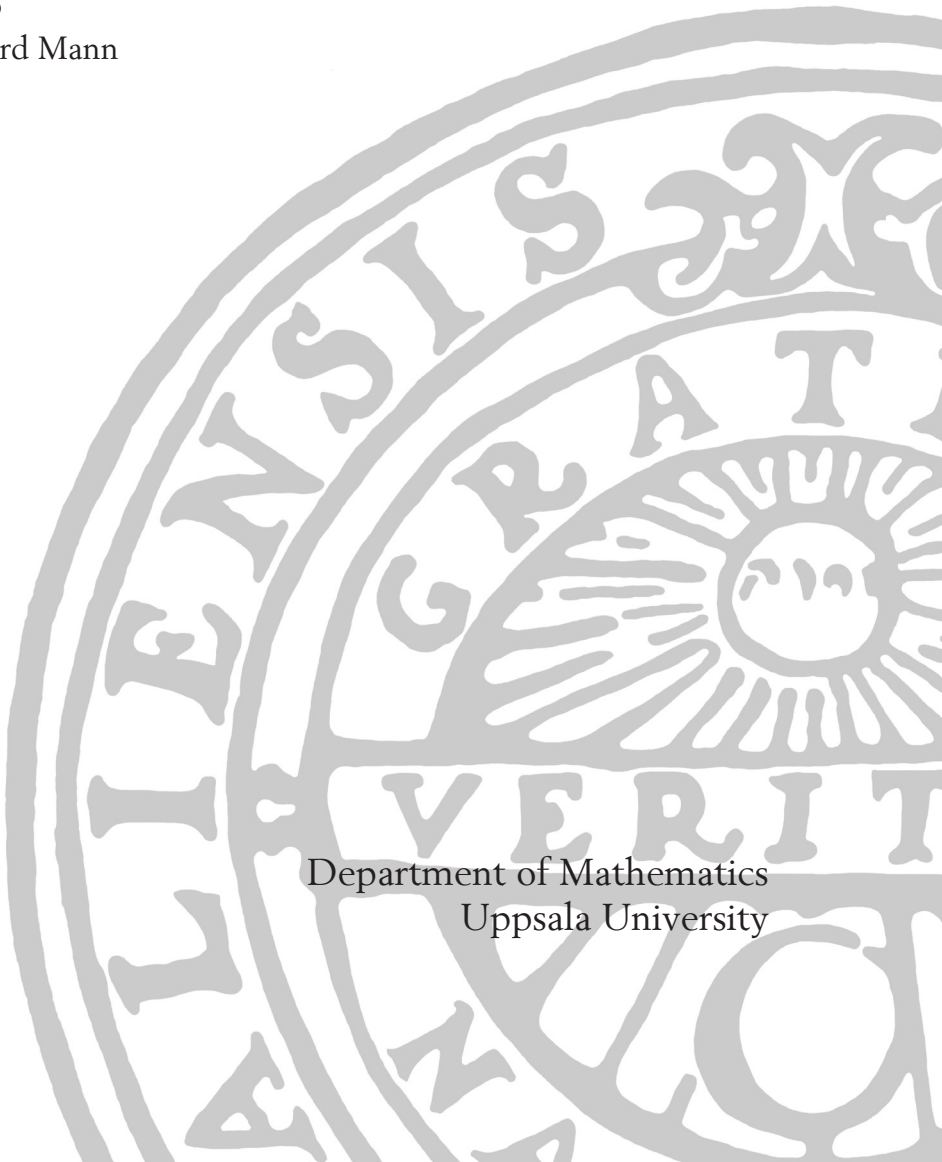
UPPSALA
UNIVERSITET

U.U.D.M. Project Report 2011:19

Movement of a prawn
a Hidden Markov Model approach

Johannes Alneberg

Examensarbete i matematik, 15 hp
Handledare och examinator: Richard Mann
Juni 2011



Department of Mathematics
Uppsala University

Abstract

This thesis presents the basic theory for hidden Markov models and standard inference algorithms for these. The performance for these algorithms are examined with the help of a small test problem and the algorithms are found to operate properly.

An application of hidden Markov models for modelling the movements of a prawn is implemented. This model is shown to be significantly better to use for the problem than a more simple model, where all movements are independent. The model also proves useful for finding the time points for where the prawn changed direction. As a brief example of how this model can be used, logistic regression is used to find that the distance to neighbouring prawns have an effect on the probability of changing direction.

Acknowledgements

Writing this thesis has been a real pleasure and I would like to thank all the people who have made it possible.

First of all, I would like to thank my supervisor Richard Mann for his excellent guidance and support.

I would also like to thank David Sumpter and the Collective Behaviour Group at the Department of Mathematics, Uppsala University, for giving me the opportunity to write about this subject.

Finally, I would like to thank all the staff at the Department of Mathematics, Uppsala University for the countless number of interesting lectures I have received and appreciated during the past three years.

Contents

1 Theory

The main objective for this chapter is to introduce a mathematical concept called hidden Markov models and further to present some algorithms often used in applications concerning this concept. A problem concerning round off errors due to computer precision, arising when implementing the hidden Markov model, and a proposed solution to this problem, will be addressed in the last section. The theory for hidden Markov models are most effectively pursued by introducing stochastic processes in general and especially Markov chains.

1.1 Stochastic Processes and Markov Chains

A stochastic process is a collection of random variables $(X(t) : t \in T)$ indexed on a set $T \subseteq \mathbb{R}$. The index t, is usually called the time of the process and the set $S \subseteq \mathbb{R}$, defined as the set of all possible values for $X(t)$, is the state space of the process. Further, we say that the process $X(t)$ is at state s at time t_0 if $X(t_0) = s$. [Stirzaker, 2005]

The main concern in this thesis will be limited to stochastic processes where both the indexing set and the state space are countable sets. Processes where the indexing set is countable, for example $T = \mathbb{Z}$ or $T = \mathbb{Z}^+$, are called time-discrete processes and the common notation for $X(n),\ n \in T$ is then X_n. Also, the notation X_n will have multitude meaning, alternating between both the process as a whole and the value of the process at time n. When the meaning otherwise would be ambiguous, it will be stated clearly what is meant.

In many cases, for example when dealing with sequential data, it proves practical to study a particular class of processes that fulfil the *Markov property*. [Bishop, 2006]

Informally, a stochastic process is said to have this property if, given full knowledge of the present state, the future is independent of the past. For the purpose of this thesis it is sufficient with a formal definition of the Markov property for the special case of time-discrete *Markov chains*.

If X_n is a stochastic process with $n \in \{0, 1, \dots\}$ and discrete state space S, then X_n is said to be a *Markov chain* if

$$\mathrm{P}(X_n = k \mid X_0 = x_0, \dots, X_{n-1} = j) = \mathrm{P}(X_n = k \mid X_{n-1} = j)$$

is true for all $n \geq 1$ and all $x_0, x_1, \dots, j, k$ in S. Further, if for all n

$$\mathrm{P}(X_n = k \mid X_{n-1} = j) = \mathbb{P}_{jk}$$

the chain is said to be homogeneous. [Stirzaker, 2005]

The homogeneity of the Markov Chain significantly simplifies calculations since it introduces a possibility to store all essential information about the process within a small square matrix $\mathbb{P}$. The elements of this matrix are simply $\mathbb{P}_{jk}$ and it is thus called the matrix of *transition probabilities*. By construction this matrix is *stochastic*, the entries are non-negative and all row sums are equal to 1.

1.2 Hidden Markov Models

When searching for a proper model to use for a specific problem, it is often a balancing act between precision in describing the problem and the effort needed to use the model.

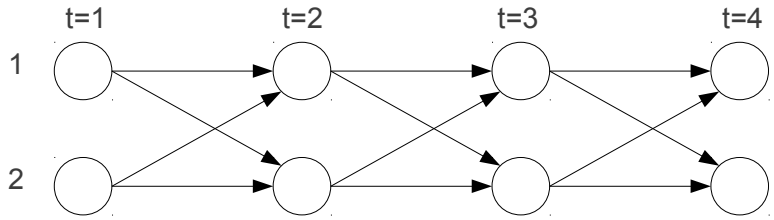


Figure 1: *An illustration of a Markov Chain with two states for four time points. Each state is presented as a circle for each time point. The arrows represent possible transitions between states.*

The main problem with Markov chains is that for some problems, they are too restrictive to efficiently describe the possible dependencies of the problem examined. A first approach to solve this problem is to expand the Markov property, letting the outcome of the process depend on more than just the present state. For obvious reasons this increase the complexity of calculations and it does so to such a degree it is often no longer practical to work with. [Bishop, 2006]

Instead, the concept of a hidden Markov model (HMM) is introduced and this proves to be a good way to balance between precision and effective calculations [Bishop, 2006]. The basic idea behind HMMs is to imagine two stochastic processes, instead of one, where one of them is a Markov chain and the other is not.

A bivariate stochastic process $\{X_n, O_n\}_{n\geq 0}$, with discrete time is a *hidden Markov model* if X_n is a Markov chain and conditional on the process X_n, the process O_n is simply a sequence of independent random variables. Further, the conditional distribution of the random variable O_n is only dependent of the random variable X_n. [Cappé et al., 2005].

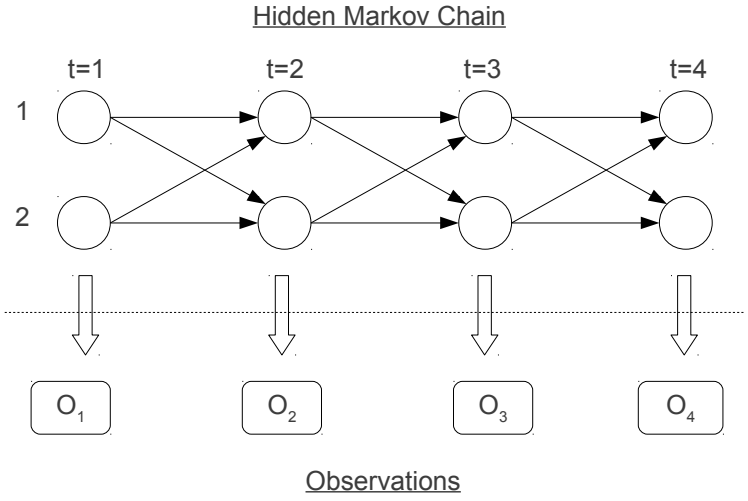


Figure 2: *A Hidden Markov Model with two states and where the distribution of O_i is determined by the state of the chain at $t = i$.*

The process X_n is the so called *hidden Markov chain* and the process O_n is usually called the *observational sequence*. The HMM can be seen as a Markov chain whose values are unobservable but where each value generates a specific observation with some probability. If further the HMM is assumed to be time homogeneous in every sense, the whole model, λ, can be described by two matrices together with an additional vector, $\lambda = \{P, Q, \pi\}$. The matrix P is the transitional matrix for the Markov chain X_n and the matrix Q, called the observational matrix, contains all conditional probabilities for observing each outcome, that is

$$\{Q\}_{ij} = \mathrm{P}(O_k = j \mid X_k = i)$$

To start a hidden Markov chain, the initial distribution π is used, a discrete probability vector giving the probability for each state to be the starting point. The following notation will be used throughout the text

$$\mathbb{O}_t^T = \{O_t = o_t, O_{t+1} = o_{t+1}, \ldots, O_T = o_T\}$$

That is, $\mathbb{O}_t^T$ is a short notation for the event of a certain observation sequence from time t to T. In a similar way $\mathbb{X}_t^T$ will be a short notation for the event of a certain sequence of states visited by X_n from time t to time T. Also x_t will be the specific state that X_n visited at time t. The different possible states for X_n is denoted $S = \{S_1, \ldots, S_N\}$.

