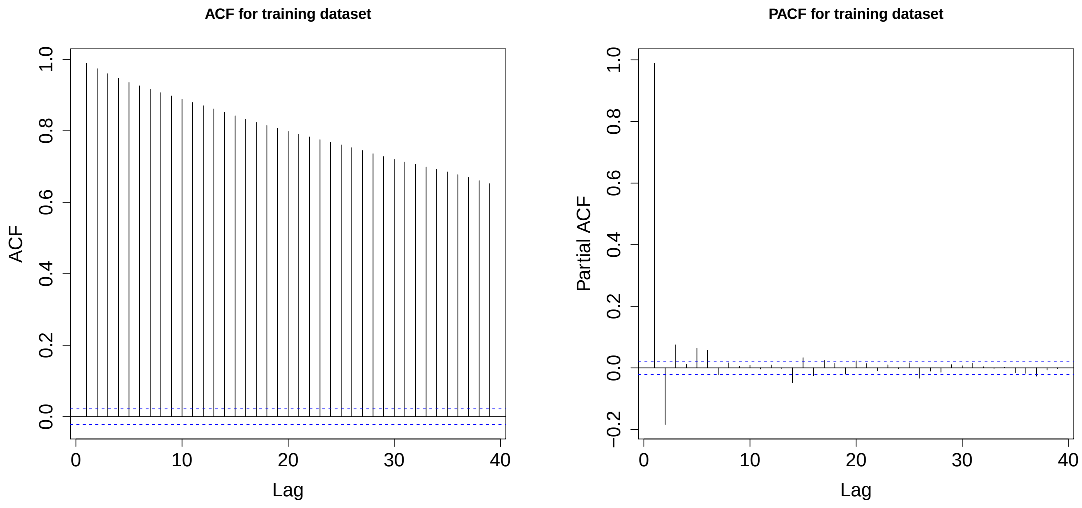


*Figure 2: ACF and PACF for the training dataset*

The difference of the dataset is calculated and the ACF and PACF for the differenced dataset is shown in figure 3. Both the ACF and the PACF can be interpreted as damped sine functions and the differenced data should probably be modelled as an ARMA process. After some trial and error an ARMA(1,2) is found to be acceptable.

The choice of model is further supported by running the automatic model detection function `auto.arima` in R. The `auto.arima` function gives the output

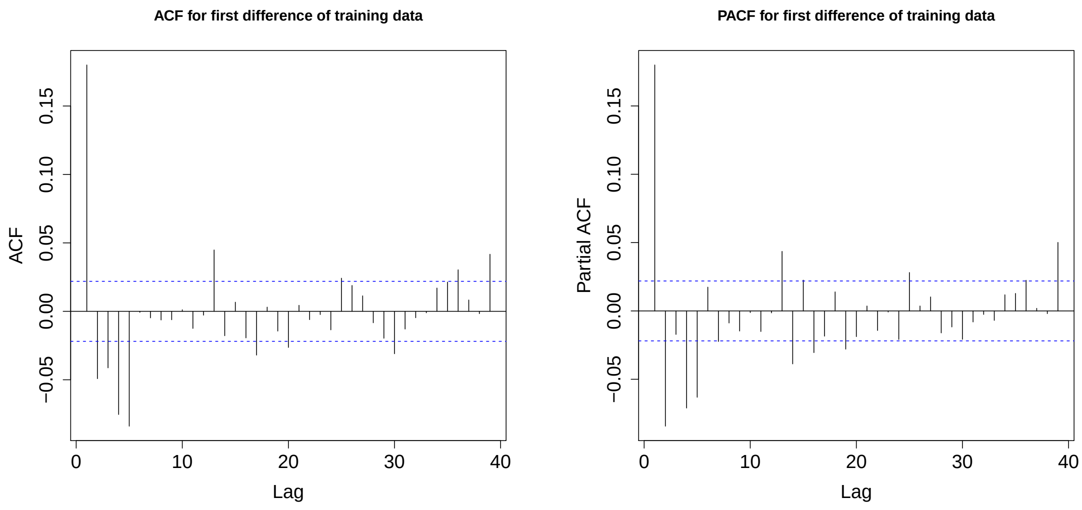


*Figure 3: ACF and PACF for the first difference of the training dataset*

```
Series: power.production
ARIMA(1,1,2)

Coefficients:
         ar1      ma1      ma2
      0.7957  -0.6087  -0.2128
s.e.  0.0402   0.0403   0.0111

sigma^2 estimated as 775023:  log likelihood=-65585.94
AIC=131179.9   AICc=131179.9   BIC=131207.8
```

which shows an ARIMA(1,1,2) model given by

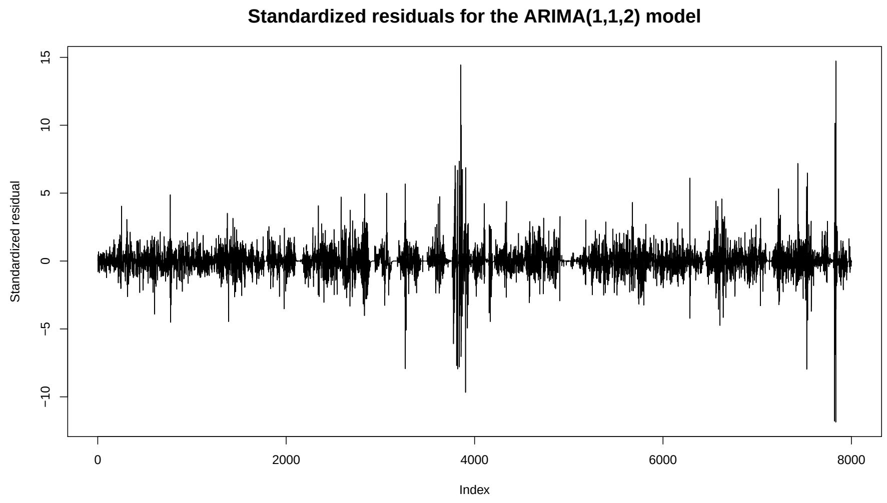
$$\nabla y_t = 0.80\nabla y_{t-1} - 0.61e_{t-1} - 0.21e_{t-2} + e_t$$

It is worth noticing that R is not including a constant when fitting differenced series. So if the differenced series is drifting the obtained fit is wrong. In the case of the bounded power production dataset this shouldn't be a problem though. See the website [4] for details.

