Imperial College London
Department of computing

Hidden Markov Models : A Continuous-Time Version of the Baum-Welch Algorithm

by

Mohamed ZRAIAA (M.Z)

Submitted in partial fulfilment of the requirements for the MSc Degree in Advanced Computing of Imperial College London

SEPTEMBER 2010

Abstract

Hidden Markov Models (HMMs) are a popular and widespread tool for modeling a large range of time series data and for generating sequences of dependent observations. HMMs have been applied successfully to various complex problems, being especially effective with those requiring a huge amount of measured data, such as workload access patterns in computer storage systems and pattern recognition in speech, handwriting and music, to name but a few. Nowadays, the theory of HMMs is well developed in the case of finite, discrete processes. However in many applications, such as storage workload modelling [1], which we discuss in some detail, discrete models are not ideal and we prefer a continuous-time approach. The aim of this project is to develop such a new theory in continuous time, based on the discrete HMMs, in particular an adaptation of the so-called *Baum-Welch* algorithm to facilitate the efficient computation of the relevant parameters of a HMM. Therefore this thesis first introduces the main concepts related to HMMs in the discrete case, then transfers its focus to a particular application, namely workloads in FLASH memory systems, while trying to spell out the limitations of the discrete approach. Finally a continuous-time approach is developed that leads to the implementation of a new, but similar, version of the *Baum-Welch* algorithm in a very simple context. The substantially increased complexity inherent in a continuous-time analysis, compared with discrete time, obviates the numerical analysis of models with more than two hidden states in the timescale of this project, but the theoretical and algorithmic extension to any number of hidden states is explained as "future work".

Acknowledgments

First of all I would like to address a special thank to my supervisor Peter Harrsion whose help and advise were invaluable. He was always around to answer my questions or give me relevant orientations about this project, sacrificing his own free time. Then I would like to thank Sarah Leyton, although we only meet once, her previous work was of great help to master the concept of HMMs. Finally I thank the Department of Computing of Imperial College for its perfect organization and for all the facilities we have at our disposal.

Contents

1 Introduction

Hidden Markov Models (HMMs) and their generalizations are used nowadays in many different areas, essentially providing parsimonious descriptions for time series that can be very long. Consider, for example, a simulation of the behaviour of a complex computer system or the logistics of a large hospital. Models of these systems require some representation for their input, namely when does the next instruction arrive at a computer and what type it is or when does the next patient enter the hospital and for what reason? It is very important to be able to describe such inputs in a concise way: HMMs often offer this possibility.

Hidden Markov Models were originally reputed for their application in speech processing which has occurred mainly within the past few years. The basic theory was published in a series of classic papers by Baum and his colleagues in the late 1960s and early 1970s and was implemented for speech recognition applications by Baker, Jelinek and their colleagues in the 1970s. The large period between the establishment of this theory and its intensive application is due to several reasons. First, the basic concept of Hidden Markov models was published in mathematical journals which were not generally read by engineers working on the specific fields where HMMs were useful. The second reason was that the original applications of the theory did not provide sufficient tutorial material for most readers to understand the theory and to be able to apply it to their own research. As a result, several tutorial papers were written which provided a sufficient level of detail for a number of research laboratories to begin work using HMMs.

We can enumerate three basic problems associated with HMMs. The first problem is, given the parameters of the model λ, compute the probabilities of a particular sequence of observations $P(O \mid \lambda)$ with $O = (o_1, o_2, ..., o_n)$ the observation sequence. This problem is solved by the *forward-backward algorithm*. Secondly, find the optimal sequence of hidden states that could have generated a given sequence of observations. This can be solved by *posteriori* statistical inference, resulting in the *Viterbi algorithm*. Thirdly, given a sequence of observations, or set of such sequences, find the most likely set of model parameters. This issue is solved by the *Baum-Welch algorithm*, using standard techniques of statistical inference.

This thesis continues previous work partly carried out by my supervisor Pete Harrison in a paper "Storage Worload Modelling by Hidden Markov Models: Application to FLASH Memory" [1], even though I will focus on a more general problem. One of the major issues discussed in this paper is to find a good representation of a flow of read and write instructions arriving at a FLASH memory (playing the role of a server), which treats these instructions according to their type and the sizes of their data transfers – obviously writes have a heavier demand than reads. However to achieve such a complex model requires a powerful tool that is able to take into account several crucial parameters of this continuous-time flow. The authors finally chose to use HMMs. Moreover, to overcome the difficulty brought about by the inherent continuous-time process, they decided to split the time axis into 5-ms bins, and in each bin they stored the number of reads and writes, resulting in a data trace that can be more easily processed. Then, the objective was to derive the parameters of the HMM by using the so-called *Baum-Welch algorithm*. They subsequently tried to reduce the size of the bins to 1 ms, but they noticed that most of the bins were empty (no instruction in it) which led to an inefficient and falsely correlated traffic model.

The next natural step of this previous work is, rather than using bins and hence discrete chains to model the input data stream, to try to use continuous-time stochastic processes to describe the flow of instructions. This obviously brings some difficulty compared to the discrete-time case and we need to make some assumptions about these processes. All this

work represents the central part of my project and it leads to a continuous-time version of the *Baum-Welch algorithm*. A much more complicated (continuous-time) form of the Baum-Welch algorithm is derived and is applied exlicitly to the case of a HMM with two hidden states. Some numerical results are obtained from a time-stamped trace monitored from Flash memory accesses, and the numerical extension to higher-order models (with more than two hidden states) is outlined.

This thesis will be organized as follows. In section 2, we present the mathematical background of the project, defining some important properties of HMMs while adopting a formal notation (inspired from [3]), necessary for the rigorous development of the theory. This is also where we deal with the three basic problems discussed earlier on. In section 3, we give an overview of previous work and identify the important aspects and assumptions concerning the continuous-time version. Section 4 develops the mathematical theory leading to the derivation of a continuous-time version of the *Baum-Welch algorithm*. Section 5 gives some preliminary numerical results.

2 Definition, properties and three basic problems related to HMMs

In this section, we define fully what is a Hidden Markov Model, what are its main properties and how to solve the three main issues we identified earlier on. Generally, when one comes to observe and characterise a series of unbounded counts (random variables), the easiest and simplest model assumes that the counts are generated by a homogeneous Poisson process, so that the counts are independent identically distributed Poisson random variables. Unfortunately it turns out that these models are too simple and don't correspond to real-life processes (especially, in practice, the variance is greater than the mean, which is called an overdispersion, whereas they should be equal given the Poisson distibution). Now if we assume that this series of counts is driven by an underlying process, which is a Markov Chain, then clearly the resulting process of counts will allow for serial dependence in addition to overdispersion: this is what we call a Hidden Markov Model. The term "Hidden" is ambiguous because it only concerns the underlying parameter process, namely the Markov chain which is assumed to be unobservable. Let us go into details and establish the notation for the remainder of this project.

2.1 Hidden Markov Models : Definition

We denote by $\{S_t\}_{t\in\mathbb{N}}$ a discrete time stochastic process. These stochastic processes are further restricted to a finite or countable set. Usually, a stochastic process is combined with a Markov chain $\{C_t\}_{t=0,1,...}$ which is a stochastic process where the transition probabilities depend only on the previous state. Note that, subsequently, we will consider that the transition probabilities are independent of time, and so the Markov chain is said to be *homogeneous* or *stationary*. The Markov chain is not observed directly; therefore, all statistical inference has to be done in terms of $\{S_t\}_{t=0,1,...}$, the associated stochastic process.

Definition 2.1 *The stochastic process* $\{C_t\}_{t=0,1,...}$ *with state space* $\mathcal{C}$ *is called a* Markov Chain *if, for each* $t \in \mathbb{N}$

$$P(C_{t+1} = c_{t+1} \mid C_t = c_t, C_{t-1} = c_{t-1}, ..., C_1 = c_1) = P(C_{t+1} = c_{t+1} \mid C_t = c_t)$$

for all possible values of $c_0, c_1, \ldots, c_t, c_{t+1} \in \mathcal{C}$. This is called the Markov Property.

The transition matrix $P = (p_{cc'})_{c,c' \in \mathcal{C}}$ *is the matrix of* one step transition probabilities $p_{cc'} = P(C_{t+1} = c' \mid C_t = c)$. *Note that* P *is a stochastic matrix, that is* $p_{cc'} \geq 0$ *and* $\sum_{c' \in S} p_{cc'} = 1, \quad c \in \mathcal{C}$.

Note 2.2 We will denote by C_s^t the collection $C_s, ..., C_t (s \leq t)$ i.e. $C_s^t = c_s^t$ means $C_s = c_s, ..., C_t = c_t$.

Proposition 2.3 The transition matrix P and the initial distribution of the chain $\nu_c = P(C_0 = c), c \in \mathcal{C}$, completely determine the chain in the sense that, for n = 1, 2, ...,

$$P(C_0^n = c_0^n) = \nu_{c_0} p_{c_0,c_1} p_{c_1,c_2} ... p_{c_{n-1},c_n}$$

Proof *The previous result can easily be proven by induction on n. Initialisation: for* $n = 1$,

$$\begin{aligned} P(C_0 = c_0, C_1 = c_1) &= P(C_1 = c_1 \mid C_0 = c_0) P(C_0 = c_0) \\ &= p_{c_0,c_1} \nu_{c_0} \end{aligned}$$

Suppose that the proposition is true for some $n \in \mathbb{N}$

$$P(C_0^n = c_0^n) = \nu_{c_0} p_{c_0,c_1} p_{c_1,c_2} ... p_{c_{n-1},c_n}$$

We would then have

$$\begin{aligned} P(C_0^{n+1} = c_0^{n+1}) &= P(C_{n+1} = c_{n+1} \mid C_0^n = c_0^n) P(C_0^n = c_0^n) \\ &= P(C_{n+1} = c_{n+1} \mid C_n = c_n) P(C_0^n = c_0^n)(MP) \\ &= P(C_0^n = c_0^n) = \nu_{c_0} p_{c_0,c_1} p_{c_1,c_2} ... p_{c_n,c_{n+1}} \end{aligned}$$

Definition 2.4 (Hidden Markov Model) *Suppose that* $\{C_t\}_{t \in \mathbb{N}}$ *is a Markov chain with state space* $\mathcal{C} = \{1, ..., r\}$, *initial distribution* $\nu_c (c \in \mathcal{S})$ *and transition matrix* $P = (p_{cc'})_{c,c' \in \mathcal{C}}$, *and* $\{S_t\}_{t \in \mathbb{N}}$ *is a stochastic process taking values in* $\mathcal{S} = \{1, ..., m\}$. *The bivariate stochastic process* $\{C_t, S_t\}_{t \in \mathbb{N}}$ *is said to be a* Hidden Markov Model (HMM) *if it is a Markov chain with transition probabilities*

$$\begin{aligned} P(C_t = c_t, S_t = s_t \mid C_{t-1} = c_{t-1}, S_{t-1} = s_{t-1}) &= P(C_t = c_t, S_t = s_t \mid C_{t-1} = c_{t-1}) \\ &= q_{c_{t-1},c_t} \cdot g_{c_t,s_t} \end{aligned}$$

where $G = (g_{cs})_{c \in \mathcal{C}, s \in \mathcal{S}}$ *is a stochastic matrix.*

Note that the conditional probability of the joint state (C_t, S_t) depends only on C_{t-1} and not S_{t-1}. A useful device for depicting the dependence structure of such model is the conditional independence graph. In such a graph the absence of an edge between two vertices indicates that the two variables concerned are conditionally independent given the other variables. Figure 1 displays the independence of the observations $\{S_t\}$ given the states $\{C_t\}$ occupied by the Markov chain, as well as the conditional independence of C_{t-1} and C_{t+1} given C_t, i.e. the Markov property.