In practise, the Markov property for the hidden Markov chain, together with the specific dependence structure for the observational sequence is what gives the HMM computational advantages. This will be clear when studying some basic inference algorithms for HMMs. [Bishop, 2006]

1.3 Inference Algorithms for HMMs

Algorithms used for inference in HMMs does mainly approach three different problems. The first of these problems is called the evaluation problem and it is the task of calculating $\mathrm{P}\big(\mathbb{O}_t^T \mid \lambda\big)$, the probability of a certain observation sequence given a specified model λ. The second problem is the question of finding out what state sequence that was used to generate the observational sequence. The third and final problem is how to change the model $\lambda = \{P, Q, \pi\}$ in order to maximise $\mathrm{P}\big(\mathbb{O}_t^T \mid \lambda\big)$ and thus in some way finding the most suitable model for the observed sequence. [Rabiner, 1989]

The first two problems will here be solved with standard inference algorithms for HMMs while the third problem will be solved with Bayesian analysis.

The first problem could be solved with a simple naive algorithm. This would include summing over all possible state sequences after calculating the probability for the observation sequence directly for each state sequence. This would however have major drawbacks in practise since the number of possible state sequences grows exponentially with respect to the number of data points. Therefore, a naive algorithm, calculating the probability for one sequence at a time, would have exponential time complexity, $O(TN^T)$. It is thus useless but for trivial cases. [Rabiner, 1989]

Instead, the first problem will be dealt with using dynamic programming in the Forward-Backward algorithm. This algorithm consist of two different steps, the forward step and the

backward step. To solve the first problem though, it is only necessary to perform the forward step, but since the backward step is later used in the solution for the second problem, it will also be presented here. The forward step is a procedure to calculate the forward probability $\alpha_t(i)$, defined as follows.

$$\alpha_t(i) = \mathrm{P}\big(\mathbb{O}_1^t, X_t = S_i \mid \lambda\big)$$

That is, the probability of the observed sequence up to time t and the chain being in state i at time t, given the model parameters. This probability can easily be calculated inductively with the following expressions given in [Rabiner, 1989].

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i)\mathbb{P}_{ij}\right] \cdot \mathbb{Q}_{jo_{t+1}}$$

With the following initial condition.

$$\alpha_1(i) = \pi_i \cdot \mathbb{Q}_{io_1}$$

Finally the probability required is established in the termination step.

$$\mathrm{P}\big(\mathbb{O}_1^T \mid \lambda\big) = \sum_{i=1}^{N} \alpha_T(i)$$

In order to calculate probabilities for single time points instead of the whole sequence, the backward step is also needed. It is a procedure to calculate the backward probability, $\beta_t(i)$ defined as follows.

$$\beta_t(i) = \mathrm{P}\big(\mathbb{O}_{t+1}^T \mid X_t = S_i, \lambda\big)$$

This definition is obviously similar to the forward probability, here with the time running backwards, but it is also conditioned on the state at time t. It is therefore not difficult to think of the forward and the backward probabilities as complementing pairs, with time running forwards and backwards respectively towards the same time point t. The computation of the backward probabilities is also similar to the forward step with the inductive procedure

$$\beta_t(i) = \sum_{j=1}^{N} \mathbb{P}_{ij}\mathbb{Q}_{jO_{t+1}}\beta_{t+1}(j)$$

Initialised with $\beta_t(i) = 1$ for all $1 \leq i \leq N$. To establish a way to calculate the probability for single states at different time points Bayes' theorem will be used. Assuming $\mathrm{P}(B) \neq 0$, Bayes' theorem is stated

$$\mathrm{P}(A \mid B) = \frac{\mathrm{P}(B \mid A)\,\mathrm{P}(A)}{\mathrm{P}(B)} \tag{1}$$

The following use of this and the dependency structure for hidden Markov models give

$$\begin{aligned}\mathrm{P}\big(X_t = S_i \mid \mathbb{O}_1^T, \lambda\big) =& \frac{\mathrm{P}\big(\mathbb{O}_1^T \mid X_t = S_i\big)\,\mathrm{P}(X_t = S_i)}{\mathrm{P}(\mathbb{O}_1^T)} = \\ =& \frac{\mathrm{P}(\mathbb{O}_1^t \mid X_t = S_i)\,\mathrm{P}\big(\mathbb{O}_{t+1}^T \mid X_t = S_i\big)\,\mathrm{P}(X_t = S_i)}{\mathrm{P}(\mathbb{O}_1^T)} = \\ =& \frac{\mathrm{P}(\mathbb{O}_1^t, X_t = S_i)\,\mathrm{P}\big(\mathbb{O}_{t+1}^T \mid X_t = S_i\big)}{\mathrm{P}(\mathbb{O}_1^T)}\end{aligned}$$

These probabilities are all familiar and it is now possible to compute the probability for single states at different time points given the observation sequence.

$$\mathrm{P}\big(X_t = S_i \mid \mathbb{O}_1^T, \lambda\big) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^N \alpha_t(j)\beta_t(j)} \tag{2}$$

The problem of finding the most probable state sequence can be approached in many ways since different criteria for the optimisation give different methods. For example, a condition on a optimal sequence can be that its states are individually optimal. This approach is solved using the formula above, computing at each time point the probability for the different states. So the optimal state at each time point t, is the argument that maximise this expression. However this is not the criteria that is most commonly used, since it can give a non-valid state sequence as the optimal one. Instead the optimal sequence will here be thought of as the one that maximise the joint probability $\mathrm{P}\big(\mathbb{X}_1^T \mid \mathbb{O}_1^T, \lambda\big)$.

The Viterbi algorithm [Viterbi, 1967] is an inductive procedure to solve the second problem with this more complex criterion for optimisation. First of all the Viterbi algorithm calculates the highest probability for a single path up to time t ending in state S_i, denoted as $\delta_t(i)$. This can be found inductively since the most probable path ending in state S_j at time $t+1$ can be found by only investigating the transition probabilities from all the possible previous states combined with how likely those previous states are. In order to find the actual sequence, the algorithm also has to keep track of the argument state S_i maximising this probability for each time point, denoted as $\psi_t(i)$. More formally, for each time point $2 \leq t \leq T$ and each state S_j with $1 \leq j \leq N$

$$\begin{aligned}\delta_{t+1}(j) &= \Big(\max_i \left[\delta_t(i) \cdot \mathbb{P}_{ij}\right]\Big) \cdot \mathbb{Q}_{jO_{t+1}} \\ \psi_{t+1}(j) &= \arg\max_{1 \leq i \leq N} \left[\delta_t(i) \cdot \mathbb{P}_{ij}\right]\end{aligned}$$

The inductive procedure is initialised with

$$\begin{aligned}\delta_1(j) &= \pi_i \mathbb{Q}_{jO_1} \\ \psi_1(j) &= 0\end{aligned}$$

Finally termination and backtracking is performed, giving the most probable state sequence $\{X_t^*\}$.

$$\begin{aligned}X_T^* &= \arg\max_{1 \leq j \leq N} \delta_T(i) \\ X_t^* &= \psi_{t+1}(X_{t+1}^*)\end{aligned}$$

The third problem is the most complex of the three and can also be solved in many different ways. An algorithm that fits the model parameters according to available data is often referred to as a *learning* algorithm, from the discipline *machine learning*, automatic statistical model fitting. As opposed to deduction, where conclusions are made out of certain facts, learning algorithms deal with uncertain information. Under some simple axioms that are easily accepted, uncertainty can be quantified consistently with the rules of probability. Under the same axioms, it can also be proved that Bayesian analysis is the only way to consistently perform model fitting. Since quantification of uncertainty and probability is equivalent, there is no need for separate notations. [Baldi and Brunak, 2001]

Fundamental to Bayesian analysis is of course Bayes' theorem, stated in equation 1. The quantity to the left in this equation is called the *posterior distribution* and expressed with model parameters θ and observation data $\mathbb{O}_1^T$, equation 1 takes the form

$$\mathrm{P}\big(\theta \mid \mathbb{O}_1^T\big) = \frac{\mathrm{P}\big(\mathbb{O}_1^T \mid \theta\big)\, p\,(\theta)}{\mathrm{P}(\mathbb{O}_1^T)}$$

where $p\,(\theta)$ is the *prior distribution* representing any knowledge of the parameters prior to the data. The other factor in the numerator, $\mathrm{P}(\mathbb{O} \mid \theta)$, is the likelihood function for the parameters. Thus the parameters maximising the posterior distribution when the prior distribution is uniform, is equivalent to the maximum likelihood estimates. The quantity in the denominator can be seen as a normalising factor, so that the sum over the parameter space is equal to one.

Although the most commonly used learning algorithm for HMMs, the Baum Welch algorithm [Durbin, 1998], would suit the purpose of this thesis, a more brute force algorithm based on Bayesian analysis is sufficient. To calculate the posterior distribution for the parameters in a HMM, only the forward algorithm need to be used to find the likelihood.

$$\mathrm{P}\big(\mathbb{O}_1^T \mid \theta\big) = \sum_{i=1}^{N} \alpha_T(i)$$

If the parameter space is large or uncountable, some discretisation is needed, giving a fixed number of possible values for each parameter. This gives the possibility to calculate the likelihood for every possible combination of the parameters to find the posterior distribution completely for the chosen discretisation. Thus, in order to estimate any parameter, either the expected value or the maximum likelihood estimation can be found. Further, with the posterior distribution, variance, standard deviation and other quantities can easily be calculated with the standard procedure for discrete distributions.