To check the obtained ARIMA(1,1,2) model the ACF for the standardized residuals is plotted and shown in figure 4. There are too many large residuals as well as small intervals with very small residuals. The residuals that neither large nor small do actually resemble white noise. This is confirmed in the QQ-plot of the residuals shown in figure 4. Even though the model isn't adequate the one-step prediction error is calculated and gives

$$R_{\text{ARIMA(1,1,2)}} = 874.73$$

This is a large improvement compared with the random walk model in the previous section.



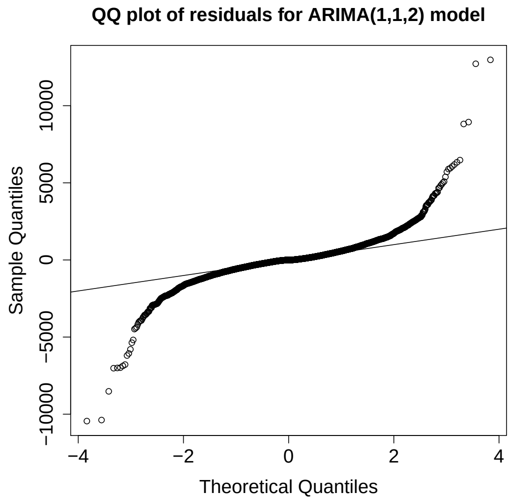


*Figure 4: ACF for the standardized residuals and a QQ-plot for the fitted ARIMA(1,1,2) model*

## Fitting HMM models

From the plot of the residuals for the ARIMA model found in the previous section it is seen that there are many residuals much larger than expected for a well fit model. These large residuals is a result of the varying characteristic of the time series for different meteorological conditions. These different characteristics are naturally modelled by a Hidden Markov Model with each state corresponding to a different characteristic.

Even though the power production measurements are only integers they will be treated as continuous measurements. The state dependent distribution is therefore chosen as a normal distribution. For each state $i = 1, 2, \ldots, m$ we then have

$$Y_t \,|\, C_t = i \;\sim\; N(\mu_i, \sigma_i^2)$$

The parameters that need to be estimated are then

$$\boldsymbol{\mu} = \begin{pmatrix} \mu_1 & \mu_2 & \cdots & \mu_m \end{pmatrix} \quad \boldsymbol{\sigma^2} = \begin{pmatrix} \sigma_1^2 & \sigma_2^2 & \cdots & \sigma_m^2 \end{pmatrix} \quad \mathbf{\Gamma} = \begin{pmatrix} \gamma_{11} & \cdots & \gamma_{1m} \\ \vdots & \ddots & \vdots \\ \gamma_{m1} & \cdots & \gamma_{mm} \end{pmatrix}$$

giving a total of $m(m+1)$ paramters to estimate (since each row in $\mathbf{\Gamma}$ sum to 1).

## 2-state normal HMM

First a 2-state HMM is fitted to the data by direct maximization of the likelihood function. Due to problems with unbounded liieklihood values, a discrete likelihood value was calculated as

$$p_i(y_t) = \Phi(y_t + 0.5; \mu_i, \sigma_i^2) - \Phi(y_t - 0.5; \mu_i, \sigma_i^2)$$

where $\Phi$ is the normal distribution function. The initial values for the parameters was

$$\boldsymbol{\mu_0} = \begin{pmatrix} 4.00 \cdot 10^{03} \\ 9.00 \cdot 10^{03} \end{pmatrix} \quad \boldsymbol{\sigma_0^2} = \begin{pmatrix} 2.00 \cdot 10^{03} \\ 4.00 \cdot 10^{03} \end{pmatrix} \quad \mathbf{\Gamma_0} = \begin{pmatrix} 9.00 \cdot 10^{-01} & 1.00 \cdot 10^{-01} \\ 1.00 \cdot 10^{-01} & 9.00 \cdot 10^{-01} \end{pmatrix}$$

which gave the estimates

$$\widehat{\boldsymbol{\mu}} = \begin{pmatrix} 4.87 \cdot 10^{03} \\ 1.55 \cdot 10^{04} \end{pmatrix} \quad \widehat{\boldsymbol{\sigma}}^{\mathbf{2}} = \begin{pmatrix} 3.15 \cdot 10^{03} \\ 2.80 \cdot 10^{03} \end{pmatrix} \quad \widehat{\mathbf{\Gamma}} = \begin{pmatrix} 9.90 \cdot 10^{-01} & 1.04 \cdot 10^{-02} \\ 1.29 \cdot 10^{-02} & 9.87 \cdot 10^{-01} \end{pmatrix}$$

Confidence intervals for the estimated parameters can be found in appendix A.1. The various model performance measures were

$$-\ell = 75745.694 \quad \text{AIC} = 151507.388 \quad \text{BIC} = 151563.286$$

To get an idea of the 2 state dependent distributions a plot of them is shown in figure 5. It is seen how the 2 state dependent distributions cover all possible values and that the variance of the two state distributions are quite similar. Also a non neglectable amount of the probability mass is outside the allowed interval of [0; 21000]. To get an idea of what states the various data points are related to a local decoding was calculated for the training dataset. The decoding is shown in figure 6

From the local decoding it seems natural to at least add one “middle” state. A 3-state HMM is therefore fitted to the data.