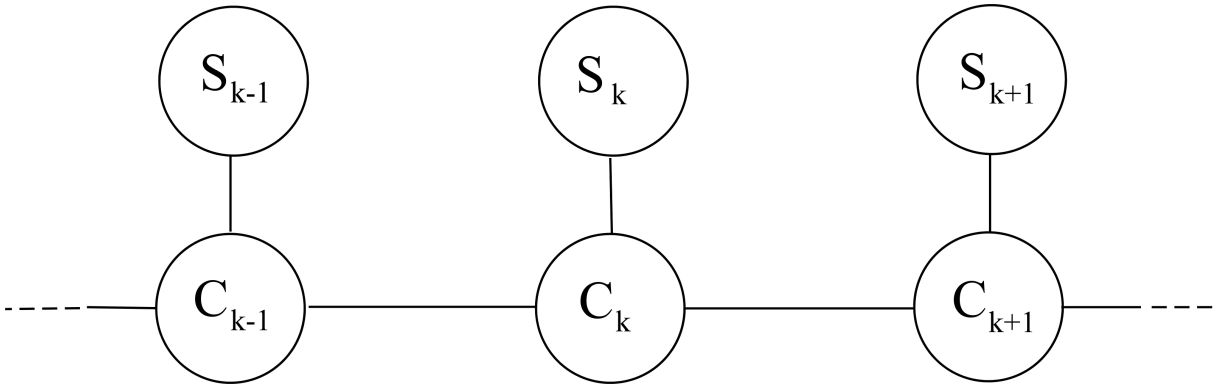

Figure 1: Conditional independence graph of hidden Markov model.

Note 2.5 The joint probability function of $C_0, S_0, ..., C_n, S_n$ is the most useful and the most complete information we can have about this bivariate process. It can be written (using proposition 2.3) as

$$\begin{aligned} J_{\nu,n}(c_0^n, s_0^n) &= P(C_0 = c_0, S_0 = s_0, ..., C_n = c_n, S_n = s_n) \\ &= P(C_0 = c_0, S_0 = s_0)P(C_1 = c_1, S_1 = s_1 \mid C_0 = c_0)... \\ &\quad ...P(C_n = c_n, S_n = s_n \mid C_{n-1} = c_{n-1}) \\ &= \nu_{c_0} g_{c_0,s_0} q_{c_0,c_1} g_{c_1,s_1} ... q_{c_{n-1},c_n} g_{c_n,s_n} \end{aligned}$$

This is therefore the full likelihood function representing the probability of joint processes. Given this full likelihood function, we can calculate any probability related to this HMM, and this is only possible if we can have access to the three sets of parameters ν_c, P and G.

2.2 Some important properties

In this section we will extract some interesting properties from the previous definition. We will also try to give an alternative approach to HMMs by giving an equivalent definition.

Proposition 2.6 $P(S_t = s_t \mid C_t = c_t) = g_{c_t,s_t}$

Proof

$$\begin{aligned}
P(S_t = s_t \mid C_t = c_t) &= \frac{P(S_t = s_t, C_t = c_t)}{P(C_t = c_t)} \\
&= \frac{1}{P(C_t = c_t)} \sum_{c_{t-1}} P(S_t = s_t, C_t = c_t, C_{t-1} = c_{t-1}) \\
&= \frac{1}{P(C_t = c_t)} \sum_{c_{t-1}} P(S_t = s_t, C_t = c_t \mid C_{t-1} = c_{t-1}) P(C_{t-1} = c_{t-1}) \\
&= \frac{1}{P(C_t = c_t)} \sum_{c_{t-1}} p_{c_{t-1}c_t} g_{c_t s_t} P(C_{t-1} = c_{t-1}) \\
&= \frac{g_{c_t s_t}}{P(C_t = c_t)} \sum_{c_{t-1}} P(C_t = c_t, C_{t-1} = c_{t-1}) \\
&= \frac{g_{c_t s_t}}{P(C_t = c_t)} P(C_t = c_t) \\
&= g_{c_t s_t}
\end{aligned}$$

One important thing concerning HMMs is the context of dependence of the random variables. Stating clearly which variables are independent or not is the key point and it would be very complicated to derive a single result without having done such a work. In fact it follows from the previous definition two major properties from the Hidden Markov Model $\{C_t, S_t\}_{t \in \mathbb{N}}$:

(a) Let $n \in \mathbb{N}$. Conditionnally on $\{C_t\}_{t=0,\dots,n}$, $\{S_t\}_{t=0,\dots,n}$ are independent, i.e

$$P(S_0^n = s_0^n \mid C_0^n = c_0^n) = \prod_{i=0}^{n} P(S_i = s_i \mid C_i = c_i)$$

(b) for each t, the conditional distribution of S_t depends on C_t only, i.e.

$$P(S_t = s_t \mid S_0^{t-1} = s_0^{t-1}, C_0^t = c_0^t) = P(S_t \mid C_t = c_t)$$

To prove (a), we use Proposition 2.3, Note 2.5 and Proposition 2.6 and we have

$$\begin{aligned}
P(S_0^n = s_0^n \mid C_0^n = c_0^n) &= \frac{P(S_0^n = s_0^n, C_0^n = c_0^n)}{P(C_0^n = c_0^n)} \\
&= \frac{\nu_{c_0} g_{c_0,s_0} \prod_{i=1}^{n} p_{c_{i-1},c_i} g_{c_i,s_i}}{\nu_{c_0} \prod_{i=1}^{n} p_{c_{i-1},c_i}} \\
&= \prod_{i=0}^{n} g_{c_i,s_i} \\
&= \prod_{i=0}^{n} P(S_i = s_i \mid C_i = c_i)
\end{aligned}$$

To prove (b), we only use (a) and we have

$$\begin{aligned} P(S_t = s_t \mid S_0^{t-1} = s_0^{t-1}, C_0^t &= c_0^t) = \frac{P(S_0^t = s_0^t, C_0^t = c_0^t)}{P(S_0^{t-1} = s_0^{t-1}, C_0^t = c_0^t)} \\ &= \frac{P(S_0^t = s_0^t, C_0^t = c_0^t)}{\sum_{s_t} P(S_0^t = s_0^t, C_0^t = c_0^t)} \\ &= \frac{P(S_0^t = s_0^t \mid C_0^t = c_0^t)}{\sum_{s_t} P(S_0^t = s_0^t \mid C_0^t = c_0^t)} \\ &= \frac{\prod_{i=0}^t P(S_i = s_i \mid C_i = c_i)}{\sum_{s_t} \prod_{i=0}^t P(S_i = s_i \mid C_i = c_i)} \\ &= \frac{\prod_{i=0}^t P(S_i = s_i \mid C_i = c_i)}{\prod_{i=0}^{t-1} P(S_i = s_i \mid C_i = c_i) \sum_{s_t} P(S_t = s_t \mid C_t = c_t)} \\ &= P(S_t = s_t \mid C_t = c_t) \end{aligned}$$

Proposition 2.7 We can state four similar properties for the stochastic process $\{S_t\}$ that could be useful. Note that for simplicity, we shall use a somewhat abbreviated notation and for instance the event $S_t = s_t$ is replaced by S_t simply.
Firstly, for $t = 0, 1, ...n$:

$$P(S_0^n \mid C_t) = P(S_0^t \mid C_t) P(S_{t+1}^n \mid C_t) \tag{1}$$

In the case $t = n$ we use the convention that $P(S_{t+1}^n \mid C_t) = 1$. Secondly, for $t = 0, 1, ..., n-1$:

$$P(S_0^n \mid C_t, C_{t+1}) = P(S_0^t \mid C_t) P(S_{t+1}^n \mid C_{t+1}) \tag{2}$$

Thirdly, for $1 \leq t \leq l \leq n$:

$$P(S_l^n \mid C_t^l) = P(S_l^n \mid C_l) \tag{3}$$

Finally, for $t = 0, 1, ..., n$:

$$P(S_t^T \mid C_t) = P(S_t \mid C_t) P(S_{t+1}^n \mid C_t) \tag{4}$$

Given the structure of the models, none of these properties is surprising, and this can be quite easily proven as follows.

Proof *Let us proof these four properties. All proofs follow the same structure, namely, we first express the probability of interest in terms of probabilities conditional on $\{C_t\}_{t=0,...n}$, then we use the fact conditionally on $\{C_t\}_{t=0,...n}$, $S_0, ..., S_n$ are independent and that S_t only depends on C_t. Finally we use the Markov property of $\{C_t\}$ if necessary. We will use 3 little lemmas to prove our claims.*

Lemma 2.8 *For all integers t and l such that $0 \leq t \leq l \leq n$:*

$$P(S_l^n \mid C_t^n) = P(S_l^n \mid C_l^n)$$

Proof. The left-hand side can be rewritten as:

$$\frac{1}{P(C_t^n)} \sum_{c_0, ..., c_{t-1}} P(S_l^n \mid C_0^n) P(C_0^n)$$

By independence, we have

$$\begin{aligned} P(S_l^n \mid C_0^n) &= P(S_l \mid C_0^n)...P(S_n \mid C_0^n) \\ &= P(S_l \mid C_l)...P(S_n \mid C_n) \end{aligned}$$

and this can be taken outside the summation where the resulting sum reduces to $P(C_t^n)$. The left-hand side is therefore $P(S_l \mid C_l)...P(S_n \mid C_n)$, which is independent of t. The right-hand side is just the case when $t = l$.

Lemma 2.9 *For $t = 0, ..., n-1$:*

$$P(S_{t+1}^n \mid C_0^t) = P(S_{t+1}^n \mid C_t)$$

Proof. The left-hand side can be rewritten as

$$\frac{1}{P(C_0^t)} \sum_{c_{t+1},...,c_n} P(C_0^n) P(S_{t+1}^n \mid C_0^n)$$

Now we apply the previous lemma twice and the Markov property to see that this equals

$$\sum_{c_{t+1},...,c_n} P(C_{t+1}^n \mid C_t) P(S_{t+1}^n \mid C_t^n)$$

The term in the sum is $P(S_{t+1}^n, C_t^n)/P(C_t)$, and the result of the sum is therefore $P(S_{t+1}^n \mid C_t)$ as required.

Lemma 2.10 *For $t = 0, ..., n$:*

$$P(S_0^t \mid C_0^n) = P(S_1^t \mid C_0^t)$$

Proof. As we have seen in the first lemma, the left-hand side is equal to $P(S_0 \mid C_0)...P(S_t \mid C_t)$. We can apply the same conditions of independence to derive the same term from the right-hand side.

Proof of Property (1). Making use of use of mutual independence of $S_1, ..., S_n$ given C_0^n, we can write the left-hand side of property (1) as

$$\frac{1}{P(C_t)} \sum_{c_0,...,c_{t-1}} \sum_{c_{t+1},...,c_n} P(C_0^n)(P(S_0^t \mid C_0^n) P(S_{t+1}^n \mid C_0^n))$$

Finally, by using the last two lemmas, we can show that this equals

$$\frac{1}{P(C_t)} P(S_{t+1}^n \mid C_t) \sum_{c0,...,c_{t-1}} P(S_0^t, C_0^t) = \frac{1}{P(C_t)} P(S_0^t, C_t) P(S_{t+1}^n \mid C_t)$$

which is the right-hand side.

Proof of Property (4). All we have to do is to sum the result of Property (1) with respect to $s_1, ..., s_{t-1}$.

Proof of Property (2). This is probably the most difficult one to prove. The left-hand side of Property (2) can be written as

$$\frac{1}{P(C_t, C_{t+1})} \sum_{c0,...,c_{t-1}} \sum_{c_{t+2},...,c_n} P(C_0^n) P(S_1^t \mid C_0^n) P(S_{t+1}^T \mid C0^n)$$

By the third and first lemmas respectively, the last two factors in the above expression reduce to $P(S_0^t \mid C_0^t)$ and $P(S_{t+1}^n \mid C_{t+1}^n)$. The Markov property of $\{C_t\}$ is then used, and after some routine manipulations of conditional probabilities it emerges that the above expression is equal to

$$P(S_0^t \mid C_t)\frac{1}{P(C_{t+1})}\sum_{c_{t+2},\ldots,n} P(S_{t+2}^n, C_{t+1}^n) = P(S_0^t \mid C_t)P(S_{t+1}^n, C_{t+1})/P(C_{t+1})$$

as required.