1.4 The Logarithmic Domain

When using a computer for calculations, the way numbers are represented are of great importance. For example, the IEEE double-precision format used by MATLAB® has a smallest positive number that can be represented. To be exact, it is $1 \cdot 2^{-1022}$[Chapra, 2006]. All smaller numbers will be treated as identically equal to zero in any calculation. This is of course something that should be avoided if possible. Experience shows that the forward probabilities for example gets smaller and smaller for each iteration. They do so in such a way that for large

data sets, the smallest positive number possible to represent will be reached and surpassed. This can generate erroneous results. In order to avoid this problem, when large data sets are used, all probabilities will be calculated in the logarithmic domain. That is, all probabilities are scaled into the negative real axis with the natural logarithm. The scaling is one to one, since the logarithm is monotonically increasing. Further, the logarithm laws make it possible to express the recursive formulas directly in log-probabilities.

The log-forward probabilities $l\alpha_t(i)$ can be calculated according to the following formula. In order to increase the readability, let m denote the maximum value of the log-forward probabilities for the most recent time point. That is

$$m_{t-1} = \max_{1 \leq i \leq N} l\alpha_{t-1}(i)$$

giving

$$l\alpha_t(i) = m_{t-1} + \log(\mathbb{Q}_{jO_t}) + \log\left(\sum_{j=1}^{N} \exp(l\alpha_{t-1}(j) - m_{t-1})\right)$$

In order to avoid the exponential function evaluating to zero, the exponents are normalised with the maximum value among them.

The log-backward probabilities $l\beta_t(i)$ can, in a very similar way, be calculated in a transformed recursive formula.

$$m_{t+1} = \max_{1 \leq i \leq N} l\beta_{t+1}(i)$$

giving

$$l\beta_t(i) = m_{t+1} + \log\left(\sum_{j=1}^{N} \exp(l\beta_{t+1}(i) - m_{t+1}) \cdot \mathbb{P}_{ij} \cdot \mathbb{Q}_{jO_{t+1}}\right)$$

Also the final step in the Forward-Backward algorithm can be transformed into the logarithmic domain. First let the respective maximum values be denoted.

$$ma_t = \max_{1 \leq i \leq N} l\alpha_t(i)$$
$$mb_t = \max_{1 \leq i \leq N} l\beta_t(i)$$

The formula then become

$$\log\left(\mathrm{P}\left(x_t = S_i \mid \mathbb{O}_1^T, \lambda\right)\right) = l\alpha_t(i) + l\beta_t(i) - ma_t - mb_t - \ldots$$
$$- \log\left(\sum_{j=1}^{N} \exp\left[l\alpha_t(j) - ma_t\right] \cdot \exp\left[l\beta_t(j) - mb_t\right]\right)$$

The Viterbi algorithm is also easily transformed, giving

$$l\delta_{t+1}(j) = \max_i \left[l\delta_{t-1}(i) + \log\left(\mathbb{P}_{ij}\right)\right] + \log\left(\mathbb{Q}_{jO_{t+1}}\right)$$
$$\psi_{t+1}(j) = \arg\max_{1 \leq i \leq N} \left[l\delta_t(i) + \log\left(\mathbb{P}_{ij}\right)\right]$$

With unchanged backtracking

$$X_T^* = \arg\max_{1 \leq j \leq N} \delta_T(i)$$
$$X_t^* = \psi_{t+1}(X_{t+1}^*)$$

2 Testing the Algorithms

In this section, the performance of the algorithms described above is examined. The algorithms was implemented in MATLAB® code, but since the ideas of the algorithms are presented in the previous sections, no details of the exact implementation is presented here. The code for the versions of the algorithms used in this section is however available in the appendix. To test the performance of the algorithms, Monte Carlo tests were used on a small problem concerning a biased and an unbiased coin. This constructed problem arise when the results of a series of coin tosses are available but there is some suspicion that at least some tosses are made with a biased coin. The assumption is furthermore that the changes between the different coins constitutes a Markov chain. This clearly constitutes a hidden Markov model with two by two transition and observation matrices. For the purpose of this text it is possible to restrict the model further, giving:

$$\mathbb{P} = \begin{pmatrix} p & 1-p \\ 1-p & p \end{pmatrix}$$

That is, there is no preference for any coin over the other. If one coin is assumed to be fair, the observation matrix is also dependent on one parameter, determining the bias of the other coin.

$$\mathbb{Q} = \begin{pmatrix} 0.5 & 0.5 \\ q & 1-q \end{pmatrix}$$

Without loss of generality, assume that the observation corresponding to the first column in $\mathbb{Q}$ is tails. With this assumption, q is the probability for observing tails in a single toss for the possibly biased coin. Of course if $q = 0.5$, both of the coins are fair. In this case, the hidden Markov model is just a complicated version of attaining a sequence of independent and identically distributed coin tosses.

The benefits of using a small constructed problem to test the algorithms are numerous. With the correct answer available, it is possible to evaluate the performance of the algorithms by inspection. The simplicity of the problem is also appealing since it makes the analysis of the algorithms easier and the conclusions clearer. A simple script was used to generate two paired sequences, the first giving the result of the coin tosses and the other giving which coin was used for each toss. With the observation sequence given as input to the algorithms, the performance can be evaluated by comparing the result with the correct answer.

2.1 Forward-Backward Algorithm

The first algorithm that was covered was the Forward-Backward algorithm. To perform a test, equation 2 can be used to find the single most probable state at each position in a sequence. A reasonable requirement is that the algorithm should perform better than an average random guess. In Figure 3 the percent of correct guessed states is shown for different values for the parameter q in the observation matrix and p in the transition matrix. The range for q is only between 0.1 and 0.5, motivated by the symmetry in the problem; it does not matter for the algorithm whether the biased coin is biased towards heads or tails. Before analysing the performance, it can be useful to point out once again that p represents the probability of not changing states and $1 - p$ is the probability of changing states.

Also, Figure 3 shows two different plots, giving two different views of the same data. In the left diagram the data is grouped according to the corresponding value for p, while different colours are used to distinguish different values for q. The other diagram is done in the opposite way, grouping different q values along the x-axis and different p values with colours.

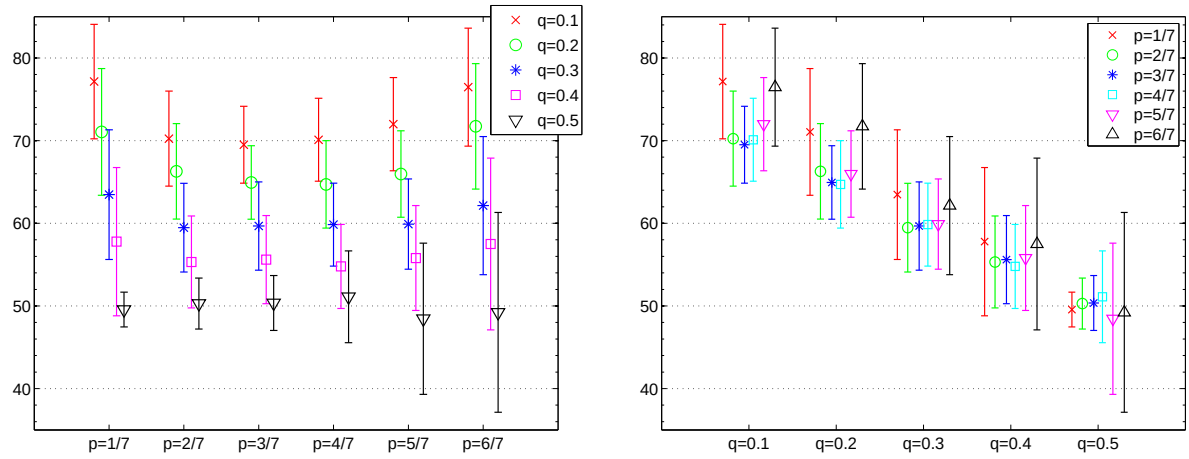


Figure 3: *The performance of the Forward-Backward algorithm for different observation and transition matrices. The parameter p is the probability of not changing states and the parameter q is the probability for heads for the possibly biased coin. The scale to the left is the percentage of correct guessed states.*

The first and most obvious conclusion that can be drawn from Figure 3 is that the accuracy of the algorithm is higher with a severely biased coin compared to a less biased coin. This is seen in the right diagram as the number of correct guessed states decrease as the value for q gets higher. This can be understood from the problem itself, since the less different the two coins are, the harder it is to draw conclusions about which coin that was used. Ultimately with $q = 1/2$, the two coins are identical and no information about which coin was used can be found in the observation sequence. On the other hand, if one of the two coins for example is greatly favouring heads, then when heads is observed, it is reasonable to suspect that the biased coin was used. Thus, in this case the algorithm has more information about what state actually generated the observation.

When the transitional matrix is varied instead, a small value for p, the probability for not changing states, apparently gives the best precision while the best accuracy is attained for both low and high values for p. This can somewhat be explained by the dependency structure. For an observation, the dependency of surrounding observations is larger for higher and lower values for p since $p = 0.5$ correspond to choosing coins independently. Thus, more information can be found from the neighbouring observations in those cases. If no information about which coin that was used is given by the observations, i.e. $q = 0.5$, this additional information is no good and cannot increase the accuracy of the model. For the other cases however, the presence of dependency can amplify the information given by the observations.

Further, the difference between a low and a high value for p is the expected number of transitions. A larger number of transitions, i.e. a low value for p, seems to increase the sample variation. This can be explained since random effects can have a larger impact on the result if there are less transitions. Overall, there is no reason to not believe that the algorithm is working properly.

2.2 Viterbi Algorithm

The same type of tests as for the Forward-Backward algorithm were performed to test the Viterbi algorithm. The results are shown in Figure 4. For the same reason as in the Forward-Backward algorithm, it is sufficient to use values for q between 0.1 and 0.5.