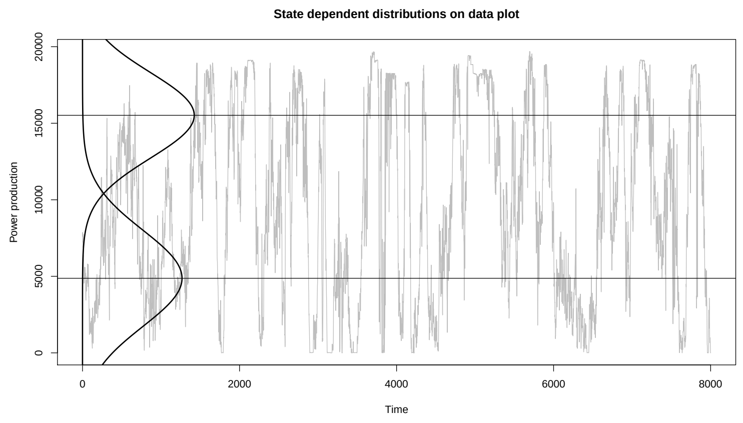


*Figure 5: Plot of state dependent distributions for the 2-state HMM.*

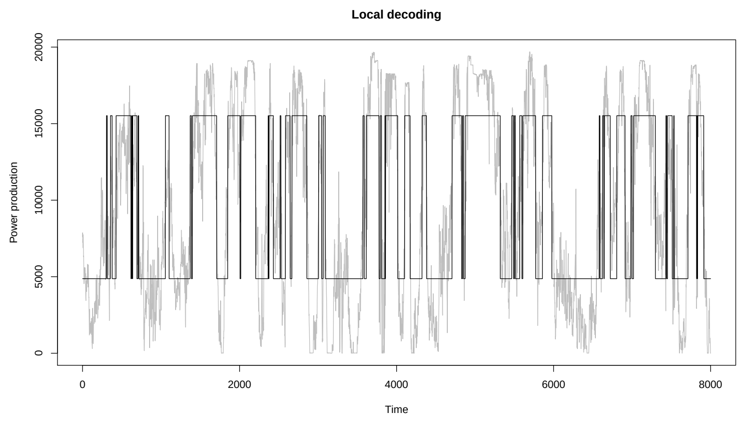


*Figure 6: Local decoding for the 2-state HMM*

## One step prediction errors

The interesting part of the model checking is to compare the forecast error of the various models. Unfortunately I had major problems with calculating one-step predictions from the HMMs. For some reason all one step prediction distributions turned out to be the same. This would naturally lead to the same one step predictions for all $t$ which wasn't expected. The reason could be that the log of the components of the forward probabilities $\boldsymbol{\alpha}_t$ are numerically so large (-70000 for $t = 8000$) for large $t$, that even though the absolute changes between $\boldsymbol{\alpha}_t$ and $\boldsymbol{\alpha}_{t+1}$ are noticeable, the change in

$$\boldsymbol{\phi}_t = \frac{\boldsymbol{\alpha}_t}{\boldsymbol{\alpha}_t \mathbf{1}'}$$

will be small. Combined with the transition probability matrix $\boldsymbol{\Gamma}$ being close to the identity matrix, makes the one step predictions

$$\Pr(Y_{t+1} = y \,|\, \boldsymbol{Y}^{(t)} = \boldsymbol{x}^{(t)}) = \boldsymbol{\phi}_t \boldsymbol{\Gamma} \boldsymbol{P}(y) \mathbf{1}'$$

almost identical, for all large $t$, if $\boldsymbol{\phi}_t \approx \boldsymbol{\phi}_{t+1}$.

That could explain the constant one step prediction distribution. Another explaination could be that I have made a coding error or have misunderstood something. The bottom line is that one step prediction errors for the HMM models isn't included in this report.

## 3-state normal HMM

A 3-state HMM is fitted to the training data. The initial values for the parameters are

$$\boldsymbol{\mu_0} = \begin{pmatrix} 1.60 \cdot 10^{04} \\ 1.00 \cdot 10^{04} \\ 4.00 \cdot 10^{03} \end{pmatrix} \quad \boldsymbol{\sigma_0^2} = \begin{pmatrix} 1.80 \cdot 10^{03} \\ 1.80 \cdot 10^{03} \\ 2.00 \cdot 10^{03} \end{pmatrix} \quad \boldsymbol{\Gamma_0} = \begin{pmatrix} 9.00 \cdot 10^{-01} & 5.00 \cdot 10^{-02} & 5.00 \cdot 10^{-02} \\ 5.00 \cdot 10^{-02} & 9.00 \cdot 10^{-01} & 5.00 \cdot 10^{-02} \\ 5.00 \cdot 10^{-02} & 5.00 \cdot 10^{-02} & 9.00 \cdot 10^{-01} \end{pmatrix}$$

Using the initial values gives these estimates (confidence intervals in appendix A.1).

$$\widehat{\boldsymbol{\mu}} = \begin{pmatrix} 1.78 \cdot 10^{04} \\ 1.10 \cdot 10^{04} \\ 3.24 \cdot 10^{03} \end{pmatrix} \quad \widehat{\boldsymbol{\sigma}}^{\mathbf{2}} = \begin{pmatrix} 1.14 \cdot 10^{03} \\ 2.60 \cdot 10^{03} \\ 2.13 \cdot 10^{03} \end{pmatrix} \quad \widehat{\boldsymbol{\Gamma}} = \begin{pmatrix} 9.76 \cdot 10^{-01} & 2.42 \cdot 10^{-02} & 4.63 \cdot 10^{-08} \\ 1.57 \cdot 10^{-02} & 9.68 \cdot 10^{-01} & 1.59 \cdot 10^{-02} \\ 9.99 \cdot 10^{-09} & 1.49 \cdot 10^{-02} & 9.85 \cdot 10^{-01} \end{pmatrix}$$

The fitted model now have a state for "low", "medium" and "high" production. The state for high production has a smaller variance than the two other states. It is also seen that shifts directly from low to high or from high to low, is very unlikely to happen ($\gamma_{13}, \gamma_{31} \ll 1$).

To get an idea of how the state dependent distributions are shaped a plot of the three distributions is shown in figure 7 and a local decoding for the training data is shown in figure 8

Both plots seems to support that 3 states captures the structure in the data better than 2 states. To numerically support that claim, the log likelihood, AIC and BIC are calculated, and gives

$$-\ell = 72765.754 \quad \text{AIC} = 145561.508 \quad \text{BIC} = 145666.316$$

which shows a rather large improvement, compared with the 2-state model. Although the 3-state model seems to fit the data well, the middle state could probably be split into two states and the mean of the lower state be decreased. This leads to a 4-state HMM.