Proof of Property (3).Here again the left-hand side can be written as

$$\frac{1}{P(Ct,\ldots,Cl)}\sum_{c_{l+1},\ldots,c_t}\sum_{c0,\ldots,c_{t-1}} P(S_l^n \mid C_0^n)P(C_0^n)$$

Now by the first lemma

$$P(S_l^n \mid C_0^n) = P(S_l^n \mid C_l^n),$$

ans the above expression for the left-hand side therefore equals

$$\sum_{c_{l+1},\ldots,c_t} P(S_l^n \mid C_l^n)P(C_{l+1}^n \mid C_t^l)$$

By the Markov property of $\{C_t\}$, this equals

$$\begin{aligned}\sum_{c_{l+1},\ldots,c_t} P(S_l^n \mid C_l^n)P(C_{l+1}^n \mid C_l) &= \frac{1}{P(C_l)}P(S_l^n, C_l^n)\\ &= P(S_l^n, C_l)/P(C_l)\\ &= P(S_l^n \mid C_l)\end{aligned}$$

as required

2.3 Three main issues related to HMMs

Hidden Markov models have been widely and successfully used in many important areas of research such as speech recognition, hand-writing recognition, ... This is almost due to the fact that Hidden Markov models only need a small number of parameters to generate parsimonious representation of real-life process. However, several problems are to be overcome when one has come to speak about Hidden Markov models and in particular three problems which are of interest. The first one is given the parameters of the model (that we shall specify), one might want to compute the probabilities of a particular sequence of observations, and of the corresponding hidden state values. This problem is solved by the *forward-backward algorithm*. Secondly, one might want to find the most likely sequence of hidden states that could have generated a given sequence of observations. This problem is solved by a *posteriori* statistical inference, resulting in the *Viterbi algorithm*. Thirdly, given the a sequence of observations, or set of such sequences, one might want to find the most likely set of model parameters. This problem is solved by the *Baum-Welch algorithm*, a consequence of statistical inference. All these three problems have been solved and this is exactly this point that gives to HMMs their strength since each problem is computationally solvable. The following of this subsection describes in details the theoretical solution to these issues as well as providing a corresponding algorithm.

2.3.1 Forward-backward algorithm

This is probably the most important and the most fundamental algorithm about HMMs. The problem is given the model parameters $\lambda = [$ the initial distribution ν, the one-step transition matrix of the hidden Markov chain P, and the matrix given the conditional probabilities of one observation given the current hidden state $G\ ]$, what can we say about $S_1, ..., S_n$. Next we define some coefficients that are of great importance for the whole project.

Definition 2.11 *For positive integers k, l, n with $l \geq k$, we define the following conditional probabilities*

$$\phi_{k:l|n}(c_k^l, s_0^n) = P(C_k = c_k, ..., C_l = c_l \mid S_0 = s_0, ..., S_n = s_n)$$

We will use the notation $\phi_{k|n} = \phi_{k:k|n}$. Sometimes when the context is obvious, we shall drop the dependence on $s_0, ..., s_n$ and only write $\phi_k(c_k)$ to represent $\phi_{k|n}(c_k, s_0^n)$.

We will see soon how and why these quantities are important to the solution of this problem. Now let us state the likelihood function of the observations which is the function we want to compute for all possible sequence of observations.

Definition 2.12 *The likelihood function of the observations $s_0, s_1, ..., s_n$ is defined as*

$$L_{\nu,n}(s_0, s_1, ..., s_n) = P(S_0 = s_0, ..., S_n = s_n)$$

It can be seen easily that given the parameters of the Hidden Markov models, we are able to compute the likelihood function of the observations in this following way using *Note 2.5*:

$$\begin{aligned} L_{\nu,n} &= \sum_{c_0,...,c_n} J_{\nu,n}(c_0^n, s_0^n) \\ &= \sum_{c_0,...,c_n} \nu_{c_0} g_{c_0,s_0} \prod_{i=1}^{n} p_{c_{i-1},c_i} g_{c_i,s_i} \end{aligned}$$

So why don't we stop here ? As you can see, this formula indeed allows us to calculate the likelihood function of observations but you can also note that the right-hand side of the formula is a sum over a multi-dimensional domain and it is clearly computationally infeasible (it is exponential in n). For instance if we assume $n = 10$ and the number of hidden states equals 6, only small numbers, then the right-hand side is a sum of $6^{10} = 60466176$ terms, more than 60M terms for relatively small parameter values. This motivates us to find an alternative way to compute the likelihood function.

2.3.2 Expression of the forward and backward variables

To derive the expression of these famous variables, we must go further in the expression of $\phi_{k|n}(c_k, s_0^n)$ to see what is happening : for $0 \leq k \leq n$,

$$\begin{aligned}
\phi_{k|n}(c_k, s_0^n) &= P(C_k = c_k \mid S_0^n = s_0^n) \\
&= \frac{P(C_k = c_k, S_0^n = s_0^n)}{P(S_0^n = s_0^n)} \\
&= \frac{\sum_{c_0,\ldots,c_{k-1}} \sum_{c_{k+1},\ldots,c_n} J_{\nu,n}(c_0^n, s_0^n)}{P(S_0^n = s_0^n)} \\
&= \frac{1}{L_{\nu,n}(s_0^n)} \sum_{c_0,\ldots,c_{k-1}} \sum_{c_{k+1},\ldots,c_n} \nu_{c_0} g_{c_0,s_0} \prod_{i=1}^{n} p_{c_{i-1},c_i} g_{c_i,s_i} \\
&= \frac{1}{L_{\nu,n}(s_0^n)} \underbrace{\sum_{c_0,\ldots,c_{k-1}} \nu_{c_0} g_{c_0,s_0} \prod_{i=1}^{k} p_{c_{i-1},c_i} g_{c_i,s_i}}_{\alpha_{\nu,k}(c_k, s_0^k)} \underbrace{\sum_{c_{k+1},\ldots,c_n} \prod_{i=k+1}^{n} p_{c_{i-1},c_i} g_{c_i,s_i}}_{\beta_{k|n}(c_k, s_{k+1}^n)}
\end{aligned}$$

So we have

$$\phi_{k|n}(c_k, s_0^n) = \frac{1}{L_{\nu,n}(s_0^n)} \alpha_{\nu,k}(c_k, s_0^k)\, \beta_{k|n}(c_k, s_{k+1}^n)$$

These are the forward and backward variables : $\alpha_{\nu,k}(c_k, s_0^k)$ is called the forward variable or sometimes the forward measure and $\beta_{k|n}(c_k, s_{k+1}^n)$ is referred to as the backward variable or the backward measure. The forward measure does depend on the initial distribution ν whereas the backward variable does not. We will see in the next section why they are given these names. However given the previous expressions of α and β it is difficult to see which quantity they represent and so we have to go a bit further to give a full meaning to these variables.
For $k = 0, \cdots, n$

$$\begin{aligned}
\alpha_{\nu,k}(c_k, s_0^k) &= \sum_{c_0,\ldots,c_{k-1}} \nu_{c_0} g_{c_0,s_0} \prod_{i=1}^{k} p_{c_{i-1},c_i} g_{c_i,s_i} \\
&= \sum_{c_0,\ldots,c_{k-1}} J_{nu,k}(c_0^k, s_0^k) \\
&= P(S_0^k = s_0^k, C_k = c_k)
\end{aligned}$$

We finally conclude that $\alpha_{\nu,k}(c_k, s_0^k)$ represents the joint probability of the kth hidden state being c_k and the first k observations being s_0^k.
Now for $k = 0, \cdots, n-1$

$$\begin{aligned}
\beta_{k|n}(c_k, s_{k+1}^n) &= \sum_{c_{k+1},\ldots,c_n} \prod_{i=k+1}^{n} p_{c_{i-1},c_i} g_{c_i,s_i} \\
&= \sum_{c_{k+1},\ldots,c_n} \frac{P(C_k^n = c_k^n, S_{k+1}^n = s_{k+1}^n)}{P(C_k = c_k)} \\
&= \frac{P(C_k = c_k, S_{k+1}^n = s_{k+1}^n)}{P(C_k = c_k)} \\
&= P(S_{k+1}^n = s_{k+1}^n \mid C_k = c_k)
\end{aligned}$$

And by convention$\beta_{n|n}(c, s_n^n) = 1$.
Here we deduce that $\beta_{k|n}(c_k, s_{k+1}^n)$ represents the conditional probability of the last $n-k$ observations being s_{k+1}^n given that the k*th* hidden state is c_k. For more simplicity we will lighten the notation by omitting dependence on s_0^n in some of the expressions in the rest of this project. For example, we would use $L_{\nu,n}$ instead of $L_{\nu,n}(s_0^n)$, $\alpha_{\nu,k}(c_k)$ instead of $\alpha_{\nu,k}(c_k, s_0^k)$ and $\beta_{k|n}(c_k)$ instead of $\beta_{k|n}(c_k, s_{k+1}^n)$ (generally the observations are supposed to be known).

2.3.3 Properties of the forward and backward variables

In this subsection, we will outline three main properties relating the forward and backward variables to the likelihood function of observations. Assuming that the forward and backward variables can be efficiently computed, the next three properties will validate the usefulness of such variables. Here is one of the most important propositions concerning the forward and backward algorithm.

Proposition 2.13 *For* $0 \leq k \leq n$

(a) $\alpha_{\nu,k}(c)\beta_{k|n}(c) = P(S_0^n = s_0^n \mid C_k = c)$

(b) $L_{\nu,k} = \sum_c \alpha_{\nu,k}(c)$

(c) $L_{\nu,n} = \sum_c \alpha_{\nu,k}(c)\beta_{k|n}(c)$

Proof *(a) We already know that*

$$\phi_{k|n}(c) = \frac{1}{L_{\nu,n}} \alpha_{\nu,k}(c)\, \beta_{k|n}(c)$$

Therefore we have

$$\begin{aligned} \alpha_{\nu,k}(c)\beta_{k|n}(c) &= \phi_{k|n}(c)L_{\nu,n} \\ &= P(C_k = c \mid S_0^n = s_0^n)L_{\nu,n} \\ &= P(C_k = c, S_0^n = s_0^n) \end{aligned}$$

(b)

$$\begin{aligned} \sum_c \alpha_{\nu,k}(c) &= \sum_c P(S_0^k = s_0^k, C_k = c) \\ &= P(S_0^k = s_0^k) \\ &= L_{\nu,k} \end{aligned}$$

(c) Finally, using (a) above we can write

$$\begin{aligned} \sum_c \alpha_{\nu,k}(c)\beta_{k|n}(c) &= \sum_c P(S_0^k = s_0^k, C_k = c) \\ &= P(S_0^n = s_0^n) \\ &= L_{\nu,n} \end{aligned}$$

This last point of the proposition is the key point since it enables us to compute efficiently the likelihood function of observations thanks to the forward and backward measures providing that we can compute them. The problem now is to know how to compute these measures quite easily to complete the solution of the first issue. the next subsection exhibit the key result and make the derivation of a possible algorithm possible.

2.3.4 The so-called forward-backward algorithm

As said earlier, in the previous subsection, we need to find an efficient way to compute the forward and backward variables. Using their summation representations would be useless because as expensive as the direct computation of the likelihood function of observations $L_{\nu,n}$. In fact the following result points out a recursive way to compute the forward and backward variables, so a recursive way to compute the likelihood function. Let us state the forward-backward recursions.

Proposition 2.14 *(a) The forward variable satisfies the recursive relation:*

$$\alpha_{\nu,k}(c) = \sum_{c'} \alpha_{\nu,k-1}(c')\, p_{c',c}\, g_{c,s_k}$$

for $k = 1, \cdots, n$ with the initial condition $\alpha_{\nu,0}(c) = \nu_c\, g_{c,s_0}$.

(b) The backward variable satisfies the recursive relation:

$$\beta_{k|n}(c) = \sum_{c'} p_{c,c'}\, g_{c',s_{k+1}} \beta_{k+1|n}(c')$$

for $k = n-1, \cdots, 0$ with the initial condition $\beta_{n|n}(c) = 1$.