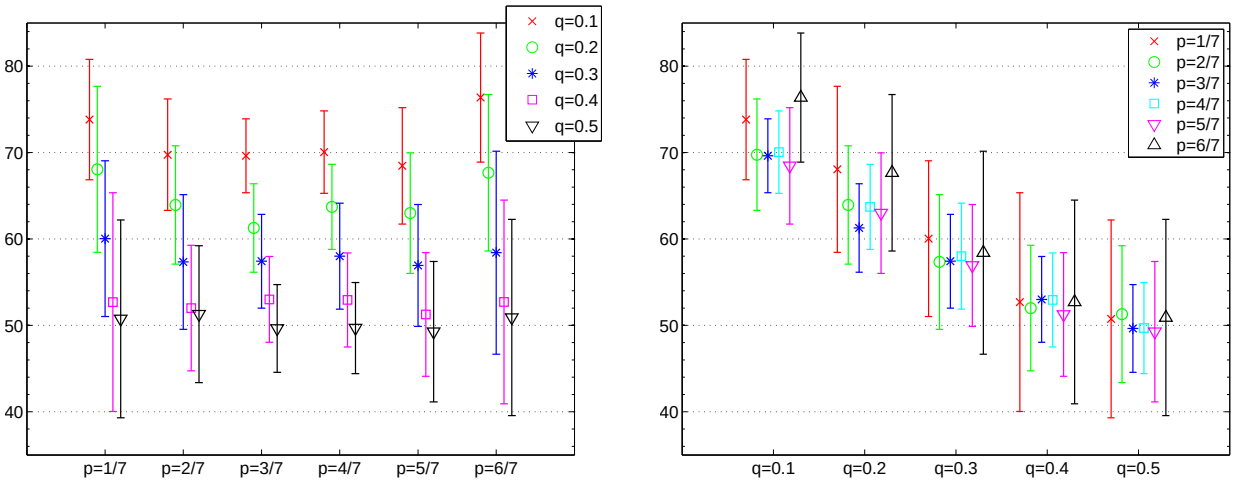


Figure 4: *The performance of the Viterbi algorithm for different observation and transition matrices. The parameter p is the probability of not changing states and the parameter q is the probability for heads for the possibly biased coin. The scale to the left is the percentage of correct guessed states.*

From studying Figure 4, it is evident that the Viterbi algorithm shows essentially the same behaviour as the Forward-Backward algorithm. The only differences seen is that the number of correct guessed states are a few less than with the Forward-Backward algorithm and that the increase of accuracy with lower value for p is not present. The reduction in the number of correct guessed states is natural since the Viterbi algorithm finds the most probable state sequence, which is often not equivalent with the sequence made out of the most probable states for each time point. The latter sequence is given by the Forward-Backward algorithm, maximising the expected number of correct guessed states. The Viterbi algorithm thus seems to work properly.

2.3 Learning the Model Parameters

When discussing the performance of the learning algorithm, only aspects concerning the ability to estimate the parameters will be included. Other aspects of an algorithm that can be associated with performance, such as execution time, will thus not be discussed here. Intuitively, the performance of the learning algorithm should be dependent of the amount of data given, since a small data set contain less information about the model parameters than a large set of data. Ultimately, some kind of convergence towards the correct values is necessary for the algorithm to be useful at all.

A Monte Carlo test was performed with the objective of studying how the performance was dependent of the amount of data. For each data set, consisting of 5000 data points, the algorithm was initially given the first 100 points then the first 800 points and so on up to finally using all 5000 data points. A total of 60 data sets was used, each evaluated in 8 overlapping pieces. Studying the results for these, the effect of additional data can be examined.

Figure 5 shows the result of the Monte Carlo test in terms of the expected value for the parameter p. As earlier mentioned, p is the probability for the hidden Markov chain to change state. The correct value, $p = 0.7$ that was used to generate the data points is shown as a dotted line. Since the distance between the lower quartile and the upper quartile decrease with longer data sets, a

Figure 5: Expected value of p

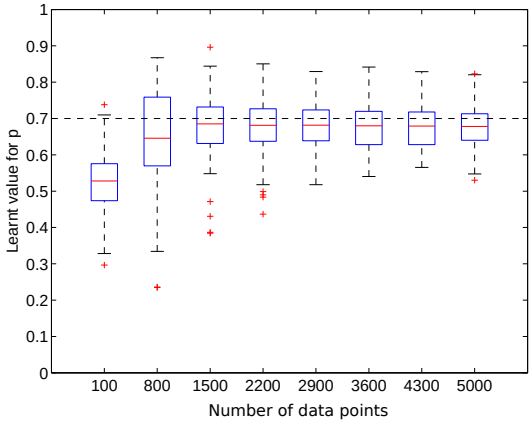


Figure 6: ML-estimation of p

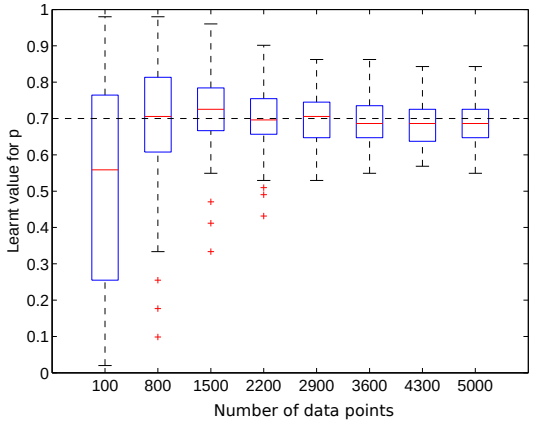


Figure 7: Expected value of q

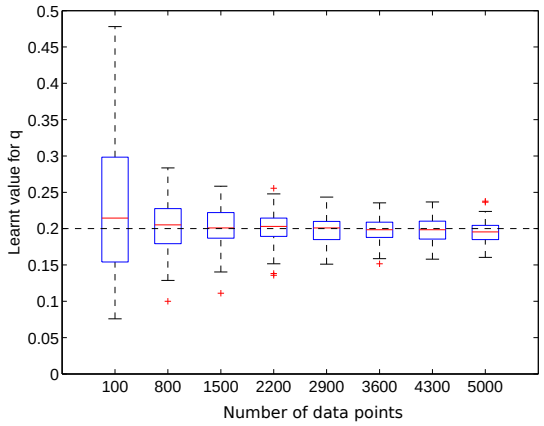
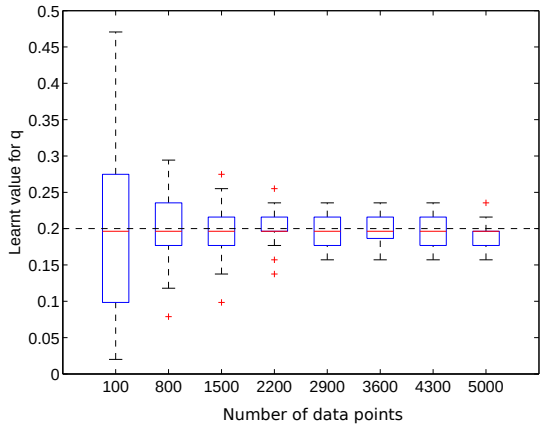


Figure 8: ML-estimation of q



Boxplots displaying the results from the learning algorithm for different sizes of data sets. The correct values are displayed as dotted lines.

tendency for convergence to some set of values is indicated. The direction of this convergence is also very important, and since the correct value is included in the boxes for all samples, except the very first, the algorithm is likely to find good a estimate of the parameter for a large data set.

Figure 6 shows the corresponding results of the maximum likelihood estimation of p. The results for this estimation method also indicates the sought after convergence in a similar way as for the expected value. Comparing the results of the two estimators however, indicates a slightly larger deviation for the maximum likelihood estimator. This suggests that the expected value is a more efficient estimator than the maximum likelihood estimator. [Alm and Britton, 2008]

Figure 7 and 8 shows the corresponding results associated with q. Recall that in the test problem, q is the parameter governing the emission probabilities for the biased coin. More specifically, q was defined to be the probability of tails for the possibly biased coin. In the same way as for p, the convergence towards the true value showed as a dotted line is evident for both the expected value and the maximum likelihood estimator. However, for this parameter there is no estimator that is obviously more efficient than the other. Both the expected value and the maximum likelihood estimator seems to be good estimators for q.

It is also possible to compare results between these two parameters but then one must first note that the figures are scaled in different ways. The figures associated with p display all possible values for a probability, that is, between zero and one, while Figure 7 and 8 only display values between zero and one half. When scaling is taken into account, the deviation for estimating q is significantly smaller than for p. This indicates that it is easier to make a good estimation of q than p. This is a way of saying that it takes less data to estimate q within a certain range of certainty than for p. To explain this, one approach is to examine the properties of the model.

Because of the symmetry in this specific problem, where each coin is equally likely and one of the coins is known to be fair, there exists a simple way to estimate q. Since half of all tosses is expected to come from the fair coin, a quarter of all tosses can be expected to be tails received from the fair coin. The rest of the observations showing tails in the observation is thus expected to come from the biased coin. Also, approximately half of all tosses can be assumed to come from the biased coin. Thus q can be estimated directly as the approximate relative frequency of tails for the biased coin.

$$q* = \frac{\#T - N/4}{N/2}$$

where $\#T$ is the number of tails in the observation sequence and N is the number of observations in total. Even if this is not the way of estimating q actual used, it shows that q can be estimated without any knowledge of p, while estimating p is a more complex task.

Another way to reason about that q is more efficiently estimated than p, is to think in terms of posterior probability. Often, the set of probable values for p is larger than the corresponding set for q. If for example one observation sequence contains 35 tails and 65 heads, this result would lead to the conclusion that $q < 0.5$, since the result is very unlikely for other values for q. However, it is not as easy to determine how many transitions that has occurred and thus be able to estimate p directly.

Evidence for transitions is only present as changes in relative frequency between different parts of the sequence. Thus for a small number of data points, it is hard for the algorithm to tell any difference between actual transitions between the coins and effects caused by randomness within the use of a single coin.

Figure 9-12 shows an example of the posterior probability for different lengths of the same observation sequence. Higher values is shown as red or yellow while the lowest values are displayed as dark blue. The conclusions that could be drawn from the Monte Carlo test is also present here. First of all, the convergence towards the true value is evident as the area where higher values are present decreases and in Figure 12, only a small area around $p = 0.7$ and $q = 0.2$ is left. Also, the highlighted area is a bit more outstretched vertically than horizontally, especially in Figure 9 and 11. This is not strange since the conclusion above is that p is more difficult to estimate compared to q. The differences between the expected value and the maximum likelihood estimator can also be seen, especially in Figure 9 where the highest posterior probability and thus the maximum likelihood estimation is severely off target. When the amount of data is increased however, both estimators converge to the correct values.