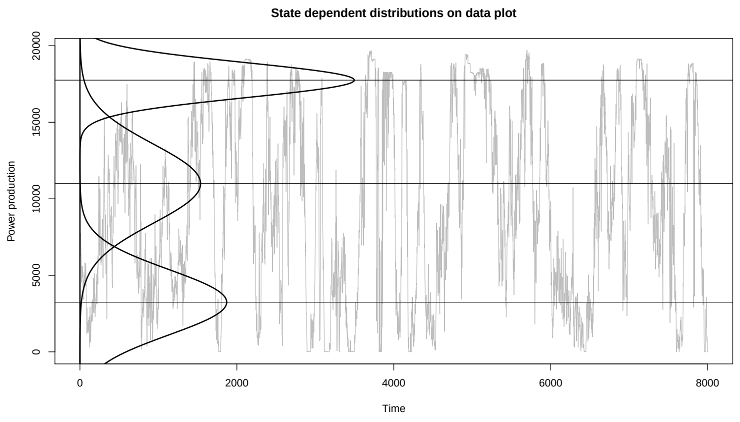


*Figure 7: State dependent distributions for the 3-state HMM*

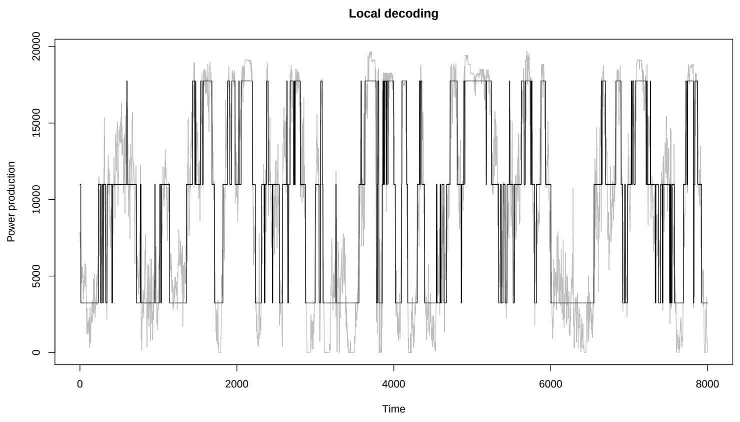


*Figure 8: Local decoding for the 3-state HMM*

## 4-state normal HMM

A 4-state HMM is fitted by direct maximization of the likelihood, using initial values given by

$$\boldsymbol{\mu_0} = \begin{pmatrix} 1.80 \cdot 10^{04} \\ 1.40 \cdot 10^{04} \\ 8.00 \cdot 10^{03} \\ 2.00 \cdot 10^{03} \end{pmatrix} \quad \boldsymbol{\sigma_0^2} = \begin{pmatrix} 1.20 \cdot 10^{03} \\ 1.20 \cdot 10^{03} \\ 1.20 \cdot 10^{03} \\ 1.00 \cdot 10^{03} \end{pmatrix} \quad \boldsymbol{\Gamma_0} = \begin{pmatrix} 9.10 \cdot 10^{-01} & 3.00 \cdot 10^{-02} & 3.00 \cdot 10^{-02} & 3.00 \cdot 10^{-02} \\ 3.00 \cdot 10^{-02} & 9.10 \cdot 10^{-01} & 3.00 \cdot 10^{-02} & 3.00 \cdot 10^{-02} \\ 3.00 \cdot 10^{-02} & 3.00 \cdot 10^{-02} & 9.10 \cdot 10^{-01} & 3.00 \cdot 10^{-02} \\ 3.00 \cdot 10^{-02} & 3.00 \cdot 10^{-02} & 3.00 \cdot 10^{-02} & 9.10 \cdot 10^{-01} \end{pmatrix}$$

which gives the parameter estimates. In general the variance of the state dependent distribtions are smaller than for the 2- and 3-state models. Also the maximum likelihood procedure seems to agree that the middle state in the 3-state model should be split in two. Like the 2- and 3-state models, the probability of staying in the same state is high and the probability of “skipping” a state is much lower than going to a neighbour state.

$$\widehat{\boldsymbol{\mu}} = \begin{pmatrix} 1.81 \cdot 10^{04} \\ 1.28 \cdot 10^{04} \\ 6.35 \cdot 10^{03} \\ 1.65 \cdot 10^{03} \end{pmatrix} \quad \widehat{\boldsymbol{\sigma}}^{\mathbf{2}} = \begin{pmatrix} 7.99 \cdot 10^{02} \\ 2.18 \cdot 10^{03} \\ 1.78 \cdot 10^{03} \\ 1.22 \cdot 10^{03} \end{pmatrix} \quad \widehat{\boldsymbol{\Gamma}} = \begin{pmatrix} 9.68 \cdot 10^{-01} & 3.17 \cdot 10^{-02} & 1.03 \cdot 10^{-04} & 9.08 \cdot 10^{-05} \\ 2.18 \cdot 10^{-02} & 9.54 \cdot 10^{-01} & 2.38 \cdot 10^{-02} & 1.75 \cdot 10^{-05} \\ 1.66 \cdot 10^{-05} & 2.44 \cdot 10^{-02} & 9.54 \cdot 10^{-01} & 2.15 \cdot 10^{-02} \\ 6.01 \cdot 10^{-05} & 1.21 \cdot 10^{-04} & 2.82 \cdot 10^{-02} & 9.72 \cdot 10^{-01} \end{pmatrix}$$

The state dependent distributions are plotted in figure 9 and the local decoding of the training data can be seen in figure 10