After this proposition we finally understand why the measures possess such names : $\alpha_{\nu,k}$ is called the forward variable because it can be computed thanks to a forward recursion (in order to compute the kth variable, we need the $k-1th$ variable) whereas $\beta_{k|n}$ is called the backward variable because it is involved in a backward recursion. The proof is quite intuitive :

Proof *(a) Using the summation representation of the forward measure, we have*

$$\begin{aligned}
\alpha_{\nu,k}(c_k) &= \sum_{c_0,\ldots,c_{k-1}} \nu_{c_0} g_{c_0,s_0} \prod_{i=1}^{k} p_{c_{i-1},c_i} g_{c_i,s_i} \\
&= \sum_{c_0,\ldots,c_{k-1}} p_{c_{k-1},c_k}\, g_{c_k,s_k}\, \nu_{c_0} g_{c_0,s_0} \prod_{i=1}^{k-1} p_{c_{i-1},c_i} g_{c_i,s_i} \\
&= \sum_{c_{k-1}} p_{c_{k-1},c_k}\, g_{c_k,s_k} \sum_{c_0,\ldots,c_{k-2}} \prod_{i=1}^{k-1} p_{c_{i-1},c_i} g_{c_i,s_i} \\
&= \sum_{c_{k-1}} p_{c_{k-1},c_k}\, g_{c_k,s_k}\, \alpha_{\nu,k-1}(c_{k-1})
\end{aligned}$$

Moreover we have

$$\begin{aligned}
\alpha_{\nu,0}(c) &= P(S_0 = s_0, C_0 = c) \\
&= P(S_0 = s_0 \mid C_0 = c) P(C_0 = c) \\
&= g_{c,s_0} \nu_c
\end{aligned}$$

(b) Using the summation representation of the backward measure, we have

$$\begin{aligned}
\beta_{k|n}(c_k) &= \sum_{c_{k+1},\ldots,c_n} \prod_{i=k+1}^{n} p_{c_{i-1},c_i} g_{c_i,s_i} \\
&= \sum_{c_{k+1},\ldots,c_n} p_{c_k,c_{k+1}}\, g_{c_{k+1},s_{k+1}} \prod_{i=k+2}^{n} p_{c_{i-1},c_i} g_{c_i,s_i} \\
&= \sum_{c_{k+1}} p_{c_k,c_{k+1}}\, g_{c_{k+1},s_{k+1}} \sum_{c_{k+2},\ldots,c_n} \prod_{i=k+2}^{n} p_{c_{i-1},c_i} g_{c_i,s_i} \\
&= \sum_{c_{k+1}} p_{c_k,c_{k+1}}\, g_{c_{k+1},s_{k+1}} \beta_{k+1|n}(c_{k+1})
\end{aligned}$$

Note 2.15 *The forward-backward algorithm is as follows:*

(1) Initialisation:

$$\alpha_{\nu,0}(c) = g_{c,s_0}\nu_c \quad for \quad all \quad c \in \mathcal{C} = 1, \cdots, r$$
$$\beta_{n|n}(c) = 1 \quad for \quad all \quad c \in \mathcal{C} = 1, \cdots, r$$

(2) Induction:

$$\alpha_{\nu,k}(c) = \sum_{c'} \alpha\nu, k-1(c')\, p_{c',c}\, g_{c,s_k}$$
$$\beta_{k|n}(c) = \sum_{c'} p_{c,c'}\, g_{c',s_{k+1}} \beta_{k+1|n}(c')$$

(3) Termination:
Either

$$L_{\nu,n} = \sum_{c} \alpha_{nu,n}(c) \quad \text{Prop} \quad 2.13$$

Or

$$L_{\nu,n} = \sum_{c} \alpha_{nu,k}(c)\beta_{k|n}(c) \quad \text{Prop} \quad 2.13$$

So we notice that we don't really need the backward variable to compute the likelihood function of observations but we will see later the backward measures are important, especially when implementing the Baum-Welch algorithm.

2.3.5 Matrix notation

As Levinson *et al.* (1983) point out, the above results can be stated more succinctly in matrix notation. Exclusively in this subsection, I will further lighten the notation and omit the dependence on ν and instead of writing $\alpha_{\nu,k}$ I will write α_k. Thus I will henceforth use β_k and L_n. If we define for all k from 0 to n, the vectors

$$\alpha_k = (\alpha_k(1), \alpha_k(2), \cdots, \alpha_k(r))$$

and similarly

$$\beta_k = (\beta_k(1), \beta_k(2), \cdots, \beta_k(r)),$$

then the result stated in Prop 2.13 (c) can be written as

$$L_n = \alpha_k \beta_k^T \quad for \quad all \quad k/$$

The recusrions for the forward and backward probabilities are given by

$$\alpha_{k+1} = \alpha_k \, B_{k+1}$$

and

$$\beta_k^T = B_{k+1}\beta_{k+1}^T$$

if we define $B_k = P\lambda(s_k)$ and $\lambda(s)$ is the r x r diagonal matrix with ith diagonal element equal to $g_{i,s}$. These recursions start from $\alpha_0 = \nu\,\lambda(s_0)$ and $\beta_n = \mathbf{1}$. The following explicit expressions for the forward and backward probabilities are therefore available:

$$\alpha_k = \nu\lambda(s_0)B_2B_3\cdots B_k$$

and

$$\beta_k^T = B_{k+1}B_{k+2}\cdots B_n\mathbf{1}^T.$$

With the convention that an empty product of matrices is the identity matrix, these expressions hold for all k from 0 to n inclusive.

2.3.6 Normalized version of the forward-Backward algorithm

This forward-backward algorithm was a big revolution in the study of Hidden Markov models and this algorithm was a way to explore the computation power of such a tool. However, a problem is still persistent and can't be detected looking at the theory. Indeed, the forward and backward measures $\alpha_{\nu,k}(c)$ and $\beta_{k|n}(c)$ as described above are largely unscaled and as n becomes important, these quantity are likely to rapidly tend to zero. The underlying numerical complication is that the computation of $L_{\nu,n}$ as described may suffer from underflow problems even for relatively small values of n : the elements $\alpha_{\nu,k}(c)$ for $c \in \mathbb{C}$ of the vector of forward probabilities α_k help at a particular stage of the algorithm may be too small to be distinguishable from zero, even if all the other parameters are of moderate size (namely even if ν_c and $g_{c,s}$ are of moderate size). Since the likelihood is additive, $Ł_{\nu,n} = \sum_{c'} \alpha_{\nu,n}(c')$, it is not possible merely to work with logarithms. What can be done, however, is described in a famous tutorial given by Rabiner in 1989 where he coined the term scaling to describe a practical solution to this problem [4]. The idea he developed is to scale the vector α_k at each stage so that the average elements is 1, i.e. to divide α_k by $\sum_{c=1}^{r} \alpha_{\nu,k}(c)$. Thus, the numeric values needed to represent $\alpha_{\nu,k}(c)$ and $\beta_{k|n}(c)$ are kept within reasonable bounds. In fact there are many variations of this technique such as : the scale factor could be chosen instead to be the largest element of the vector α_k. Here we want to replace the measure $\alpha_{\nu,k}(c)$ and $\beta_{k|n}(c)$ by their scaled versions $\overline{\alpha}_{\nu,k}(c)$ and $\overline{\beta}_{k|n}(c)$ satisfying both

$$\sum_c \overline{\alpha}_{\nu,k}(c) = 1$$

and

$$\sum_c \overline{\alpha}_{\nu,k}(c)\overline{\beta}_{k|n}(c) = 1$$

Definition 2.16 *The normalized forward measure $\overline{\alpha}_{\nu,k}(c)$ is defined by*

$$\overline{\alpha}_{\nu,k}(c) = \sum_{c'} \alpha_{\nu,k}(c')\alpha_{\nu,k}(c) = \frac{\alpha_{\nu,k}(c)}{L_{\nu,k}}$$

The normalized backward measure $\overline{\beta}_{k|n}(c)$ is defined by

$$\overline{\beta}_{k|n}(c) = \frac{L_{\nu,k}}{L_{\nu,n}}\beta_{k|n}(c)$$

Let us check that the two relations cited above are satisfied with these definitions. Indeed we have

$$\sum_c \overline{\alpha}_{\nu,k}(c) = L_{\nu,k}^{-1} \sum_c \alpha_{\nu,k}(c) = 1$$

Since we saw in Prop 2.13 that $L_{\nu,k} = \sum_c \alpha_{\nu,k}(c)$ Now concerning the second relation we have

$$\sum_c \overline{\alpha}_{\nu,k}(c)\overline{\beta}_{k|n}(c) = L_{\nu,n}^{-1} \sum_c \alpha_{\nu,k}(c)\beta_{k|n}(c) = L_{\nu,n}^{-1} L_{\nu,n} = 1$$

still using Prop 2.13. Therefore the two relations are indeed satisfied. We can also note that in terms of probability we have :

$$\begin{aligned} \overline{\alpha}_{\nu,k}(c) &= \frac{P(S_0^k = s_0^k, C_k = c)}{P(S_0^k = s_0^k)} \\ &= P(C_k = c \mid S_0^k = s_0^k) \\ &= \phi_{k|k}(c) \end{aligned}$$

Therefore the new normalized forward $\overline{\alpha}_{\nu,k}(c)$ variable is nothing but the condition probability of the kth hidden state being c given the first k observation points s_0^k. From now on we will prefer the notation $\phi_{k|k}(c)$ rather than $\overline{\alpha}_{\nu,k}(c)$ to have a clearer idea of what quantity it represents. Moreover we have the following relation from the definition of $\phi_{k|n}(c)$:

$$\begin{aligned} \phi_{k|n}(c) &= L_{\nu,n}^{-1}\alpha_{\nu,k}(c)\beta_{k|n}(c) \\ &= L_{\nu,k}^{-1}\alpha_{\nu,k}(c)L_{\nu,n}^{-1}L_{\nu,k}\beta_{k|n}(c) \\ &= \overline{\beta}_{k|n}(c)\overline{\beta}_{k|n}(c) \end{aligned}$$

Thus the expression of $\phi_{k|n}(c)$, namely the probability of the kth hidden state being c given the $n$observation points s_0^n, is simpler when expressed as a function of the new normalized forward and backward measures.

Now we must try to derive a new *Forward-Backward algorithm* from these new variables to definitely solve the problems of underflows.

Proposition 2.17 (The Normalized Forward-Backward algorithm) *(a) The new normalized variable $\phi_{k|k}(c)$ can be computed recursively according to the new recursive relation*

$$\phi_{k|k}(c) = d_{\nu,k}^{-1} \sum_{c'} \phi_{k-1|k-1}(c') p_{c'c} g_{cs_k}$$

with

$$d_{\nu,k} = \sum_c \sum_{c'} \phi_{k-1|k-1}(c') p_{c'c} g_{cs_k}$$

for k from 1 to n and the initial condition is

$$\phi_{0|0}(c) = \frac{g_{cs_0}\nu_c}{\sum_{c'} g_{c's_0}\nu_{c'}}$$

Note that $d_{\nu,k}$ is just a normalizing term and that the vector $\phi_{k|k} = \overline{\alpha}_{\nu,k}$ does sum to 1 for all k from 0 to n.

(b) The normalized backward measure satisfies the following recursive relation

$$\overline{\beta}_{k|n}(c) = d_{\nu,k+1}^{-1} \sum_{c'} p_{cc'} g_{c's_{k+1}} \overline{\beta}_{k+1|n}(c')$$

for k from $n-1$ to 0 and with initial condition $\overline{\beta}_{n|n} = 1$.

We note a strong similarity with the first version of the *Forward-Backward algorithm* since the only major difference is the presence of a normalizing term $d_{\nu,k}$ which makes sure that the forward vector $\phi_{k|k}(c)$ sums up to 1 and hence we can't obtain an underflow problem. The next part of this subsection is entirely reserved to the proof of this proposition which is probably the heavier proof in this whole project.