Figure 9: 100 data points

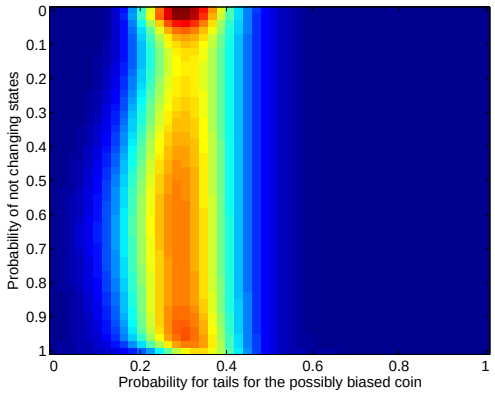


Figure 10: 800 data points

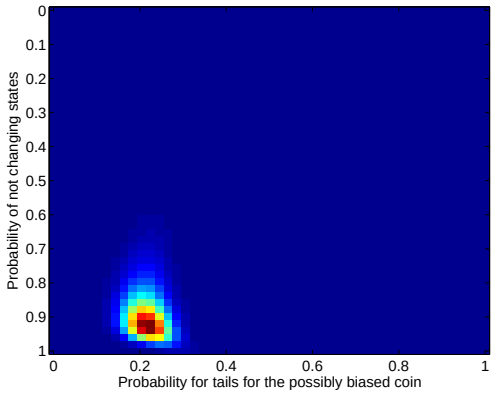


Figure 11: 1500 data points

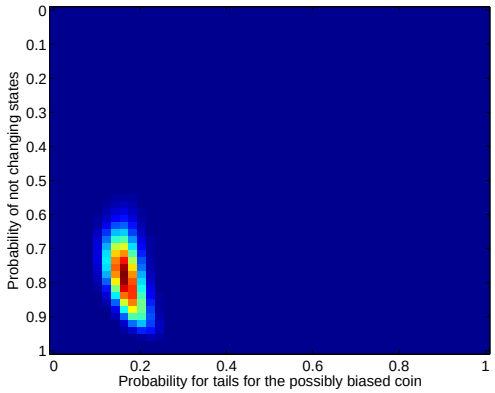
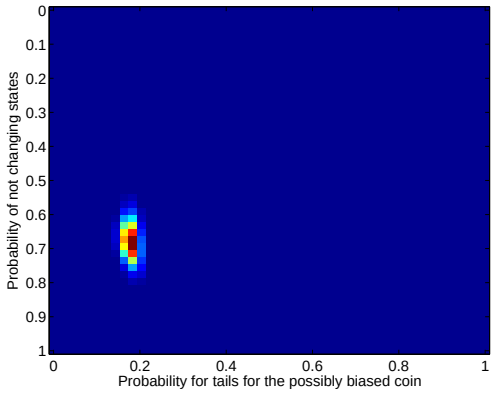


Figure 12: 5000 data points



Examples of the posterior distribution for the parameters after evaluating the specified number of data points. The scale to the left in each figure corresponds to p, the probability of not changing states. The scale on the bottom of each figure corresponds to q, the probability of attaining tails for the possibly biased coin. The correct value was $p = 0.7$, $q = 0.2$

3 HMMs applied to Prawn Movement

In this section, real experimental data will be studied and the theory of hidden Markov models will be applied to these. The experimental data that was used for this thesis origins from a system of co-moving freshwater prawns that was studied by researchers from the University of Sydney interested in collective animal behaviour. The experimental procedure was to store each prawn's position 15 times per second for as long as 6 minutes and the process was repeated for different prawns and different set ups. In this thesis, 58 experiments with a set up of three prawns was used, giving a total of 174 data sets with about 5500 data points for each prawn.

In the experiment, the prawns were located inside an aquarium with a cylindrical shape where also a cylinder in the middle was taken away. With this shape it is possible to treat the prawns movements as if they were in only one dimension, not taking into account the distance from the centre. Thus the only quantity actually used as data was the change in angular position, measured in radians.

As a part of a larger research context, the main objective for the application of HMMs in this case was to automatically find out when a prawn changed direction. At first sight, this could seem to be a trivial task, since positive movements would indicate that the prawn was heading forward and vice versa. The problem arose when discovering that even though the prawn was really heading forward, some values for the movement could be zero and some points could even take on negative values. The hope is that an automatic procedure to find out when a prawn changed direction would be a step in the process of finding out something about the decision processes governing the movement of a prawn.

3.1 Pre-processing the Data

The observed data for the prawns consisted of measured movements between two subsequent frames created by constructing the difference of observed values for the angular position. A problem encountered during the process was that some samples seemed to be contaminated with some erroneous measurements. Figure 13 shows an example of a data set where a vast majority of the observations are between -0.2 and 0.2 but some single values are much larger in absolute value. Since these values are isolated it is reasonable to believe that they are erroneous.

After concluding that these values were not valid, there were different ways that could have been used to solve this problem. One way could have been to delete these values completely, making the data sequence shorter. Another way could have been to set all these values to 0, keeping the same lengths for the data sequences. In order to respect the model that was used and how the way of solving the problem would affect the result, the method that was used was to set all values greater than $\pi/10 \approx 0.31$ in absolute value to zero. To chose a factor of π as a limit is motivated from the use of radians as the unit.

It was also reasonable to believe that other points was erroneous, for example, singular points separated with some significant distance from its neighbours. Since the natural laws prohibit the prawns from too rapid movements and the time points were sufficiently close to each other, there should have existed some limitation on how big the differences could have been between different movements. Also, singular negative points in a section where all other values are positive can be very influential when deciding the direction of the prawn. This problem was solved by setting any point distanced more than $\pi/60 \approx 0.05$ from both its two preceding and its two succeeding neighbours to zero. Also here, the factor of π is motivated by the use of radians as the unit.

3.2 Two Proposed Models

The model for the circular movement of a prawn that was primarily used in this thesis, is a HMM with two states, each associated with 'heading clockwise' and 'heading counter clockwise' respectively. In this way, a transition between any of the two states corresponds to a change of direction for the prawn. The distributions used to govern the observations from these states are very similar to each other, but differ in the sense that the mean values has opposite signs. To be precise, the distribution for 'heading counter clockwise' is the same as the distribution for 'heading clockwise', only reflected in the line $x = 0$. This choice can be justified by the assumption that there is no essential difference between the two directions, the prawn moves in the same manner, whether it is heading clockwise or counter clockwise.

It is not obvious which distribution that should have been used to model the amount of movement per time unit. In nature, this amount should be able to take any value in some interval including zero, which suggest that its distribution should have been continuous. However, the

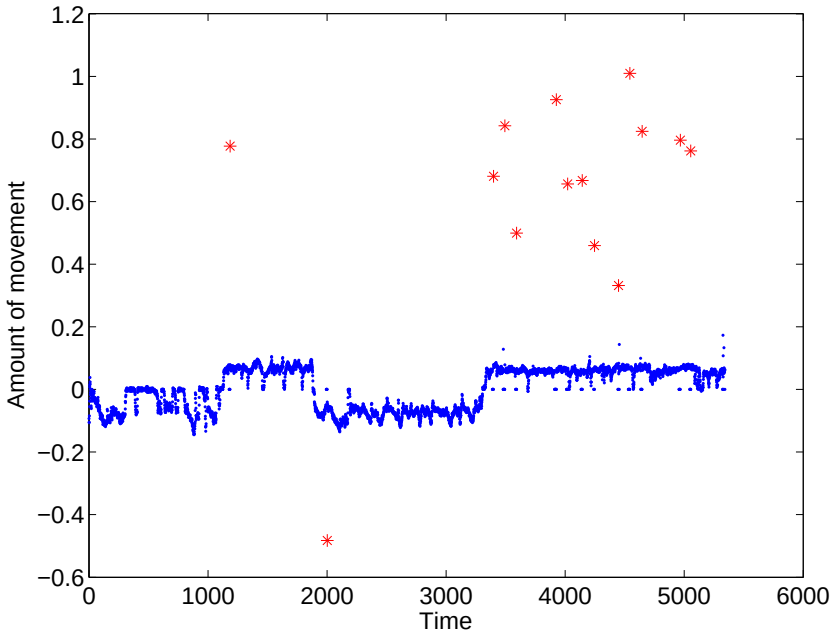


Figure 13: *An example of a data set with noise present. Separate markers are used for points outside the interval* $(-\frac{\pi}{10}, \frac{\pi}{10})$ *to clarify that these are considered to be noise .*

measuring devices used in the experiment have limitations in precision and cannot distinguish between more than a countable set of values. Though, with sufficiently good precision, this would not have been an absolute reason to set aside the continuous assumption. More importantly, the data showed a significant amount of values exactly equal to zero. This is not consistent with the assumption of a continuous distribution, since any single value has probability zero for a continuous distribution. To overcome this, a mixture of a one-point distribution and a normal distribution was used to model the movements for each state. With probability p_0, the first distribution was chosen, and the movement was set equal to zero, and with probability $1 - p_0$ the normal distribution was chosen, and the movement was generated according to this.

A mixture of continuous and discrete distributions would however have been rather difficult to use in the inference algorithms since this would imply combining true probabilities and densities in the likelihood calculations. The normal distribution was thus discredited to simplify computations and make the distribution for each state entirely discrete.

Of course, other hidden Markov models could have been used to model the movements of a prawn, but the model presented here is arguably the most simple but non-trivial HMM that is suitable for the problem. However, for the sake of comparison, an even more simple HMM was also used. This second model is basically the same as the first presented but with the difference that all entries in the transition matrix were set equal to 0.5. So two different distributions, associated with 'heading clockwise' and 'heading counter clockwise' were still used to model the movements of the prawn but in this model there was no correlation between these over time. This model can also be described as, for each time point, first randomly choosing a direction and then find the amount of movement randomly from the discretised Gaussian distribution.

3.3 The Learnt Parameters

For computational reasons, only seven data sets with movements was used as data for the learning algorithm for the HMM. These sets were randomly picked out among all data sets. The HMM presented above is dependent of four parameters, three of which was learnt with the learning algorithm presented and one that was estimated directly. The probability of attaining zero for both of the observational distributions, p_0, was estimated directly as the relative frequency of zeros in all observations. Further, two other parameters concerning the distributions directly, the mean value, μ, and the standard deviation, σ, for the discretised normal distribution was learnt with the learning algorithm. The last parameter to be learnt was the parameter governing the transitions of the hidden Markov chain, p, not to be confused with the parameter p_0. Quite the opposite of earlier use of this parameter name, this parameter was defined as the probability of changing direction at a specific time point, this gives the transitional matrix

$$\mathbb{P} = \begin{pmatrix} 1-p & p \\ p & 1-p \end{pmatrix}$$

The result of the basic inference and the learning algorithm was

$$\mu = 0.113 \text{ rad} \qquad p = 9.141 \cdot 10^{-4} \qquad \sigma = 6.04 \cdot 10^{-2} \text{ rad} \qquad p_0 = 0.227$$

Figure 15 gives a sketch of this model, to be compared with Figure 14, where all observed data is shown in a histogram. The model seems to fit the data rather well but the mean values for the discretised normal distributions seem to be a bit too large. This could be caused by randomness due to the small sample of data that was used as training data. Though, it could also be a sign that the observations are collected from some heavy tailed distribution. The points judged as noise is, as described above, set to zero, so the erroneous data is therefore not what caused the displacement of the distribution. Also, judging by the histogram in Figure 14, no evidence for heavy tailed distributions can be seen so the most probable cause of the high value on μ is the random selection of input data.