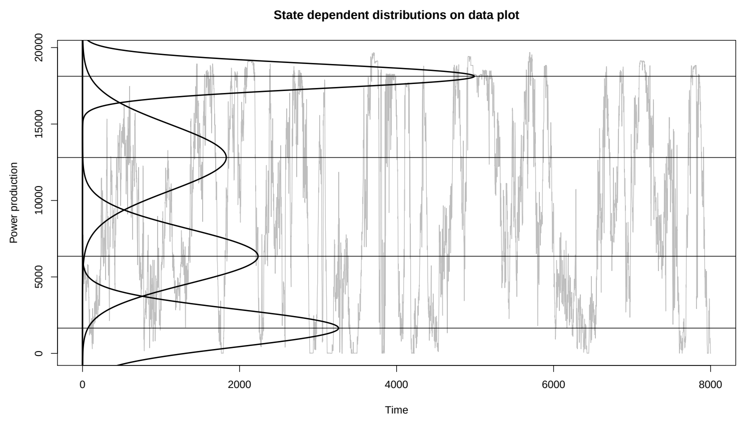


*Figure 9: State dependent distributions for the 4-state HMM*

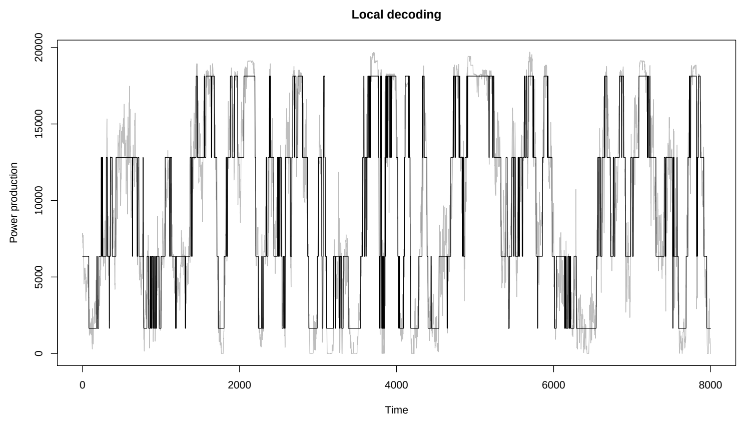


*Figure 10: Local decoding for the 4-state HMM*

From the plots, it seems to be the case that the 4-state model further improves the model fit compared with the 3-state model. This is supported by the likelihood value, the AIC and the BIC score

$$-\ell = 70901.060 \quad \text{AIC} = 141850.119 \quad \text{BIC} = 142017.812$$

## Conclusion

I underestimated the amount of work in this exercise. I would also have fitted HMM with AR processes as state dependent processes, but the time ran out. The conclusion is that the best model found is the 4-state HMM with normal distributed state dependent distributions.

## A Appendices

All R code created for this assignment is included here. All source code incl. latex code for this report can be found at `https://github.com/alphabits/dtu-spring-2012/tree/master/02433/assignment-3`

### A.1 Confidence intervals for parameter estimates

| | Estimate | 95% Conf.Int | | Std.Dev |
|---|---|---|---|---|
| $\mu_1$ | 4870.8987 | 4732.4981 | 5009.2993 | 70.6126 |
| $\mu_2$ | 15518.8414 | 15366.4029 | 15671.2800 | 77.7748 |
| $\sigma_1$ | 3148.0966 | 3057.0146 | 3239.1786 | 46.4704 |
| $\sigma_2$ | 2803.8226 | 2698.8129 | 2908.8323 | 53.5764 |
| $\gamma_{11}$ | 0.9896 | 0.9865 | 0.9927 | 0.0016 |
| $\gamma_{12}$ | 0.0129 | 0.0091 | 0.0168 | 0.0020 |
| $\gamma_{21}$ | 0.0104 | 0.0073 | 0.0135 | 0.0016 |
| $\gamma_{22}$ | 0.9871 | 0.9832 | 0.9909 | 0.0020 |

*Table 1: Parameter estimates with 95% confidence intervals calculated from the inverse hessian at the maximum*

| | Estimate | 95% Conf.Int | | Std.Dev |
|---|---|---|---|---|
| $\mu_1$ | 17752.1945 | 17685.8922 | 17818.4967 | 33.8277 |
| $\mu_2$ | 10990.9932 | 10850.3124 | 11131.6740 | 71.7759 |
| $\mu_3$ | 3242.9870 | 3141.1525 | 3344.8214 | 51.9564 |
| $\sigma_1$ | 1140.9292 | 1088.5064 | 1193.3520 | 26.7463 |
| $\sigma_2$ | 2595.1943 | 2512.5841 | 2677.8045 | 42.1481 |
| $\sigma_3$ | 2133.0879 | 2064.8555 | 2201.3203 | 34.8125 |
| $\gamma_{11}$ | 0.9758 | 0.9687 | 0.9830 | 0.0037 |
| $\gamma_{12}$ | 0.0157 | 0.0110 | 0.0204 | 0.0024 |
| $\gamma_{13}$ | 0.0000 | -0.0000 | 0.0000 | 0.0000 |
| $\gamma_{21}$ | 0.0242 | 0.0170 | 0.0313 | 0.0037 |
| $\gamma_{22}$ | 0.9684 | 0.9618 | 0.9751 | 0.0034 |
| $\gamma_{23}$ | 0.0149 | 0.0104 | 0.0193 | 0.0023 |
| $\gamma_{31}$ | 0.0000 | -0.0000 | 0.0000 | 0.0000 |
| $\gamma_{32}$ | 0.0159 | 0.0112 | 0.0206 | 0.0024 |
| $\gamma_{33}$ | 0.9851 | 0.9807 | 0.9896 | 0.0023 |

*Table 2: Parameter estimates with 95% confidence intervals calculated from the inverse hessian at the maximum*

| | Estimate | 95% Conf.Int | | Std.Dev |
|---|---|---|---|---|
| $\mu_1$ | 18125.7563 | 18074.6042 | 18176.9084 | 26.0980 |
| $\mu_2$ | 12809.6717 | 12677.6285 | 12941.7148 | 67.3690 |
| $\mu_3$ | 6352.5029 | 6183.4159 | 6521.5900 | 86.2689 |
| $\mu_4$ | 1650.2441 | 1522.5537 | 1777.9345 | 65.1482 |
| $\sigma_1$ | 799.4026 | 757.0832 | 841.7220 | 21.5915 |
| $\sigma_2$ | 2179.0277 | 2099.0645 | 2258.9909 | 40.7975 |
| $\sigma_3$ | 1784.4378 | 1712.4125 | 1856.4632 | 36.7476 |
| $\sigma_4$ | 1223.6234 | 1140.5388 | 1306.7080 | 42.3901 |
| $\gamma_{11}$ | 0.9681 | 0.9592 | 0.9770 | 0.0045 |
| $\gamma_{12}$ | 0.0218 | 0.0157 | 0.0279 | 0.0031 |
| $\gamma_{13}$ | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| $\gamma_{14}$ | 0.0001 | -0.0003 | 0.0004 | 0.0002 |
| $\gamma_{21}$ | 0.0317 | 0.0228 | 0.0406 | 0.0045 |
| $\gamma_{22}$ | 0.9544 | 0.9455 | 0.9633 | 0.0046 |
| $\gamma_{23}$ | 0.0244 | 0.0179 | 0.0310 | 0.0033 |
| $\gamma_{24}$ | 0.0001 | 0.0001 | 0.0001 | 0.0000 |
| $\gamma_{31}$ | 0.0001 | -0.0004 | 0.0006 | 0.0003 |
| $\gamma_{32}$ | 0.0238 | 0.0173 | 0.0303 | 0.0033 |
| $\gamma_{33}$ | 0.9541 | 0.9448 | 0.9634 | 0.0047 |
| $\gamma_{34}$ | 0.0282 | 0.0202 | 0.0362 | 0.0041 |
| $\gamma_{41}$ | 0.0001 | -0.0003 | 0.0005 | 0.0002 |
| $\gamma_{42}$ | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| $\gamma_{43}$ | 0.0215 | 0.0150 | 0.0279 | 0.0033 |
| $\gamma_{44}$ | 0.9716 | 0.9636 | 0.9796 | 0.0041 |