Proof *This proof will be organized as follows :*

- *We show that* $d_{\nu,k} = \frac{L_{\nu,k}}{L_{\nu,k-1}} = \frac{P(S_0^k = s_0^k)}{P(S_0^{k-1} = s_0^{k-1})}$
- *Secondly, we prove the recursive relation for the normalized forward variable using the recursive relation of the unnormalized one.*
- *Thirdly, we prove the recursive relation for the normalized backward variable using the recursive relation of the unnormalized one.*

First given the formula of $d_{\nu,k}$ *in the last Proposition, we have*

$$\begin{aligned}
d_{\nu,k} &= \sum_c \sum_{c'} \phi_{k-1|k-1}(c') p_{c'c} g_{cs_k} \\
&= \sum_c \sum_{c'} P(C_{k-1} = c' \mid S_0^{k-1} = s_0^{k-1}) p_{c'c} g_{cs_k} \\
&= \frac{1}{P(S_0^{k-1} = s_0^{k-1})} \sum_c \sum_{c'} P(C_{k-1} = c', S_0^{k-1} = s_0^{k-1}) p_{c'c} g_{cs_k} \\
&= L_{\nu,k-1}^{-1} \sum_c \sum_{c'} P(C_{k-1} = c', S_0^{k-1} = s_0^{k-1}) \\
&\times \quad P(C_k = c, S_k = s_k \mid C_{k-1} = c') \text{ By def 2.4} \\
&= L_{\nu,k-1}^{-1} \sum_c \sum_{c'} P(C_{k-1} = c', S_0^{k-1} = s_0^{k-1}) \\
&\times \quad P(C_k = c, S_k = s_k \mid C_{k-1} = c', S_0^{k-1} = s_0^{k-1}) \\
&= L_{\nu,k-1}^{-1} \sum_c \sum_{c'} P(C_{k-1} = c', S_0^{k-1} = s_0^{k-1}, C_k = c) \\
&= L_{\nu,k-1}^{-1} P(S_0^k = s_0^k) \\
&= \frac{L_{\nu,k}}{L_{\nu,k-1}}
\end{aligned}$$

as required.
Secondly, the right-hand side of the normalized forward recursion is

$$\begin{aligned}
d_{\nu,k}^{-1} \sum_{c'} \phi_{k-1|k-1}(c') p_{c'c} g_{cs_k} &= \frac{L_{\nu,k-1}}{L_{\nu,k}} \sum_{c'} \frac{\alpha_{\nu,k-1}(c')}{L_{\nu,k}} p_{c'c} g_{cs_k} \\
&= \frac{\alpha_{\nu,k}(c)}{L_{\nu,k}} \text{ By recursive relation of the unnormalized forward variable} \\
&= \phi_{k|k}(c)
\end{aligned}$$

Thirdly, the right-hand side of the normalized backward recursion is

$$\begin{aligned}
\frac{L_{\nu,k}}{L_{\nu,k+1}} \sum_{c'} p_{cc'} g_{c's_{k+1}} \overline{\beta}_{k+1|n}(c') &= \frac{L_{\nu,k}}{L_{\nu,k+1}} \sum_{c'} p_{cc'} g_{c's_{k+1}} \frac{L_{\nu,k+1}}{L_{\nu,n}} \beta_{k+1|n}(c') \\
&= \frac{L_{\nu,k}}{L_{\nu,n}} \sum_{c'} p_{cc'} g_{c's_{k+1}} \beta_{k+1|n}(c') \\
&= \frac{L_{\nu,k}}{L_{\nu,n}} \beta_{k|n}(c') \\
&= \overline{\beta}_{k|n}(c)
\end{aligned}$$

Finally concerning the initial conditions

$$\begin{aligned}
\phi_{0|0}(c) &= P(C_0 = c \mid S_0 = s_0) \\
&= \frac{P(C_0 = c, S_0 = s_0)}{P(S_0 = s_0)} \\
&= \frac{P(C_0 = c, S_0 = s_0)}{\sum_{c'} P(C_0 = c', S_0 = s_0)} \\
&= \frac{g_{cs_0}\nu_c}{\sum_{c'} g_{c's_0}\nu_{c'}}
\end{aligned}$$

and

$$\overline{\beta}_{n|n}(c) = \frac{L_{\nu,n}}{L_{\nu,n}} \beta_{n|n}(c) = 1$$

Now that we have derived a new and more efficient version of the forward-backward algorithm (avoiding underflows), we must find the new expression of the likelihood function of observations $L_{\nu,k}$ for k from 0 to n. There are several ways to do so but one tricky thing one might notice is given the expression of $d_{\nu,k}$ we have

$$\begin{aligned}
d_{\nu,k} = \frac{L_{\nu,k}}{L_{\nu,k-1}} &= \frac{P(S_0^k = s_0^k)}{P(S_0^{k-1} = s_0^{k-1})} \\
&= P(S_0^k = s_0^k \mid S_0^{k-1} = s_0^{k-1}) \\
&= P(S_k = s_k \mid S_0^{k-1} = s_0^{k-1})
\end{aligned}$$

Therefore we can see that $L_{\nu,k} = \prod_{i=0}^{k} d_{\nu,i}$ and we can compute the likelihood function of the first kth observations recursively using the previous normalized forward-backward algorithm. However, to be cautious, it is better to evaluate the log likelihood defined by $\log\ L_{\nu,k} = \sum_{i=0}^{k} \log\ d_{\nu,i}$.

2.3.7 The Baum-Welch Algorithm

The Baum-Welch algorithm represents the central part of my project. Historically speaking, The Baum-Welch algorithm was developed by L.E Baum and his co-workers in a series of papers published between 1966 and 1972: Baum and Petrie (1966), Baum and Eagon (1972), Baum ans Sell (1968), Baum *et al.* (1970), and Baum (1972). The name of Welch seems to appear only as joint author (with Baum) of a paper listed by Baum *et al.* (1970) as submitted for publication. Since the beginning of this paper, in the previous subsections, we have supposed that all the parameters of the model, often referred to as λ in the literature where λ represents the conjunction of the initial distribution ν of the underlying Markov chain $\{C_t\}_{t\in\mathbb{N}}$, the transition matrix of this same Markov chain, and the conditional

probability matrix G supplying the distribution of observations given the hidden state, were fully known. In fact, in many real-life processes we can't fully specify the model parameters and most of them have to be estimated from the observed data. We have already seen that the likelihood function can be efficiently computed thanks to the forward-backward algorithm. As far as the Baum-Welch algorithm is concerned, it computes a maximum likelihood estimates of the parameters and for that reason it is often called 'The Baum-Welch re-estimation algorithm'. The algorithm is in fact an early example of an algorithm of Expectation-Maximisation (EM) algorithm. However in the literature, and in particular the expository article of Juang and Rabiner (1991) gives only a brief account on the relation between the Baum-Welch and EM algorithms, and it therefore seems useful to discuss that relation later in the project. Indeed, I will delay the theory of the Baum-Welch algorithm until section 4 since I want to make a parallel between the discrete case and the continuous-time case. For now, I only give the results for the re-estimates of the concerned parameters:

Proposition 2.18 (Baum-Welch) *The optimal (with respect to EM) re-estimates for the HMM parameters during 1 iterative step in the Baum-Welch algorithm are*

$$\begin{aligned}
\hat{\nu}_j &= \frac{\alpha_{\nu,0}(j)\beta_{0|n}(j)}{\sum_l \alpha_{\nu,0}(l)\beta_{0|n}(l)} \\
\hat{g}_{js} &= \frac{\sum_{i=0}^{n} \delta_{s_i,s}\alpha_{\nu,i}(j)\beta_{i|n}(j)}{\sum_{i=0}^{n} \alpha_{\nu,i}(j)\beta_{i|n}(j)} \\
\hat{p}_{jk} &= \frac{p_{jk}\sum_{i=0}^{n-1} \alpha_{\nu,i}(j)g_{k,s_{i+1}}\beta_{i+1|n}(k)}{\sum_{i=0}^{n-1} \alpha_{\nu,i}(j)\beta_{i|n}(j)}
\end{aligned}$$

where $\delta_{i,j}$ is the Kronecker symbol taking the value 1 if $i = j$ and 0 otherwise. Note that we are using the forward and backward variables and so the Baum-Welch algorithm must necessarily be implemented along with the Forward-Backward algorithm. In fact this is an iterative process where the forward and backward variables are evaluated with the old values of the parameters ν,P and G. Once the new values of the parameters are calculated thanks to the above set of formulas, we re-compute the forward and backward variables with the updated values of parameters and so on. It can be shown that this process necessarily converges.

2.3.8 The Viterbi Algorithm

There is one remaining problem that we haven't treated so far which is the one concerning the most likely sequence of hidden states that could have generated a given (fully known) sequence of observations. In the following we assume that the sequence of observations is fixed. In speech recognition and other applications, it is of interest to determine the states of the Markov chain that are most likely (under the fitted model which is also assumed to be specified) to have given rise to the observation sequence. In the context of speech recognition this is known as the 'decoding' problem: see Juang and Rabiner (1991). More specifically, 'localized decoding' of the state at time k refers to the determination of the state $\overline{c}_k$ which is (*a posteriori*) most likely, that is :

$$\overline{c}_k = \arg \max_{0 \le c \le r} P(C_k = c \mid S_0^n = s_0^n)$$

for all k from 0 to n.
In contrast, 'global decoding' refers to the determination of the jointly optimal sequence of

states $\hat{c}_0, ..., \hat{c}_n$ which maximizes the conditional probability :

$$P(C_0^n = c_0^n \mid S_0^n = s_0^n)$$

In general the sequences $\overline{c}_0^n$ and $\hat{c}_0^n$ are different. This subsection will deal with the second case, that is global decoding (determination of the sequence $\hat{c}_0^n$) which can be carried out by using a dynamic programming method known as the Viterbi algorithm. Here the quantity of interest is therefore $\phi_{0:k|k}$ (up to time k for $k = 0, ..., n$) and it could be useful to derive a recursive relation as follows

$$\begin{aligned}
\phi_{0:k+1|k+1}(c_0, ..., c_{k+1}) &= P(C_0^{k+1} = c_0^{k+1} \mid S_0^{k+1} = s_0^{k+1}) \\
&= \frac{P(C_0^{k+1} = c_0^{k+1}, S_0^{k+1} = s_0^{k+1})}{P(S_0^{k+1} = s_0^{k+1})} \\
&= \frac{1}{L_{\nu,k+1}} \nu_{c_0} g_{c_0,s_0} \prod_{i=1}^{k+1} p_{c_{i-1},c_i} g_{c_i,s_i} \\
&= \frac{L_{\nu,k}}{L_{\nu,k+1}} \frac{1}{L_{\nu,k}} \nu_{c_0} g_{c_0,s_0} \prod_{i=1}^{k} p_{c_{i-1},c_i} g_{c_i,s_i}\ p_{c_k,c_{k+1}} g_{c_{k+1},s_{k+1}} \\
&= \frac{L_{\nu,k}}{L_{\nu,k+1}} \phi_{0:k|k}(c_0, ..., c_k)\, p_{c_k,c_{k+1}} g_{c_{k+1},s_{k+1}}
\end{aligned}$$

Now that we have derived such recursive relation we are going to think in terms of log-likelihood. In many other books, the approach is undertaken directly on the likelihood. If denote $l_k = L_{\nu,k}$ and take the logarithm of the previous equation we have

$$\log(\phi_{0:k+1|k+1}(c_0, ..., c_{k+1})) + l_{k+1} = \log(\phi_{0:k|k}(c_0, ..., c_k)) \ + \ l_k \ + \ \log(p_{c_k,c_{k+1}}) \ + \ \log(g_{c_{k+1},s_{k+1}})$$

We then define a couple of terms $(m_k(i), b_k(i))$:

$$m_k(i) = \max_{c_0,...,c_{k-1}} \log(\phi_{0:k|k}(c_0, ..., c_{k-1}, i)) \ + \ l_k$$

which is the maximum a posteriori log probability up to time k and ending with state i. Note that maximizing the log probability is equivalent to maximizing the probability since the log is an increasing function. And we define $b_k(i)$ as the value of c_{k-1} for which the maximum is achieved in the definition of$m_k(i)$. Thus $b_k(i)$ is the second state of the optimal state sequence up to time k and ending with state i. Now it is easy to see that thanks to the previous recursive relation we have

$$m_{k+1}(j) = \max_i (m_k(i) + \log(p_{ij})) + \log(g_{j,s_{k+1}})$$

and in this context it naturally comes that $b_{k+1}(j)$ is the i for which the maximum is achieved in the above formula.

We can also remark that

$$\begin{aligned}
m_0(i) &= \log(\phi_{0:0|0}(i) L_{\nu,0}) \\
&= \log(P(C_0 = i \mid S_0 = s_0) P(S_0 = s_0)) \\
&= \log((P(C_0 = i, S_0 = s_0)) \\
&= \log(\nu_i g_{i,s_0})
\end{aligned}$$

At this point we have all the elements to extract the optimal hidden state sequence. Indeed, by starting the process with $m_0(i) = log(\nu_i g_{i,s_0})$, we can obtain the final term $m_n(j)$ using

the formula $m_k(i) = \max_{c_0,...,c_{k-1}} \log(\phi_{0:k|k}(c_0, ..., c_{k-1}, i)) + l_k$ for $k = 1, ..., n-1$. So if we maximize $m_n(j)$, we obtain the final state $\hat{c}_n$ of the jointly optimal sequence of states. In fact $\hat{c}_n$ is just the j for which $m_n(j)$ is maximum. Now in order to obtain the rest of the optimal sequence we just have to run the backward recursion

$$\hat{c}_k = b_{k+1}(\hat{c}_{k+1})$$

for k=n-1,...,0 because we know that $\hat{c}_k$ is the value of i for which the maximum of $m_{k+1}(\hat{c}_{k+1})$ is attained. We can summarize all of this in the Viterbi algorithm.