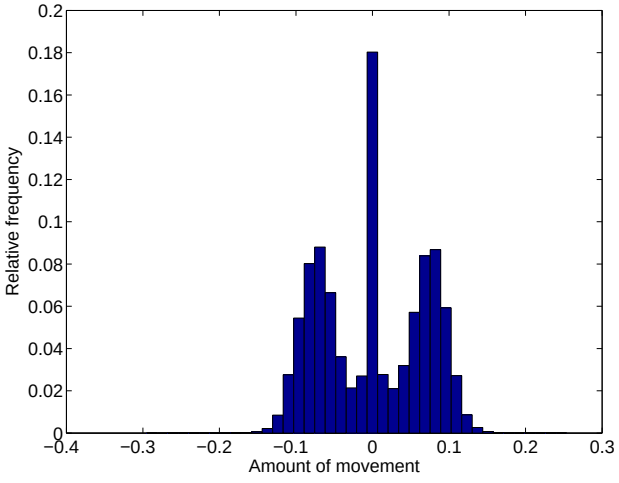


Figure 14: *Histogram over all observed movements with noise set to zero.*

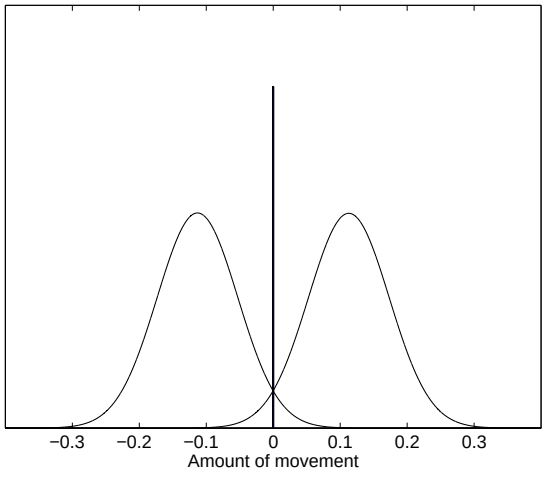


Figure 15: *Illustration of the distribution for observations from the HMM.*

The parameters for the second model, except the fixed parameter $\hat{p} = 0.5$, were also found with the learning algorithm, given seven data sets. These data sets was again picked at random from the complete observational data, so these was allowed to differ from the ones picked for the first model. The parameters that were found was

$$\hat{\mu} = 6.80 \cdot 10^{-2} \text{ rad} \qquad \hat{\sigma} = 2.80 \cdot 10^{-2} \text{ rad} \qquad p_0 = 0.227$$

The mean value is here significantly smaller than in the first model. This could be explained by the fact that it is harder to separate the two different directions for this model but the effect caused by the random choice of data could also be the cause. The value attained for this second model is also closer to the impression given by Figure 14, increasing the suspicion that the parameters for the first model is somewhat off target. The standard deviation, $\hat{\sigma}$ is also smaller compared to the first model. This is also likely to have been caused by the random selection of data sets. The same value for p_0 is used by both models since this value was based upon all observational data.

3.4 Model Comparison

The main difference between the two models presented is the extra parameter in the transition matrix for the first presented HMM. Complexity can always be added to a model to make it fit the observed data better. However, the model with this additional complexity is seldom the best model when it comes to predicting new data. This is since more complex models, by definition, can account for more complex observations. By a rule called *Occam's razor*, a simple model should be preferred to a more complex one if they both fit the data. [MacKay, 2003]

This preference for simplicity is actually included automatically with Bayesian model comparison. Comparing two models is done with the quotient

$$\frac{\mathrm{P}\big(M_1 \mid \mathbb{O}_1^T\big)}{\mathrm{P}\big(M_2 \mid \mathbb{O}_1^T\big)} = \frac{\mathrm{P}(M_1)\,\mathrm{P}\big(\mathbb{O}_1^T \mid M_1\big)}{\mathrm{P}(M_2)\,\mathrm{P}\big(\mathbb{O}_1^T \mid M_2\big)}$$

and since models with unjustified complexity will assign a smaller probability for the data, Occam's razor is here applied automatically. The purpose of this model comparison is thus to try to justify the extra complexity for the first presented HMM compared to the more simple HMM where all transition probabilities was set to 0.5.

The probabilities in the quotient above is not given directly from any of the algorithms covered. They can however be derived from the posterior distribution for the model parameters. As noted before, the posterior distribution is calculated as follows

$$\mathrm{P}\big(\theta, M \mid \mathbb{O}_1^T\big) = \frac{\mathrm{P}\big(\mathbb{O}_1^T \mid \theta, M\big)\,\mathrm{P}(\theta, M)}{\mathrm{P}\big(\mathbb{O}_1^T\big)}$$

Rearranging this gives

$$\mathrm{P}\big(\theta, M \mid \mathbb{O}_1^T\big)\,\mathrm{P}\big(\mathbb{O}_1^T\big) = \mathrm{P}\big(\mathbb{O}_1^T \mid \theta, M\big)\,\mathrm{P}(\theta, M)$$

Where the first factor on the right side is given by the forward-algorithm and $\mathrm{P}(\theta, M)$ is assumed to be uniform over all allowed parameter values and for both models compared. Thus, this product can be calculated for all parameter values and models, at the same time giving the values for the left product. By summing over all possible parameter values, the probability for the model, given the observed data is indirectly attained.

$$\sum_{\theta} \mathrm{P}\big(\theta, M \mid \mathbb{O}_1^T\big)\,\mathrm{P}\big(\mathbb{O}_1^T\big) = \mathrm{P}\big(\mathbb{O}_1^T\big)\,\mathrm{P}\big(M \mid \mathbb{O}_1^T\big)$$

Further, since $\mathrm{P}\big(\mathbb{O}_1^T\big)$ is the same for all models, it can be cancelled out and therefore the required quotient is attained

$$\frac{\mathrm{P}\big(\mathbb{O}_1^T\big)\,\mathrm{P}\big(M_1 \mid \mathbb{O}_1^T\big)}{\mathrm{P}\big(\mathbb{O}_1^T\big)\,\mathrm{P}\big(M_2 \mid \mathbb{O}_1^T\big)} = \frac{\mathrm{P}\big(M_1 \mid \mathbb{O}_1^T\big)}{\mathrm{P}\big(M_2 \mid \mathbb{O}_1^T\big)}$$

If this quantity is between 0 and 1, the second model, M_2, is the one more likely and if it is larger than 1, the first model, M_1 is more likely. For the two models presented in this thesis, the logarithms of these probabilities were calculated, giving

$$\log\big(\mathrm{P}\big(\mathbb{O}_1^T\big)\,\mathrm{P}\big(M_1 \mid \mathbb{O}_1^T\big)\big) \approx -265750$$
$$\log\big(\mathrm{P}\big(\mathbb{O}_1^T\big)\,\mathrm{P}\big(M_2 \mid \mathbb{O}_1^T\big)\big) \approx -294204$$

Where M_1 represent the first presented hidden Markov model and M_2 represent the model where all transition probabilities were equal. So calculating the logarithm of the quotient gives

$$\log\left(\frac{\mathrm{P}\big(\mathbb{O}_1^T\big)\,\mathrm{P}\big(M_1 \mid \mathbb{O}_1^T\big)}{\mathrm{P}\big(\mathbb{O}_1^T\big)\,\mathrm{P}\big(M_2 \mid \mathbb{O}_1^T\big)}\right) = \log\big(\mathrm{P}\big(M_1 \mid \mathbb{O}_1^T\big)\big) - \log\big(\mathrm{P}\big(M_2 \mid \mathbb{O}_1^T\big)\big) \approx 28456 \quad \Leftrightarrow$$
$$\frac{\mathrm{P}\big(M_1 \mid \mathbb{O}_1^T\big)}{\mathrm{P}\big(M_2 \mid \mathbb{O}_1^T\big)} \approx e^{28456}$$

This clearly indicates that the hidden Markov model is much more suitable than the simple model where all movements were independent. The extra complexity in the previous model is thus justified empirically.

3.5 Finding Changes of Direction

As discussed above, with the use of this model, the changes of direction for a single prawn is equivalent to the transitions between different states in the HMM. Therefore, the use of the Viterbi algorithm is an automatic way of finding out where the prawn actually changed its direction. If this data would be available and reliable, further research could be made about why these changes of directions occur. Suitably, as a part of earlier research, this specific experiment was manually examined and the actual time points for change of direction were marked for each prawn. This gave the opportunity to test how well the Viterbi algorithm suffices to find the actual changes of direction.

The first thing examined by these tests was the difference in number of transitions for the automatic procedure and the manual procedure. The second thing that was examined is the distance between the corresponding transitions coming from different procedures.

The number of transitions were counted for each data set to be able to easily compare the procedures. For the manual sequences, the mean value for the number of transitions was 2.39 and for the Viterbi sequences, the mean value for the number of transitions was 4.67. Since the transitions given by the Viterbi algorithm were found to be more numerous, the number of transitions manually marked were subtracted from these to construct a mainly positive difference. This difference is showed in Figure 16.

The first conclusion to be drawn is that a substantial amount of data sets showed equal number of transitions for the two methods, showing as the bar at zero. This is a good sign, showing that the model is somewhat accurate in describing the movement of a prawn. However, some data sets did not have the same number of transitions for the two procedures. Especially, some data sets

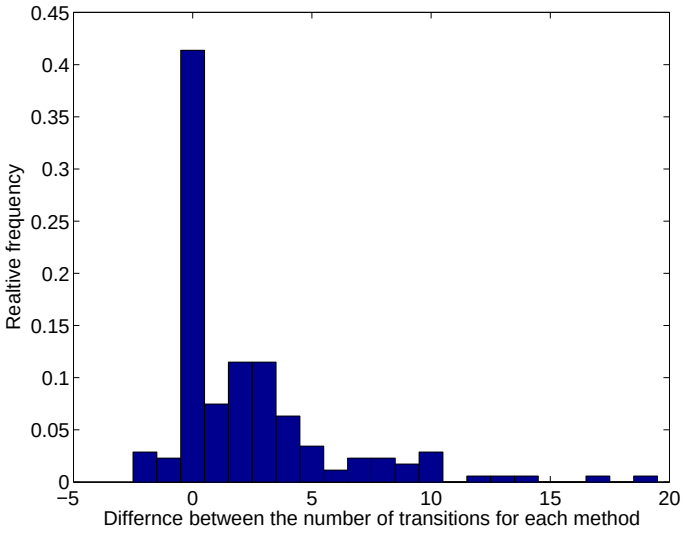


Figure 16: *The number of transitions given from the Viterbi sequence subtracted with the number of manually marked direction changes for each data set.*

showed more transitions for the Viterbi sequence than the manually marked transitions indicated. The largest difference for the number of transitions was 19 and this specific data set is showed in Figure 17.

In this figure, it is however evident that the Viterbi sequence correspond very well to the observed data. All transitions between two different states in the automatically found sequence has corresponding evidence in the data. It should be subject for further studies why the data shows evidence for some change of direction on time points where no manual mark is present. One reason could be lack of precision in the manually marked data due to mistakes and another possibility could be faulty measurements. The latter cause is not unlikely since the data has in obvious ways shown prone to noise before.