*Table 3: Parameter estimates with 95% confidence intervals calculated from the inverse hessian at the maximum*

## A.2 Load data

```
measurements.pr.day = 288
num.training.samples = 8000

dat = read.csv('data/myklimdata-noheader.csv', header=FALSE)

data.set.size = length(dat[,1])
num.test.samples = data.set.size - num.training.samples

train.idx = 1:num.training.samples
test.idx = (num.training.samples+1):data.set.size

dat.train = dat[train.idx,]
dat.test = dat[test.idx,]

times = dat.train[,1]
day.of.year = dat.train[,2]
dates = as.Date("2002-01-01") + day.of.year
```

```
power.production = dat.train[,3]

times.test = dat.test[,1]
day.of.year.test = dat.test[,2]
dates.test = as.Date("2002-01-01") + day.of.year.test
power.production.test = dat.test[,3]

day.shifts = which(diff(ceiling(day.of.year))==1)
first.day.shift = day.shifts[1]
first.day.shift.test = which(diff(ceiling(day.of.year.test))==1)[1]
```

## A.3 Data plot

```
source('src/loaddata.R')
source('src/functions.R')

SAVEPLOTS = TRUE


plot.trainingdata = function (idx, ...) {
    first.day.shift = day.shifts[which(day.shifts>=idx[1])[1]]
    date.idx = seq(first.day.shift, idx[length(idx)], by=2*measurements.pr.day)
    plot(dates[idx], power.production[idx], xlab="Date", ylab="Power production in kW",
         type="l", main="Power production at Klim wind farm at 5 minute resolution",
         xaxt="n", ...)
    axis(1, at=dates[date.idx], labels=format(dates[date.idx], "%d %b"),
         cex.axis=list(...)$cex.axis)
}

plot.and.save('training-dataset.pdf', 14, 6, 1.5, plot.trainingdata, 1:8000)
plot.and.save('training-dataset-1.pdf', 14, 6, 1.5, plot.trainingdata, 1:4000)
plot.and.save('training-dataset-2.pdf', 14, 6, 1.5, plot.trainingdata, 4001:8000)

plot.and.save('acf-training-dataset.pdf', 7, 7, 1.5, acf, power.production,
              main="ACF for training dataset")
plot.and.save('pacf-training-dataset.pdf', 7, 7, 1.5, pacf, power.production,
              main="PACF for training dataset")

sink('results/data-summary-training-data.txt')
print(summary(power.production))
sink()

plot(power.production[1:1000], type="l", main="Testing my setup which is nice")
res = ets(power.production)
```

## A.4 Exponential smoothing

```
source('src/loaddata.R')
source('src/functions.R')

library('forecast')


auto.fit = ets(power.production)
```

```
sink('results/exponential-smoothing.txt')
print(auto.fit)
sink()

R.random.walk = sqrt(sum(diff(power.production.test)^2)/1998)
sink('results/prediction-error-random-walk.txt')
cat(sprintf("%0.2f", R.random.walk))
sink()
```

## A.5 ARIMA model fitting

```
source('src/loaddata.R')

model.1 = auto.arima(power.production)
model.2 = arima(power.production, order=c(6,0,0))
model.3 = arima(power.production, order=c(5,0,0))

calc.setup = list(
    list(model=model.1, predictions=rep(0,2000), errors=rep(0,2000)),
    list(model=model.2, predictions=rep(0,2000), errors=rep(0,2000)),
    list(model=model.3, predictions=rep(0,2000), errors=rep(0,2000))
)

for (k in 1:num.test.samples) {
    idx = num.training.samples + k
    for (model.num in 1:length(calc.setup)) {
        setup = calc.setup[[model.num]]
        tmp.prediction = predict(setup$model, n.ahead=1)
        calc.setup[[model.num]]$model = Arima(dat[1:idx,3], model=setup$model)
        calc.setup[[model.num]]$predictions[k] = tmp.prediction$pred[1]
        calc.setup[[model.num]]$errors[k] = tmp.prediction$se[1]
    }
}

qqplot.fns = function(x, ...) {
    qqnorm(x, ...)
    qqline(x)
}

# Forgot the labels and prediction error calculations in calc.setup
org.models = list(model.1, model.2, model.3)
model.labels = c("auto-arima", "arima600", "arima500")
model.descriptions = c("ARIMA(1,1,2)", "ARIMA(6,0,0)", "ARIMA(5,0,0)")
prediction.errors = rep(0, length(calc.setup))

for (i in 1:3) {
    lbl = model.labels[i]
    desc = model.descriptions[i]
    setup = calc.setup[[i]]
    tmp.R = sqrt(sum((setup$predictions - power.production.test)^2)/1999)
    prediction.errors[i] = tmp.R
    save.result(sprintf('prediction-error-%s.txt', lbl), cat, round(tmp.R, 2))

    save.result(sprintf('%s-model.txt', lbl), print, org.models[[i]])

    plot.and.save(sprintf('acf-residuals-%s.pdf', lbl), 8, 7, 1.5, acf, residuals(org.models[[i]]),
                  main=sprintf("ACF for residuals for %s model", desc))
    plot.and.save(sprintf('rstandard-%s.pdf', lbl), 12, 7, 1.5, plot, rstandard(org.models[[i]]),
                 xlab="Index", ylab="Standardized residual", type="l",
                 main=sprintf("Standardized residuals for the %s model", desc))


    plot.and.save(sprintf('qq-plot-%s.pdf', lbl), 7, 7, 1.5, qqplot.fns, residuals(org.models[[i]]),
                 main=sprintf("QQ plot of residuals for %s model", desc))
}


plot.and.save('acf-diffed-data.pdf', 7, 7, 1.5, acf, diff(power.production),
              main="ACF for first difference of training data")
plot.and.save('pacf-diffed-data.pdf', 7, 7, 1.5, pacf, diff(power.production),
              main="PACF for first difference of training data")
```