Proposition 2.19 (Viterbi Algorithm) *This algorithm is therefore split into two different steps:*

- *Forward recursion to find the last optimal state $\hat{c}_n$:*

  $$m_0(i) = log(\nu_i g_{i,s_0})$$

  For $k = 0, ..., n-1$, compute $m_{k+1}(j)$ for all states j using the equation

  $$m_{k+1}(j) = \max_i(m_k(i) + \log(p_{ij})) + \log(g_{j,s_{k+1}})$$

  Finally $\hat{c}_n = \arg\max_j m_n(j)$

- *Backward recursion to find the rest of the optimal sequence:*
  Once we have determined $\hat{c}_n$, we can determine $\hat{c}_{n-1}$ using the fact that $\hat{c}_{n-1}$ is just the $n-1$th state for which $m_n(\hat{c}_n)$ is maximum.

  $$\hat{c}_{n-1} = \arg\max_i(m_k(i) + \log(p_{i\hat{c}_n})) + \log(g_{\hat{c}_n,s_{k+1}})$$

  or equivalently

  $$\hat{c}_{n-1} = b_n(\hat{c}_n)$$

  So for $k = n-1, ..., 0$, $\hat{c}_k$ is the state i for which the maximum is attained in the expression $m_{k+1}(\hat{c}_{k+1})$, or equivalently, $\hat{c}_k = b_{k+1}(\hat{c}_{k+1})$.

3 An Application of HMMs to Flash Memory

This project was initiated by my supervisor Peter Harrison and is in fact a logical suite, as outlined in the introduction, of a previous project "Storage Worload Modelling by Hidden Markov Models: Application to FLASH Memory" [1] he worked in with S.K. Leyton, N.M. Patel and S. Zertal. In this latter paper, they used HMMs to find a representative model to characterize the stream of read and write instructions arriving at Flash memory chip. After obtaining industrial benchmark traces, they presented a technique that processes data from operation type traces and creates a Hidden Markov model to represent efficiently the workload that generated those traces. They finally tested their HMM by comparing its autocorrelation function with the one of the data trace. Once a HMM is found it is used to generate new representative traces which is priceless in many areas because as we know, obtaining a complete trace from industries is far from being an easy and quick task, and since HMMs only need a small number of parameters, they constitute a compact way to create new faithful traces very easily. We shall describe the outlines of this previous project and see how my project came into existence.

3.1 Summary of the work

3.1.1 Motivations

Currently, we are witnessing an increasing number of networked or/and on-line storage spaces. Many Internet services are specifically dedicated to supply a virtualized storage space, and one can easily understand that such services need to treat traffic from many servers without affecting the storage performance to all applications. In fact there may be some bottlenecks in the storage systems and the workload models need to take it into account as well as the time-varying correlated traffic streams. More and more storage systems present a large layer of Flash in the memory hierarchy between DRAM and Hard Disk Drives (HDD). That is why one should focus on studying the workload models for Flash memory.

We all know the great importance of IO traces for workload modelling. However, obtaining an IO trace from a production system is often a difficult procedure as traces are very long (many Gigabytes) and therefore time-consuming. This emphasizes the need to find an alternative method to the direct use of IO traces. Instead, we could think of a compact way of representing traces thanks to a small number of key parameters determined on a first initial trace. Then things become far easier as providing a trace would be equivalent to providing all the characteristic parameters of this trace. In fact it would be more correct to have a set of parameters for a class of traces because we may want to deal with a trace with particular characteristics (for instance a write-dominated trace) and thus with particular values of parameters.

3.1.2 Flash memories

In the current market, two types of hard disk drives are in competition: high speed storage (SAS HDD) and high capacity storage (SATA HDD). Furthermore, a new technology has recently appeared and threatens the domination of HDDs: this is solid-state drive (SSD) technology also referred to as Flash-based storage. In fact, one crucial factor to assess the performance of a HDD-based system is to evaluate the disparity between random and sequential IO requests. Indeed, for a SATA drive, it can treat 100 MB per second for sequential reads whereas it can only do 0.5 MB per second for random reads. We say that

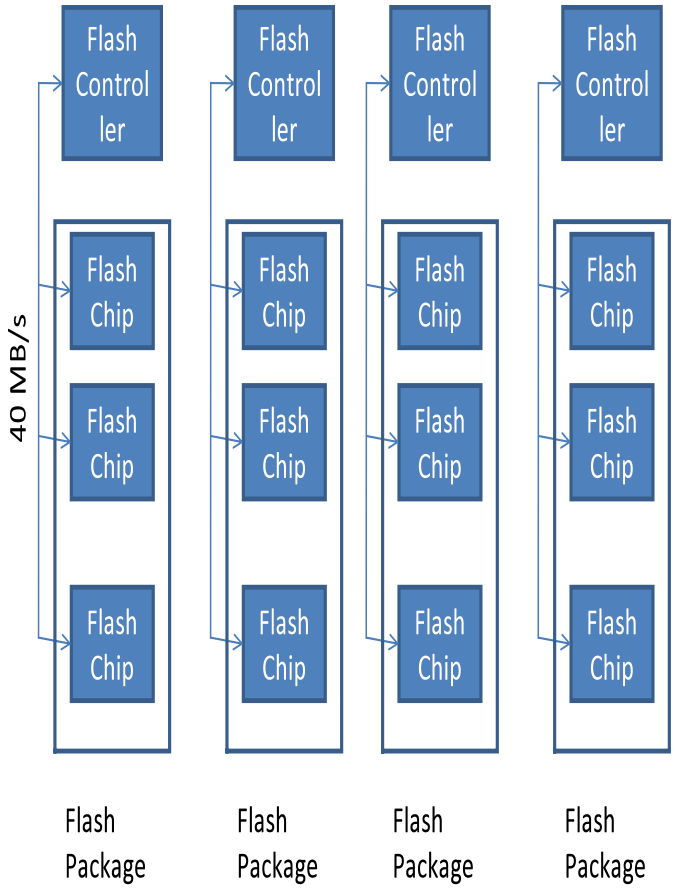

Figure 2: Vector of Flash packages with 40 MB/s channels

there is a huge order of disparity for read requests. Concerning write requests, we can solve the problem using a Write anywhere file system such as WAFL (described in figures 2 and 3) since writes would be located in pre-selected regions of the disk.

With Flash-based storage, the sequentiality or randomness of IO requests is not a big issue although we can note that random writes influence the performance of the system, and so the CPU resources required are better matched to the number of Flash devices. This allow us to claim that Flash is better suited to the mixture of workloads where different modes over time arise. Concerning Flash, three main modes are of interest: reads only, writes only and mixed reads/writes. It can be shown experimentally, that the performance of a Flash-based system is lower when there is a mixture of reads and writes since read and write rates vary over time.

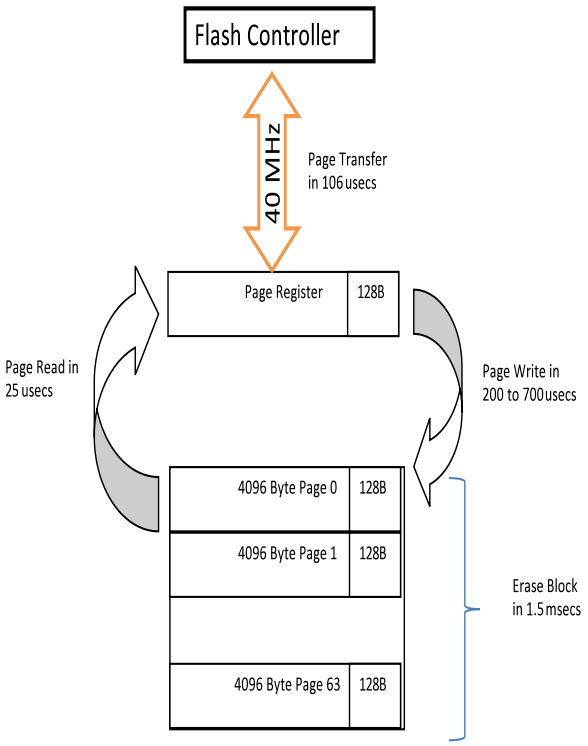

Figure 3: Flash chip and controller

3.1.3 Construction of the models

One of their main objective was to construct a portable model in terms of Markov modulated Poisson Processes (see section 3.3) that can faithfully represent correlation and burstiness in multi-application workloads. Therefore, the underlying workload dynamics is assumed to be a Markov chain and it is important to derive the key characteristics of this Markov chain, namely its initial distribution and its one-step transition matrix. HMMs are particularly suitable when we know that the behaviour of a time series is partly influenced by a sequence of switching modes (observable or not): these modes will constitute the hidden states of the HMM and follow the Markovian conditions as described in section 2.

There are several ways to collect traces and it can be done at different stages or layers in the Flash storage controller stack (figure 4). Along this stack, workloads go through different layers, each of which proceeds to a transformation that potentially amplifies the number of IO requests. For instance, in the Flash stack, the workload goes through a Flash Translation Layer (FTL) that introduces additional traffic depending on the specific Flash design choices implementation. They chose to collect the trace just after storage virtualization and before the FTL, because this was the lowest level they could practically collect detailed traces.

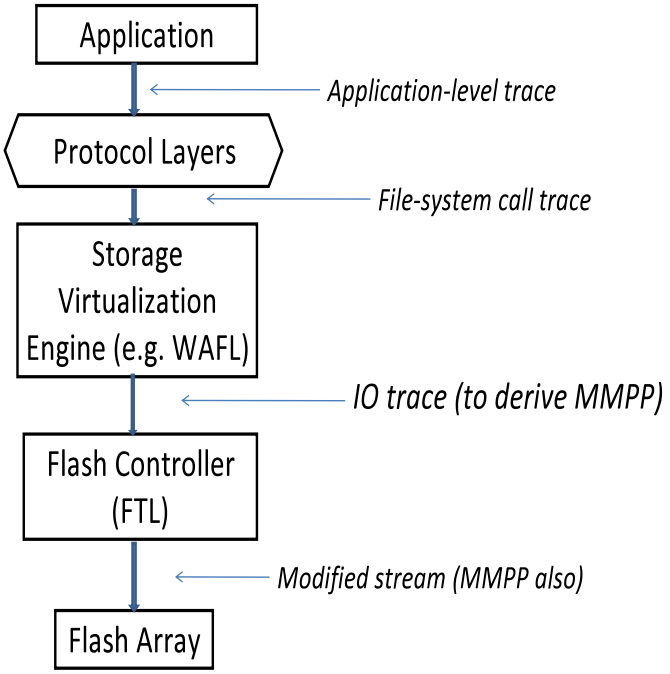

Figure 4: Flash chip and controller

From this trace, they created a "binned trace" constituted of a sequence of 5 msec bins representing the number of reads and writes. They further reduced the size of the trace by applying a clustered algorithm: they used two values for the number of reads (low number of reads and high number of reads) and 4 values for the number of writes. This implies that the observation state space is composed of 8 values that are given below

$$\begin{pmatrix} 5.7 & 0.28 \\ 4.45 & 31.3 \\ 5.11 & 82.1 \\ 4.49 & 183.0 \\ 23.7 & 0.217 \\ 24.5 & 31.7 \\ 27.3 & 81.0 \\ 25.8 & 165.8 \end{pmatrix}$$

Each row is a cluster representative and as we can see the observation values of 1 to 4 represent low reads and increasing writes whilst observation values 5 to 8 represent high reads and increasing writes. Now from this new processed trace, they applied the Baum-Welch algorithm and they found out the following parameter values (to 4 decimal places):

$$\text{Transition matrix} \quad P = \begin{pmatrix} 0.9972 & 0.0022 & 0.0006 \\ 0.0005 & 0.9965 & 0.0030 \\ 0.021 & 0.0005 & 0.9974 \end{pmatrix}$$

$$\text{Conditional probability matrix} \quad G = \begin{pmatrix} 0.634 & 0.0 & 0.0 & 0.0 & 0.366 & 0.0 & 0.0 & 0.0 \\ 0.076 & 0.118 & 0.237 & 0.112 & 0.068 & 0.115 & 0.199 & 0.076 \\ 0.241 & 0.393 & 0.0 & 0.158 & 0.205 & 0.0 & 0.0 & \end{pmatrix}$$

3.1.4 Validation of the HMM

As usual, in order to test the faithfulness of a HMM, they decided to generate a Monte Carlo simulation of the HMM using the parameters previously estimated. They also had to choose some key metrics to evaluate and compare the performance of the HMM: here they chose the numbers of reads and writes in each bin. During the performance evaluation, they considered three traces which are the original binned trace, the clustered trace and the HMM-generated trace. When they compared the three traces, they found out that the three traces were very similar with respect to the key metrics, even though the clustering algorithm involved a loss of information that was measurable thanks to the HMM-generated trace.