In Figure 18, the set with the most transitions of all the data sets with the same number of transition for both procedures is shown. With a close inspection it is evident that the sequences in this case are extremely consistent with each other. This is a good indication on how well this model can, in the best case scenario, automatically pick out the points where changes of direction occurred.

Before measuring the distance between corresponding transitions, some explanation about what is meant by corresponding transitions is needed. It is clear that if the number of transitions are not equal for the two procedures, there cannot be any one-to-one correspondence. Thus, all measurements used here are from data sets where the difference in number of transitions is zero. Further, the method used is to measure the distance to the transition in the Viterbi sequence closest to the chosen manually marked transition. An overview over all these distances is displayed in Figure 19.

This figure shows that a vast majority of the measured distances are within 15 frames, equivalent to 1 second, 122 of 137 to be precise. Also 71 out of 137 compared transitions turned out to be placed at the exact same time point. This is an extremely good result when considering that deciding where the change of direction actually occurred is somewhat vague, even if done manually.

Figure 20 shows the two data sets where the largest distances was found. In both of these, the section between the badly matched transitions show observations close to zero. These are situations

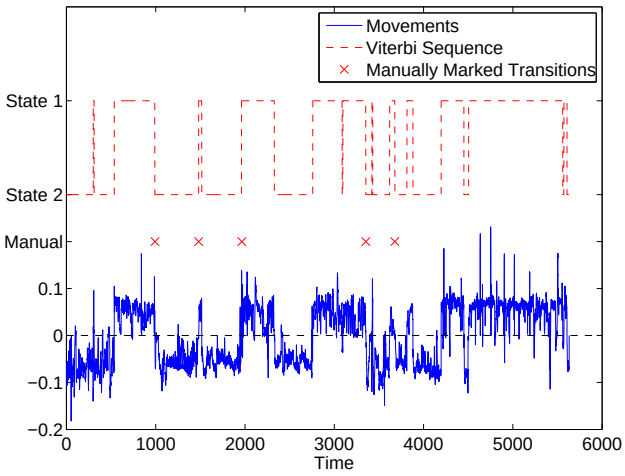


Figure 17: *The observations and the transition sequences where the largest difference in number of transitions was found. The Viterbi sequence contained 19 additional transitions compared to the manually marked sequence*

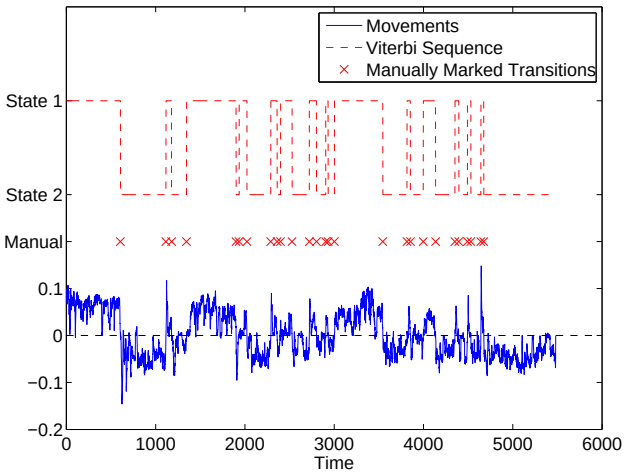


Figure 18: *The data set among those with equal number of transitions for both methods that showed the largest number of transitions in total. Both sequences contained 27 direction changes.*

where it theoretically should be difficult for the Viterbi algorithm to find the correct time points. Since observations close or equal to zero is approximately equally probable for both directions, it is difficult to decide where the prawn is headed. Observations close to zero can be seen as a limit case and it should be hard for *any* model to find the correct time points for transitions there.

3.6 A Logistic Regression Approach to Prawn Direction Changes

With the data available, some simple tests and inference could be made to find out something about why the prawns changed direction. The hypothesis that was tested here is basically that a prawn's propensity to change direction is dependent in some way of the distance to the other prawns in the aquarium. The method that was used to test this is logistic regression.

Regression is a method to examine the relationship between a single dependent variable, called the response variable, and several regressor variables. One of the goals is to estimate the parameters governing the relationship. If the response variable depend on the regressor variables linearly, the model used is linear regression. The computational advantages of this procedure is significant, since the least squared estimates of the parameters are found by some matrix multiplications, making the linearity assumption desirable. [Montgomery, 2008]

If the response variable is binary, logistic regression is suitable, making use of the linearity assumption through a transformation. Since the expected value for a binary random variable $Y \in \{0, 1\}$, given the regressor variables X, is equal to the probability $p(X) = \mathrm{P}(Y = 1 \mid X)$, logistic regression use the probability parameter p as the dependent variable. Further, p is transformed using the link function logit, making sure that the value for $p(X)$ is always in the interval $[0, 1]$

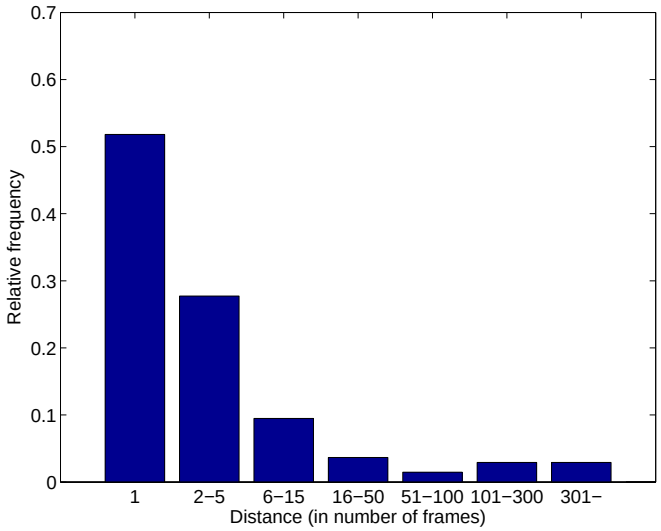


Figure 19: *Distance from the Viterbi transitions to the closest manually marked direction change. These distances was measured for each transition in the data sets with equal number of transitions for both sequences.*

[Hosmer and Lemeshow, 2000]. This specifies the model used in the experiment as

$$\log(\frac{p(X)}{1+p(X)}) = \beta_0 + \beta_1 X_1 + \beta_2 X_2$$

Where X_1 is the distance to the closest neighbour and X_2 is the distance to the second closest neighbour at the time point studied. In the experiment where the data was taken from, only three prawns were in the aquarium at the same time, so no more regressor variable is reasonable. The parameters that should be estimated are β_0, β_1 and β_2. The first parameter, β_0 is the intercept, giving the value of the response variable if theoretically all prawns were at the same spot. The second and third parameters β_1 and β_2 are the quantities governing the effect X_1 and X_2 has on the output. The amount of effect for one regressor variable is proportional to the absolute value of the corresponding parameter. Further, the sign of the parameters determine in which way the regressor variables effect the output.

A standard general linear model algorithm was used to estimate the model parameters and the result for the prawn data was

$$\beta_0^* = -5.9932 \qquad \beta_1^* = -0.010 \qquad \beta_2^* = -0.0005$$

The p-value for β_2 was evaluated to 0.165, thus there was no evidence for β_2 to be non-zero. However the p-values for both β_0 and more importantly β_1 was almost equal to zero, giving that both of these parameters were evidently non-zero. Further β_1^* is negative, indicating that the nearest neighbour makes it more probable for the prawn to change direction, the closer it get. The first parameter is not of great interest but the negative value is indicating that the chance of changing direction is quite small even if the closest neighbour is really close. This is since $\beta_0 = 0$ would have given $p(0) = 1/2$. So, to conclude, as the distance to the closest neighbouring prawn gets smaller, the probability of changing direction at a specific time point gets bigger.

There is a large number of tests and developments that could have been done in connection with this procedure of logistic regression. The objective for this section was however only to give a brief example of possible research connected to the application of HMMs to prawn movements.

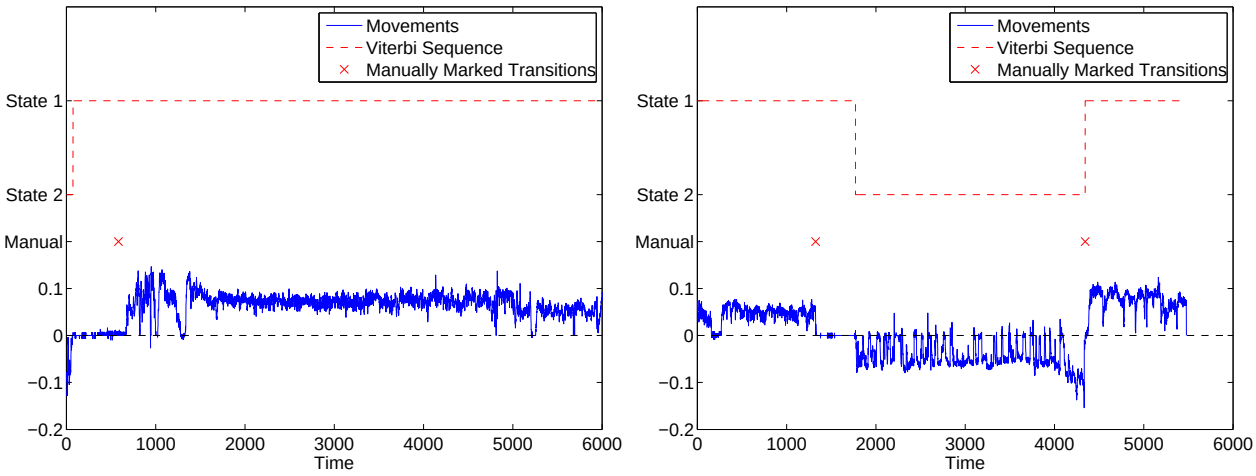


Figure 20: *The two data sets where the largest distances between two corresponding transitions was found.*

4 Conclusions

In this thesis the concept of hidden Markov models has been presented as an extension to Markov chains with a capacity of expressing more complex dependency structures. Each state in the hidden Markov chain is simply associated with a distribution of observations. Hidden Markov models are also characterised by the relatively few parameters included so they are simple and easy to grasp.

The algorithms concerning inference for hidden Markov models was efficiently implemented using simple recursive formulas. The Forward-Backward algorithm and the Viterbi algorithm was used to calculate and find information about the state sequence, given the observation sequence. These algorithms was found to perform better when there was a large difference between the states in the model while the performance of the learning algorithm, used to fit the model parameters to the observed data, was found to be dependent on the size of the input data. For larger and larger data sets, the result of the learning algorithm converged towards the correct answer.