## A.6 HMM model function

```
inv.hessian.from.working.hessian = function (working.hessian, M) {
    inv.working.hessian = solve(working.hessian)
    return(t(M) %*% inv.working.hessian %*% M)
}

statdist = function (gamma) {
    m = dim(gamma)[1]
    one.row = rep(1, m)
    U = matrix(rep(one.row, m), nrow=m)
    Id = diag(one.row)
    inv = solve(Id - gamma + U)
    return(one.row %*% inv)
}

get.M.from.natural.params = function (natural.params) {
    mu = natural.params$mu
    sd = natural.params$sd
    gamma = as.vector(natural.params$gamma)
    m = length(mu)
    gamma.submatrix = matrix(NA, m*(m-1), m^2)
    rows = m*(m+1)
    cols = m*(m+2)
    M = matrix(0, rows, cols)
    M[1:m,1:m] = diag(mu)
    M[(m+1):(2*m),(m+1):(2*m)] = diag(sd)

    for (row.idx in 1:(m*(m-1))) {
        for (col.idx in 1:m^2) {
            fake.row.idx = row.idx + ceiling(row.idx/m)
            if (fake.row.idx == col.idx) {
                value = gamma[col.idx]*(1-gamma[col.idx])
            } else if (abs(fake.row.idx-col.idx) == m) {
                value = -gamma[fake.row.idx]*gamma[col.idx]
            } else {
                value = 0
            }
            gamma.submatrix[row.idx, col.idx] = value
        }
    }

    M[(2*m+1):(m*(m+1)),(2*m+1):(m*(m+2))] = gamma.submatrix
    return(M)
}

my.log = function (x) {
    return(log(max(x, 1e-16)))
}

get.HMM.model = function (x, m) {
    T = length(x)

    HMM.gamma.natural.to.working = function (gamma) {
        working.matrix = log(gamma/diag(gamma))
        # Return off diagonal elements
        return(working.matrix[!diag(m)])
    }

    HMM.gamma.working.to.natural = function (working.gamma) {
        natural.gamma = diag(m)
        # Fill the off diagonal elements
        natural.gamma[!natural.gamma] = exp(working.gamma)
        natural.gamma = natural.gamma/rowSums(natural.gamma)
        return(natural.gamma)
    }

    HMM.natural.params.to.working = function (mu, sd, gamma) {
        return(c(log(mu), log(sd), HMM.gamma.natural.to.working(gamma)))
    }

    HMM.working.params.to.natural = function (parvector, return.list=TRUE) {
        mu = exp(parvector[1:m])
        sd = exp(parvector[(m+1):(2*m)])
        gamma = HMM.gamma.working.to.natural(parvector[(2*m+1):(m*(m+1))])
        if (return.list) {
            return(list(mu=mu, sd=sd, gamma=gamma))
        } else {
            return(c(mu, sd, as.vector(gamma)))
        }
    }

    HMM.get.probs.matrix = function (mu, sd, dat=NULL) {
        if (!is.null(dat)) {
            x = dat
            T = length(x)
        }

        probs = matrix(0, T, m)
        for (state in 1:m) {
            probs[,state] = discrete.dnorm(x, mu[state], sd[state], 1)
        }
        return(probs)
    }

    HMM.mllk = function(working.params, debug.info=FALSE, ...) {
        natural.params = HMM.working.params.to.natural(working.params)
        mu = natural.params$mu
        sd = natural.params$sd
        gamma = natural.params$gamma

        # For debugging
        print(mu)
        print(sd)
        print(gamma)
        Ps = HMM.get.probs.matrix(mu, sd)
        Ps[is.na(Ps)] = 1

        phi = Ps[1,]
        phi.sum = sum(phi)
        llk = my.log(phi.sum)
        phi = phi/max(phi.sum, 1e-16)

        for (t in 2:T) {
            phi = phi %*% gamma * Ps[t,]
            phi.sum = sum(phi)
            llk = llk + my.log(phi.sum)
            phi = phi/max(phi.sum, 1e-16)
        }

        return(-llk)
    }

    HMM.mle = function(mu0, sd0, gamma0, ...) {
        initial.working.params = HMM.natural.params.to.working(mu0, sd0, gamma0)
        res = nlm(HMM.mllk, initial.working.params, x=x, hessian=TRUE, ...)
        mle = HMM.working.params.to.natural(res$estimate, return.list=FALSE)
        mle.list = HMM.working.params.to.natural(res$estimate)
        mllk = res$minimum
        num.params = length(mle)
        AIC = 2*(mllk+num.params)
        num.obs = sum(!is.na(x))
        BIC = 2*mllk+num.params*log(num.obs)
        # M is the coordinate shift matrix as defined
        # in equation 3.2 in Zucchini 2009
        list(mle=mle, code=res$code, mllk=mllk, AIC=AIC, BIC=BIC,
             nlm.res=res, mle.list=mle.list)
    }

    HMM.log.forward.backward = function (params, dat=NULL) {
        if (!is.null(dat)) {
            x = dat
            T = length(x)
        }

        log.alpha = log.beta = matrix(NA, m, T)

        mu = params$mu
        sd = params$sd
        gamma = params$gamma

        Ps = HMM.get.probs.matrix(mu, sd, x)
        Ps[is.na(Ps)] = 1

        # Calculate alpha
        delta = statdist(gamma)
        phi = delta*Ps[1,]
        phi.sum = sum(phi)
        lscale = my.log(phi.sum)
        phi = phi/phi.sum
        log.alpha[,1] = log(phi)+lscale
        for (t in 2:T) {
            phi = phi%*%gamma*Ps[t,]
            phi.sum = sum(phi)
            lscale = lscale+log(phi.sum)
            phi = phi/phi.sum
            log.alpha[,t] = log(phi)+lscale
        }

        # Calculate beta
        log.beta[,T] = rep(0, m)
        phi = rep(1/m, m)
        lscale = log(m)
        for (t in (T-1):1) {
            phi = gamma%*%(Ps[t+1,]*phi)
            log.beta[,t] = log(phi)+lscale
            phi.sum = sum(phi)
            phi = phi/phi.sum
            lscale = lscale+log(phi.sum)
        }

        return(list(log.alpha=log.alpha, log.beta=log.beta))
    }

    HMM.one.step.predictions = function (data.start.t, predict.x, params) {
        alpha.Ts = HMM.log.forward.backward(params, c(power.production[data.start.t:8000], predict.x))$log.alpha
        dist.resolution = 200
        xs = seq(0, 21000, length.out=dist.resolution)
        mu = params$mu
        sd = params$sd
        gamma = params$gamma
        num.preds = length(predict.x)
        pred.distributions = matrix(0, dist.resolution, num.preds)

        for (pred.num in 1:num.preds) {

            alpha.T = t(alpha.Ts[,(8000+pred.num-data.start.t)])
            L.T = sum(alpha.T)
            phi.T = alpha.T/L.T

            for (idx in 1:dist.resolution) {
                pred.distributions[idx, pred.num] = sum((phi.T%*%gamma)*discrete.dnorm(xs[idx], mu, sd, 1))
            }
        }

        return(pred.distributions)
    }

    HMM.cond.state.probs = function(params) {
        fb = HMM.log.forward.backward(params)
        la = fb$log.alpha
        lb = fb$log.beta
        c = max(la[,T])
        llk = c+log(sum(exp(la[,T]-c)))
        stateprobs = matrix(NA, ncol=T, nrow=m)
        for (i in 1:T) {
            stateprobs[,i] = exp(la[,i]+lb[,i]-llk)
        }
        return(stateprobs)
    }

    HMM.local.decoding = function(params) {
        stateprobs = HMM.cond.state.probs(params)
        ild = rep(NA, T)
        for (t in 1:T) {
            ild[t] = which.max(stateprobs[,t])
        }
        return(ild)
    }

    return(list(mle=HMM.mle, mllk=HMM.mllk,
                local.decoding=HMM.local.decoding,
                data=x,
                natural.params.to.working=HMM.natural.params.to.working,
                working.params.to.natural=HMM.working.params.to.natural,
                probs.matrix=HMM.get.probs.matrix,
                log.forward.backward=HMM.log.forward.backward,
                cond.state.probs=HMM.cond.state.probs,
                one.step.predictions=HMM.one.step.predictions))
}

discrete.dnorm = function (x, mu, sd, resolution) {
    return(pnorm(x+0.5*resolution, mu, sd) - pnorm(x-0.5*resolution, mu, sd))
}
```