3.1.5 Simple measures

As mentioned in the previous section, using the key metrics, which are the mean count and its standard deviation per bin, the derived Hidden Markov model has been validated and a summary of this metrics comparison is given in this following table:

Reads/bin				Writes/bin			
Raw		HMM		Raw		HMM	
Mean	Std Dev	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
12.71	12.15	12.84	12.04	27.98	44.82	28.44	45.51

We can remark excellent agreement is obtained for both the mean values and more impressively for the standard deviations.

3.1.6 Comparison of the autocorrelation functions

An important characteristic of any time series model is its serial dependence structure. In the case of the usual normal-theory models this is specified by the autocorrelation function (ACF). While that is not true in general of non-normal models, the autocorrelation function can still be a useful tool in the case of discrete-valued models and only if the observations are quantitative (the observations space must be a number space) and not merely categorical.

The graphs in figures 5 and 6 show the two autocorrelation functions of the raw data trace and HMM-generated trace respectively for writes only. As we can see the global shape of the HMM ACF is very similar to the raw trace's. The HMM ACF is more noisy that the raw trace's one which is a bit weird as usually built models tend to erase noise. We can see that both ACFs start at about 0.5, then they decrease until becoming negative at larger lags. However, the behaviour for very large lags is completely different since the HMM ACF stays in the negative domain for a very short interval and then switches back to the positive domain which is not the case for the raw trace ACF. This is the reason why my supervisor Peter Harrison wondered if it was not worth starting thinking about a continuous time model rather than binned traces.

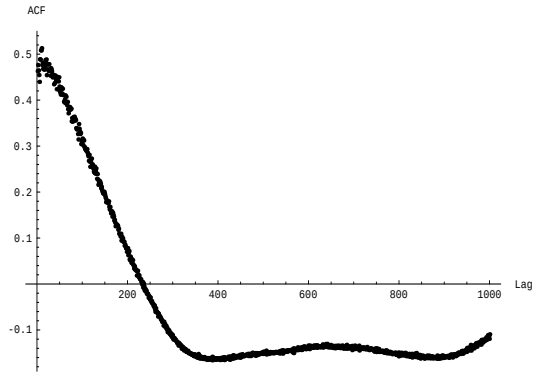

Figure 5: ACF for raw writes

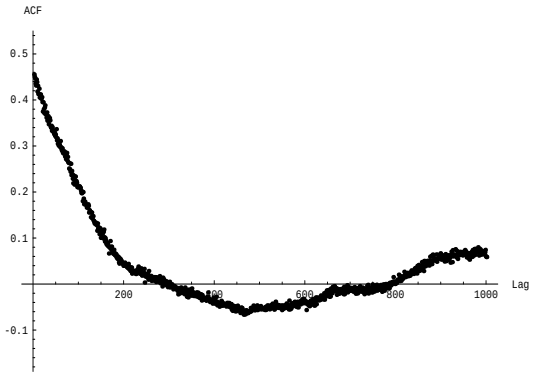

Figure 6: ACF for HMM writes

3.1.7 Towards a continuous-time version

The performances shown in figures 5 and 6 underline the correctness of the derived HMM while at the same time showing us the limit of this discrete model. Instead of using 'binned traces' along with the clustering algorithm, one should then try to consider the sequences of hidden states and the observation points as a doubly stochastic process $\{(C(t), S(t)), t \geq 0\}$ such that $\{C(t), t \geq 0\}$ is a continuous-time Markov chain which is explained in the following section 3.2 and $\{S(t), t \geq 0\}$ a simple stochastic process. So the three main hypothesis that we are establishing for this continuous-time study are: $\{C(t), t \geq 0\}$ is a continuous-time Markov chain, $\{(C(t), S(t)), t \geq 0\}$ is a MMPP which we shall see in section 3.3, and the doubly stochastic process $\{(C(t), S(t)), t \geq 0\}$ embedded at arrival times is a HMM.

3.2 Continuous-time Markov Chain

Adopting a continuous-time approach always brings a part of complexity. However, all the concepts we introduced in the discrete-case have an analogy with continuous time. In continuous time, the stochastic process is observed at arbitrary instants whereas in discrete case, the process is supposed to be observed only at particular moments.

To make everything clear, here is a concrete example distinguishing the discrete case from the continuous case. Assume we observe the number of packets present in a router and we denote by $X(t)$ the number of packets in the router at time t. We can either make these observations at integer instants $t = 0, 1, \cdots$ and in this case $\{X(t)\}_{t=0,1,\cdots}$ is a discrete time stochastic process, or we can make these observations at times $T_1 < T_2 < \cdots < T_n < \cdots$. These instants $(T_n)_n$ can be deterministic (for instance $T_n = n$ and this is equivalent to the previous example) or random. For example, T_n can be the time at which the nth packet leave the router (which is a random time in general) et therefore $X(T_n)$ represents the number of packets in the router just after the nth packet has gone. In this case, the process $\{X(T_n)\}_{n=0,1,\cdots}$ is still a discrete time stochastic process. On the other hand, if the observations are made at arbitrary instants, then $\{X(t), t \geq 0\}$ is a continuous-time stochastic process.

3.2.1 Defintion

Definition 3.1 (Continuous Time Markov Chain) *A stochastic process $\{C(t), t \geq 0\}$ with state space $\mathcal{C}$ (assumed to be finite or countable) is said to be a homogeneous continuous time Markov chain (CTMC), or Markov process if and only if:*

- **Markov Property (MP):** *for all $n \in \mathbb{N}$, all n-tuple of real numbers $t_0 < t_1 < \cdots < t_n < t_{n+1}$ and all $n+2$-tuple $j_0, j_1, \cdots, j_n, j_{n+1}$ elements of $\mathcal{C}$,*

$$P[X(t_{n+1}) = j_{n+1} \mid X(t_n) = j_n, \cdots, X(t_0) = j_0] = P[X(t_{n+1}) = j_{n+1} \mid X(t_n) = j_n];$$

- **Homogeneity:** *for all real numbers s, t and u and all couple (i, j) of $\mathcal{C}$,*

$$P[X(t+u) = j \mid X(s+u) = i] = P[X(t) = j \mid X(s) = i] = P_{t-s}(i, j),$$

independently on t.

One of the most famous and important results about CTMC is the so-called *Chapman-Kolmogoroff equation* which states that:
For all $t, u, s \in \mathbb{R}$, and for all state couple (i, j) we have

$$P_{t+s}(i, j) = \sum_{k \in \mathcal{C}} P_t(i, k) P_s(k, j)$$

or put into matrix form,

$$\mathbf{P_{t+s}} = \mathbf{P_t P_s}$$

where $\mathbf{P_u}$ is the matrix where the entry (i, j) is $P_u(i, j)$. This result is nothing but the mathematical expression of the following statement :
'In order to get from state i at time 0 to state j at time $t + s$, the process must be in some state k at time t'.

Proof (Chapman-Kolmogoroff equation) *Using the Markov property and the homogeneity property*

$$\begin{aligned}
P_{t+s}(i, j) &= P[X(t+s) = j \mid X(0) = i] \\
&= \sum_{k \in \mathcal{C}} P[X(t+s) = j, X(t) = k \mid X(0) = i] \\
&= \sum_{k \in \mathcal{C}} P[X(t+s) = j \mid X(0) = i, X(t) = k] P[X(t) = k \mid X(0) = i] \\
&= \sum_{k \in \mathcal{C}} P[X(t+s) = j \mid X(t) = k] P[X(t) = k \mid X(0) = i] \\
&= \sum_{k \in \mathcal{C}} P_s(k, j) P_t(i, k)
\end{aligned}$$

3.2.2 The infinitesimal generator matrix Q

The major issue we are confronted to when dealing with continuous-time Markov chains is that there are no smallest time steps and we can't use transition matrices (as defined in Section 2) any more. One quantity that might be of interest is the probability of the process to be in state j at time t+h given that the process was in state i at time t, for relatively small values of h and different states i and j,

$$P[X(t+h) = j \mid X(t) = i].$$

Let g be the function such that: $g(h) = P[X(t+h) = j \mid X(t) = i]$. Then clearly $g(0) = 0$ since the states are different. Now assuming that g is differentiable at 0, and since we are

dealing with small values of h, one important information we could obtain is the first-order derivative of g

$$g'(0) = \lim_{h\to 0} \frac{g(h) - g(0)}{h} = \lim_{h\to 0} \frac{P[X(t+h) = j \mid X(t) = i]}{h} = q_{ij}.$$

We also define $q_{ii} = -\sum_{k\in\mathcal{C}, k\neq i} qik$ and we can create the matrix $Q = (q_{ij})_{i,j\in\mathcal{C}}$ which is referred to as *the infinitesimal generator matrix* of the Markov process or *the Q-matrix* of the Markov process. Note that this matrix possesses the following properties

- all off-diagonal elements $q_{ij}, i \neq j$, are non-negative,
- all diagonal entries q_{ii} are non-positive and,
- all rows sum to 0.

It can be shown that under certain regularity conditions, we have

$$\lim_{t\to 0} \mathbf{P_t} = \mathbf{I}$$

and the Q-matrix satisfies

$$Q = \lim_{t\to 0} \frac{\mathbf{P_t} - \mathbf{I}}{t}$$

There exists a deep link between the Q generator matrix and the temporal evolution of the Markov process: when the process enters state i, it remains in state i according to an exponential distribution of parameter $-q_{ii} > 0$ and then it instantly jumps to another state $j \neq i$ with probability $-q_{ij}/q_{ii}$. Therefore, a CTMC is fully characterized by its initial distribution $\nu = (\{P[X(0) = i]\}_{i\in\mathcal{C}})$ and the Q-matrix (note that we had a similar result for the discrete case with ν and the transition matrix P). The equations justifying this result are giving right below:

$$\mathbf{P_t} = e^{t\mathbf{Q}} = \sum_{n=0}^{\infty} \frac{t^n}{n!}\mathbf{Q}^n.$$

Moreover, if we denote the distribution of the process at any time t by $\mathbf{p_t}$ where

$$\mathbf{p_t} = (\{P[X(t) = k]\}_{k\in\mathcal{C}}) = (P[X(t) = 1],\ P[X(t) = 2],\ , \cdots)$$

then we have

$$\mathbf{p_t} = \nu\mathbf{P_t} = \nu e^{t\mathbf{Q}}$$

Proof *First we note that the matrix* $\mathbf{P_t}$ *satisfies, for all* $t, s \in \mathbb{R}$*:*

$$\begin{cases} \mathbf{P_{t+s}} = \mathbf{P_t}\mathbf{P_s} \\ \mathbf{P_0} = \mathbf{I} \end{cases}$$

Therefore we know that there exists a matrix X *such that:*

$$\mathbf{P_t} = \mathbf{P_0}e^{tX}.$$

Using the fact that $\mathbf{P_0} = \mathbf{I}$*, we can further express* $\mathbf{P_t} = e^{tX}$ *and if we now take the first derivative we have*

$$\mathbf{P'_t} = Xe^{tX}$$

but for $t = 0$ *we find* $\mathbf{P'_0} = X$ *which is also known to be* $Q = \lim_{t\to 0} \frac{\mathbf{P_t}-\mathbf{I}}{t}$.
Then, to prove the second result we just have to note that

$$P[X(t) = i] = \sum_{k\in\mathcal{C}} P[X(t) = i \mid X(0) = k]P[X(0) = k]$$

As in discrete time, it is possible to compute the probabilities for all t, but this will require the computation of e^{tQ}. This is very expensive in general (especially when Q is very large). In fact in general we are interested in stationary probabilities that we shall define in the next subsection.