One of the possible applications of hidden Markov models is the movements of a prawn and a specific example of these models was found to be very useful in this biological application. A basic hidden Markov model with one state associated with 'heading clockwise' and one state associated with 'heading counter clockwise' was fitted with the use of experimental data. The subject of interest was here the time points for direction changes so the Viterbi algorithm was used to find the most likely transitions between states. These transitions could be compared directly with manually found time points of direction changes. The transitions found by the Viterbi algorithm was often more numerous than the manually marked transitions. One example showed however that the sequence found by the Viterbi algorithm were more consistent with the observed movements than the manually marked transitions. Among the data sets with equal number of transitions, a vast majority of the corresponding transitions were not separated by more than a second.

Finally, with logistic regression, the probability of direction changes was found to increase as the distance to the closest neighbour decreased. This is a good example of how this data can be used for research about the decision process governing the prawns collective behaviour.

References

[Alm and Britton, 2008] Alm, S. and Britton, T. (2008). *Stokastik*. Liber, Stockholm.

[Baldi and Brunak, 2001] Baldi, P. and Brunak, S. (2001). *Bioinformatics: the machine learning approach*. MIT Press.

[Bishop, 2006] Bishop, C. (2006). *Pattern recognition and machine learning*, volume 4. Springer New York.

[Cappé et al., 2005] Cappé, O., Moulines, E., and Rydén, T. (2005). *Inference in hidden Markov models*. Springer Verlag.

[Chapra, 2006] Chapra, S. (2006). *Applied numerical methods with MATLAB for engineers and scientists*. McGraw-Hill Science/Engineering/Math.

[Durbin, 1998] Durbin, R. (1998). *Biological sequence analysis: Probabilistic models of proteins and nucleic acids*. Cambridge university press.

[Hosmer and Lemeshow, 2000] Hosmer, D. and Lemeshow, S. (2000). *Applied logistic regression*, volume 354. Wiley-Interscience.

[MacKay, 2003] MacKay, D. (2003). *Information theory, inference, and learning algorithms*. Cambridge Univ Pr.

[Montgomery, 2008] Montgomery, D. (2008). *Design and analysis of experiments*. John Wiley & Sons Inc.

[Rabiner, 1989] Rabiner, L. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

[Stirzaker, 2005] Stirzaker, D. (2005). *Stochastic processes and models*. Oxford University Press, USA.

[Viterbi, 1967] Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, 13(2):260–269.

A Implementation in Matlab

A.1 The Forward Algorithm

```
function [forw] = forwardProb(Obs, P , Q, pi)
    %FORWARDPROB(Obs, P, Q, pi) computes the forward probabilities for
    % the observation-sequence Obs (vector), based on the model
    % corresponding to the transitional matrix P, observation matrix Q,
    % and initial distribution pi.

    %Initializing
    [~,n] = size(P);
    T = length(Obs);
    forw = zeros(T,n);

    % Computing the forward probabilities.
    forw(1,:) = pi.*(Q(:,Obs(1)))';
    for t=2:T
        for j=1:n
            forw(t,j) = (forw(t-1,:)*(P(:,j))) * Q(j,Obs(t));
        end
    end
end
```

A.2 The Backward Algorithm

```
function [backw] = backwardProb(Obs, P , Q)
    %BACKWARDPROB(Obs, P, Q, pi) computes all the backward
        probabilities
    % for the observation-sequence Obs (vector), based on the model
    % corresponding to the transitional matrix P, observation matrix Q,
    % and initial distribution pi.

    % Initializing
    [n,~] = size(P);
    T = length(Obs);
    backw = ones(T,n);

    %Compute the backward probabilities
    for t=(T-1):(-1):1
        rightVector=Q(:,Obs(t+1)).*backw(t+1,:)';
        backw(t,:) = P*rightVector;
    end
 end
```

A.3 The Forward-Backward Algorithm

```
function [gamma] = forward_backward(Obs, P, Q, pi)
    %FORWARD_BACKWARD(Obs, P, Q, pi) computes the probability vectors
    % P(x_t = S_i|Obs) for the hidden markov chain for each state S_i
        and
    % for time points t from 1 up to T, where T is the last time point
    % in Obs, the observational sequence. This is done by using
    % the forward- and the backward- probabilities.
    % The model is described by the transitional matrix P, observation
    % matrix Q, and initial distribution pi.

    % Computing the forward-backward probabilities in external
        functions.
    forw = forwardProb(Obs, P, Q, pi);
    backw = backwardProb(Obs, P, Q);

    %Computing the different probability vectors
    gamma = forw.*backw;

    for t=1:length(Obs)
        gamma(t,:) = gamma(t,:)/(forw(t,:)*((backw(t,:))'));
    end
end
```

A.4 The Viterbi Algorithm

```
function [prob, seq ] = viterbi(Obs, P, Q, Pi)
    %VITERBI(Obs, P, Q, Pi)
    % Computes the probability for and find the jointly most probable
    % sequence of states for the hidden markov chain, given the
    % observation sequence Obs. The model corresponding to the
    % transitional matrix P, the observation matrix Q and the
    % initial distribution Pi.

    % Initialising constants
    T = length(Obs);
    [~,N] = size(P);

    % Construction of the vectors delta(i,t) and psi(i,t) where the
        first
    % argument correspond to the a state in the state space and the
        second
    % argument correspond to a time point.
    delta = zeros(N,T);
    psi = zeros(N,T);
```

```
    % Initialising
    delta(:,1)=Q(:,Obs(1)).*Pi';

    % Computing the delta- and psi- sequences.
    for t=2:T
        for j=1:N
            [X, I] = max(delta(:,t-1).*P(:,j));
            delta(j,t) = X*Q(j,Obs(t));
            psi(j,t) = I;
        end
    end
    [X,I] = max(delta(:,T));
    prob = X;
    seq = zeros(1, T);
    seq(T) = I;
    for t=(T-1):(-1):1
        seq(t) = psi(seq(t+1),t+1);
    end
end
```

A.5 The Learning Algorithm

```
function [lppn, pMLL, qMLL, pExp, qExp, pVar, qVar] = training(obs, N)
    %TRAINING An algorithm using Bayesian statistics to examine
    %the distribution of the parameters determining this specific
    %hidden Markov model. Obs is the observation sequence and N is
    %the number of points on each side of the grid used for
    %evaluation. The posterior distribution is given in
    %the log-probability domain.

    % Number of parameters:
    numb_par = 2;

    % Construct the parameter space
    pMin = 1/N;
    pMax = 1-1/N;
    par=linspace(pMin,pMax,N);

    M=N^numb_par;
    lppn = zeros(N); %Logarithmic Posterior Probability Non-normalized
    T=length(obs);

    %The initial and apriori distribution is assumed to be uniform
    pi=[0.5, 0.5];
    logPrio = - log(M);

    %% Compute the posterior probabilities
```

```
for indexP=1:N
    p=par(indexP);
    P=[p 1-p; 1-p p];
    for indexQ=1:N
        q=par(indexQ);
        Q=[0.5 0.5; q 1-q];
        lalpha = logforwardProb(obs, P, Q, pi);
        m=max(lalpha(T,:));
        lppn(indexP,indexQ) = m+log(sum(exp(lalpha(T,:)-m))) +...
                                    +logPrio;
    end
end

%% Maximum Likelihood
[row,col] = find(lppn==max(max(lppn)));
pMLL = par(row);
qMLL = par(col);

%% Find the normalizing factor and the marginal Distributions

% The maximum value for every row
mVecRow=max(lppn,[],2);
% The maximum value for every column
mVecCol=max(lppn);
%The maximum value, in total.
mx = max(mVecRow);
% A matrix of the same size as lppn, with all elements equal to the
%maximum values for lppn, on each row respectively.
maxMatRow = repmat(mVecRow,1,N);
% A matrix of the same size as lppn, with all elements equal to the
%maximum values for lppn, on each column respectively.
maxMatCol = repmat(mVecCol,N,1);

% The logarithm of the non-normalized marginal distributions
%for every p, log(P'(p)).
logProbMargPn = mVecRow + log(sum(exp(lppn-maxMatRow),2));
% The logarithm of the non-normalized marginal distributions
%for every q, log(P'(q)).
logProbMargQn = mVecCol + log(sum(exp(lppn-maxMatCol),1));

% The normalizing factor, e.g. the sum of all non normalized
% posterior probabilities.
log_norm_factor = mx + log(sum(exp(logProbMargPn-mx)));

% Normalized log_probabilities
logProbMargP = logProbMargPn - log_norm_factor;
mxP=max(logProbMargP);
```

```
logProbMargQ = logProbMargQn - log_norm_factor;
mxQ=max(logProbMargQ);

%% Expectation of p
pExpt=zeros(N,1);
for indexP=1:N
    %Compute the marginal distribution divided by exp(mx).
    pProb=exp(logProbMargP(indexP)-mxP);
    %Compute the actual term for the expected value
    pExpt(indexP)=par(indexP)*pProb;
end
%Adjust for exp(mx), sum over all values
lpExp=mxP+log(sum(pExpt));
% The real expected value of p
pExp=exp(lpExp);

%% Expectation of q
qExpt=zeros(N,1);
for indexQ=1:N
    %Compute the marginal distribution divided by exp(mx).
    qProb=exp(logProbMargQ(indexQ)-mxQ);
    %Compute the actual term for the expected value
    qExpt(indexQ)=par(indexQ)*qProb;
end
%Adjust for exp(mx) and the normalizing factor, sum over all values
lqExp=mxQ+log(sum(qExpt));
% The real expected Value of p
qExp=exp(lqExp);

%% Variance of p
%Expectation for p-squared
pSquaredExpected=zeros(N,1);
for indexP=1:N
    p=par(indexP);
    pSq = p^2;
    %Compute the marginal distribution divided by exp(mx).
    pProb=exp(logProbMargP(indexP)-mxP);
    %Compute the actual term for the expected value
    pSquaredExpected(indexP)=pSq*pProb;
end
lpSquaredExpect = mxP + log(sum(pSquaredExpected));
pSquaredExpect = exp(lpSquaredExpect);
pVar= pSquaredExpect - pExp^2;

%% Variance of q
%Expectation for q-squared
qSquaredExpected=zeros(N,1);
```

```
    for indexQ=1:N
        q=par(indexQ);
        qSq = q^2;
        %Compute the marginal distribution divided by exp(mx)
        qProb=exp(logProbMargQ(indexQ)-mxQ);
        %Compute the actual term for the expected value
        qSquaredExpected(indexQ)=qSq*qProb;
    end
    lqSquaredExpect=mxQ + log(sum(qSquaredExpected));
    qSquaredExpect=exp(lqSquaredExpect);
    qVar= qSquaredExpect - qExp^2;

end
```