## A.7 HMM model fitting

```
source('src/loaddata.R')
source('src/functions.R')
source('src/hmm-model.R')


hmm.model.2 = get.HMM.model(power.production, 2)
hmm.model.3 = get.HMM.model(power.production, 3)
hmm.model.4 = get.HMM.model(power.production, 4)

mu2 = c(4000, 9000)
sd2 = c(2000, 4000)
gamma2 = matrix(c(0.9, 0.1, 0.1, 0.9), nrow=2)

mu3 = c(16000, 10000, 4000)
sd3 = c(1800, 1800, 2000)
gamma3 = matrix(c(rep(c(0.9, rep(0.05, 3)), 2), 0.9), nrow=3)

mu4 = c(18000, 14000, 8000, 2000)
sd4 = c(1200, 1200, 1200, 1000)
gamma4 = matrix(c(rep(c(0.91, rep(0.03, 4)), 3), 0.91), nrow=4)

timing.2 = system.time(assign("mle.2", hmm.model.2$mle(mu2, sd2, gamma2)))
timing.3 = system.time(assign("mle.3", hmm.model.3$mle(mu3, sd3, gamma3)))
timing.4 = system.time(assign("mle.4", hmm.model.4$mle(mu4, sd4, gamma4)))

M.2 = get.M.from.natural.params(mle.2$mle.list)
var.cov.2 = inv.hessian.from.working.hessian(mle.2$nlm.res$hessian, M.2)
M.3 = get.M.from.natural.params(mle.3$mle.list)
var.cov.3 = inv.hessian.from.working.hessian(mle.3$nlm.res$hessian, M.3)
M.4 = get.M.from.natural.params(mle.4$mle.list)
var.cov.4 = inv.hessian.from.working.hessian(mle.4$nlm.res$hessian, M.4)

get.decoding = function (model, params) {
    decod = model$local.decoding(params)
    m = length(params$mu)
    for (i in 1:m) {
        decod[decod==i] = params$mu[i]
    }
    return(decod)
}

hmm.res.2 = list(
    time.info=timing.2,
    initial.values=list(mu=mu2, sd=sd2, gamma=gamma2),
    mle=mle.2,
    var.cov=var.cov.2,
    decoding=get.decoding(hmm.model.2, mle.2$mle.list)
)

hmm.res.3 = list(
    time.info=timing.3,
    initial.values=list(mu=mu3, sd=sd3, gamma=gamma3),
    mle=mle.3,
    var.cov=var.cov.3,
    decoding=get.decoding(hmm.model.3, mle.3$mle.list)
)

hmm.res.4 = list(
    time.info=timing.4,
    initial.values=list(mu=mu4, sd=sd4, gamma=gamma4),
    mle=mle.4,
    var.cov=var.cov.4,
    decoding=get.decoding(hmm.model.4, mle.4$mle.list)
)

save.mle.res(hmm.res.2, ’2-state-normal’)
save.mle.res(hmm.res.3, ’3-state-normal’)
save.mle.res(hmm.res.4, ’4-state-normal’)
```

## References

[1] Jonathan D. Cryer and Kung-Sik Chan. *Time Series Analysis with Applications in R*. Springer Science+Business Media, 2nd edition, 2008.

[2] Rob Hyndman, Anne Koehler, Keith Ord, and Ralph Snyder. *Forecasting with Exponential Smoothing*. Springer Berlin, 1st edition, 2008.

[3] Henrik Madsen. *Time Series Analysis*. Chapman & Hall/CRC, 1st edition, 2008.

[4] Robert H. Shumway and David S. Stoffer. `http://www.stat.pitt.edu/stoffer/tsa3/Rissues.htm`, 2011.

[5] Walter Zucchini and Iain L. MacDonald. *Hidden Markov Models for Time Series*. Chapman & Hall/CRC, 1st edition, 2009.