3.2.3 Stationary probabilities

The stationary probabilities represent the distribution of the Markov process for $t \to \infty$. Therefore in vector notation, we want to find the vector $\mathbf{p} = \lim_{t\to\infty} \mathbf{p_t}$ regardless of what the initial state is. Let $\epsilon \in \mathbb{R}_+^*$, we can write

$$\begin{aligned} \mathbf{p}_t &= \mathbf{p}_{t-\epsilon} e^{\epsilon Q} \\ &= \mathbf{p}_{t-\epsilon}(I + \epsilon Q + \frac{\epsilon^2}{2} Q^2 + \cdots). \end{aligned}$$

Now by making $t \to \infty$, we see that

$$\mathbf{p} = \mathbf{p}(I + \epsilon Q + \frac{\epsilon^2}{2} Q^2 + \cdots)$$

which leads to

$$0 = \mathbf{p}(\epsilon Q + \frac{\epsilon^2}{2} Q^2 + \cdots).$$

Finally if we divide by ϵ and make $\epsilon \to 0$, we obtain

$$\mathbf{p}\, Q = 0.$$

We have just shown that if the stationary state or steady-state vector $\mathbf{p}$ exists then it satisfies $\mathbf{p}\,\mathrm{Q} = 0$. The following result supplies the conditions under which the stationary state of a CTMC exists, regardless of the initial state.

Proposition 3.2 (Stationary probabilities) *If the system of equations*

$$\begin{cases} \boldsymbol{\pi} Q = 0 \\ \boldsymbol{\pi} \cdot \mathbf{1} = 1 \end{cases}$$

has a strictly positive solution (all elements in $\boldsymbol{\pi}$ are strictly positive) then the stationary distribution $\mathbf{p}$ exists, is independent of the initial distribution and is given by

$$\mathbf{p} = \boldsymbol{\pi}.$$

3.3 Markov modulated Poisson Process

Since an important assumption made by my supervisor's group was to consider the bivariate stochastic process as a Markov-modulated Poisson process (MMPP), one should introduce very briefly what are MMPPs. As said before, the easiest model we could consider to describe an arrival stochastic process is to consider the process as a homogeneous Poisson process (each inter-arrival time is a random variable following an exponential law of a given parameter called 'rate' and all these random variables are independent and identically distributed), however, in many applications, especially in communications modeling, it appears that this kind of model is not desirable because not representative enough. In fact a better model would be to allow these rates to vary randomly over time. Indeed, by adopting this new approach, we qualitatively model the time-varying arrival rate and capture some of

the important correlations between the inter-arrival times while still remaining tractable. The Markov-modulated Poisson processes are specifically designed for this purpose and have been extensively used in the area of computer networks. The MMPP is a special case of Markovian Arrival Process (MAP). We are going to give the definition of an MMPP and try to extract an interpretation.

3.3.1 Definition

The Markov-modulated Poisson process (MMPP) is a doubly stochastic process whose current arrival rate is determined or modulated by a continuous-time Markov chain. Thus the arrival rate at time t is given by $\lambda \cdot C(t)$, where $\{C(t), t \geq 0\}$ is a CTMC with state-space $\mathcal{C} = \{1, 2, ..., r\}$. An equivalent definition would be to say that a MMPP is just a Poisson process whose rate is modulated over time by a r-state Markov process. To give an illustrative idea, when the Markov chain is in state i, arrivals occur according to a Poisson process of rate λ_i. Therefore, an MMPP is fully characterized by the infinitesimal generator matrix Q and the initial distribution of the associated Markov process and the r arrival rates $\lambda_1, \lambda_2, \cdots, \lambda_r$. We use the notation

$$\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \cdots, \lambda_r)^T,$$

and

$$\Delta = diag(\lambda_1, \lambda_2, \cdots, \lambda_r)$$

We assume that the MMPP is homogeneous, which means Q and Δ do not depend on t.

3.3.2 Interpretation

One of the most common mistakes made about MMPPs is to think that they are renewal processes. A renewal process is a generalization of Poisson process in the sense that the inter-arrival times are not necessarily distributed according to an exponential law, they only have to be independent and identically distributed. A Markov renewal process is a stochastic process for which the embedded jump chain (the discrete chain recording the successive states of the continuous chain) is a Markov chain and each inter-arrival time depends on the two states delimiting this same inter-arrival time. Here is a rough justification: assume that there is a first arrival at time t_1 and another consecutive one at time t_2, and suppose that $C(t_1) = i$ and $C(t_2) = j$. During the evolution of the Markov chain, there is a first transition or jump between state i to another state k (which can be j) and so there is a first evolution according to the rate λ_i. Moreover, when the final state j is attained, there is no arrival until time t_2 and therefore there is another evolution according to the rate λ_j. We conclude that the distribution between t_1 and t_2 depends on state i as well as state j, which implies that an MMPP is a MAP.

One thing one might want to do is to extract the transition probability matrix for the embedded Markov chain, that is the discrete-time process $\{C_k\}_{k\in\mathbb{N}} = \{C(T_k)\}_{k\in\mathbb{N}}$ with $\{T_k\}_{k\in\mathbb{N}}$ the arrival times. Let us define the inter-arrival times by $\{D_k\}_{k\in\mathbb{N}}$ where $D_k = T_{k+1} - T_k$. Then the bivariate process $\{C_k, D_k\}_{k\in\mathbb{N}}$ is a Markov renewal process with transition probability matrix (the proof is omitted)

$$\begin{aligned} F(d) &= \int_0^d e^{(Q-\Delta)u} du\ \Delta \\ &= [I - e^{(Q-\Delta)d}](\Delta - Q)^{-1}\Delta \end{aligned}$$

where the elements $F_{ij}(d)$ represent the conditional probabilities

$$P[C_k = j, D_k \leq d \mid C_{k-1} = i].$$

Finally, making $d \to \infty$, we easily see that $F(\infty) = (\Delta - Q)^{-1}\Delta$ which is a stochastic matrix, the transition probability matrix of the Markov chain embedded at arrival times.

Besides, one common approximation when talking about MMPPs is to approximate the distribution of the underlying Markov chain at time t $\boldsymbol{p_t}$ by the stationary vector $\boldsymbol{\pi}$. Therefore, it is clear that the output of a MMPP can modeled as the mixture of exponential variables with probability density function

$$f(t) = \sum_{i=1}^{r} \pi_i \lambda_i e^{-\lambda_i t}.$$

This mixture is called the *hyperexponential random distribution*. In fact the parameters $\boldsymbol{\lambda}$ can be directly estimated from the hyperexponential density except for the Q-matrix. This Q parameter is one the parameter I was able to evaluate thanks to an adaptation of the Baum-Welch algorithm that we will discuss in the next section.

4 A continuous-time version of the Baum-Welch algorithm

This section describes in details the work I achieved. There is one remaining issue related to HMMs we have not discussed yet. This concerns the problem of determining the parameters of a Hidden Markov model. Namely we would like to compute the initial distribution ν of the underlying Markov chain, the conditional probability matrix G which gives the probability of the outcome of an observation conditionally on the hidden state, and finally the one-step transition matrix P of the underlying Markov chain. In general when we focus on a particular application, it is very rare that the parameters are known from the operator. Therefore the first work to do is to choose an appropriate model based on the data and the context of the specific application, and then try to accurately estimate the parameters. Hence when a Hidden Markov model appears to be suited, the first task is to apply the famous Baum-Welch algorithm to estimate the relevant parameters cited above. The following subsection gives a precise idea of how it works.

4.1 The Baum-Welch algorithm

As mentioned earlier on, the Baum-Welch algorithm is an early version of the Expectation-Maximization algorithm. One should first present the Expectation-Maximization algorithm before presenting The Baum-Welch algorithm which is just an application of the EM algorithm to HMMs.

4.1.1 The Expectation-Maximization algorithm

The EM algorithm is a method based on the computation of Maximum likelihood estimates (MLE) and was first introduced in a 1977 paper by Arthur Dempster, Nan Laird, and Donald Rubin. In essence, the EM algorithm is an iterative method aiming at finding MLEs, based on the idea of replacing one difficult likelihood maximization by a sequence of easier maximizations. The EM algorithm has been extensively used for incomplete data models. In this kind of model, a part of the data is missing or unobservable and we want to find MLEs to overcome this problem. We directly understand why such an algorithm is suitable for Hidden Markov models since clearly the hidden states are unobservable (they are fictive).

In the context of incomplete data models, we usually denote by $(\boldsymbol{X}, \boldsymbol{Y}) = (X_1, \cdots, X_n, Y_1, \cdots, Y_n)$ the vector of complete data, where $\boldsymbol{Y} = (Y_1, \cdots, Y_n)$ is the observed data, and $\boldsymbol{X} = (X_1, \cdots, X_n)$ is the unobserved data. We will also use the notation $L(\theta; \boldsymbol{x}, \boldsymbol{y})$ to describe the likelihood function of the complete data where θ is a vector of unknown parameters of size K. The EM algorithm performs an alternative sequence of expectation steps (E), which compute the expected value of the log-likelihood (we always work with log-likelihood to avoid underflow) under the current estimates of parameters, and maximization steps (M) which compute a parameters maximizing of the expected log-likelihood found on the (E) step. In fact the maximum likelihood estimate is determined by the marginal likelihood function $L(\theta; \boldsymbol{x})$

$$L(\theta; \boldsymbol{x}) = \int L(\theta; \boldsymbol{x}, \boldsymbol{y}) d\boldsymbol{x}$$

which is the actual likelihood of the observations. Of course, the integral symbol is replaced by a sum symbol if $\boldsymbol{X}$ has a discrete state-space. However this marginal likelihood is often intractable (see section 2.3.1). Instead, we maximize $L(\theta; \boldsymbol{x})$ by working with only $L(\theta; \boldsymbol{x}, \boldsymbol{y})$

and the conditional probability density function of $\boldsymbol{X}$ given $\boldsymbol{Y}$ and θ. This conditional probability densitity function (pdf) can be easily expressed as

$$P(\boldsymbol{x} \mid \boldsymbol{y}; \theta) = \frac{L(\theta; \boldsymbol{x}, \boldsymbol{y})}{L(\theta; \boldsymbol{x})} = \frac{L(\theta; \boldsymbol{x}, \boldsymbol{y})}{\int L(\theta; \boldsymbol{x}, \boldsymbol{y}) d\boldsymbol{x}}$$

Note that the logarithm is an increasing function and maximizing the log-likelihood is equivalent to maximizing the likelihood itself. Here the likelihood function of observations is assumed to be positive (which is always the case since the likelihood is always expressed as a probability function) and so it is possible to take the log. Therefore during an E-step, we want to compute the expected value of the full log-likelihood $\log L(\theta; \boldsymbol{X}, \boldsymbol{y})$, when $\boldsymbol{X}$ given $\boldsymbol{Y}$ is distributed according to the conditional pdf $P(\boldsymbol{x} \mid \boldsymbol{y}; \theta')$ for a possibly different value θ' of the parameter (which will represent the most recent version of the estimated parameter during the iterative process of the EM algorithm). Therefore the quantity of interest called the intermediate quantity or the expectation function is

$$\mathcal{Q}(\theta; \theta') = E[\log L(\theta; \boldsymbol{X}, \boldsymbol{x}) \mid \theta', \boldsymbol{y}] = \int \log L(\theta; \boldsymbol{x}, \boldsymbol{y}) P(\boldsymbol{x} \mid \boldsymbol{y}; \theta') d\boldsymbol{x}$$

Note that it is important to distinguish between the first and second arguments of $\mathcal{Q}$. The second argument is just a conditioning argument to the expectation and is regarded as fixed and known during an E-step. Now in the rest of the project we assume that the observations are fixed $\boldsymbol{Y} = \boldsymbol{y}$ and we will denote $L(\theta; \boldsymbol{y}) = L(\theta)$. The concavity of the log function implies the following inequality holds for the intermediate quantity:

$$\log L(\theta) - \log L(\theta') \geq \frac{\mathcal{Q}(\theta; \theta') - \mathcal{Q}(\theta'; \theta')}{L(\theta')}$$

Since $L(\theta') \in [0, 1]$ we can further claim that

$$\log L(\theta) - \log L(\theta') \geq \mathcal{Q}(\theta; \theta') - \mathcal{Q}(\theta'; \theta')$$

Now this is the key point of the EM algorithm. Instead of trying to maximize the log-likelihood, we see that if we manage to find a value of θ such that $\mathcal{Q}(\theta, \theta')$ is increased over $\mathcal{Q}(\theta', \theta')$, then it will correspond to an increase of $\log L(\theta)$ that is at least as large. This is the strength of the EM algorithm and this is the reason why it is reputed for its speed of convergence.

To sum up what has been said, the EM algorithm is an iterative process that construct recursively a sequence $(\theta^i)_{i \geq 1}$ of parameter estimates with an initial value (guess) θ_0 and alternates between two kinds of steps as follows:

- E-step: calculate $\mathcal{Q}(\theta, \theta^i)$
- M-step: Find a value θ^{i+1} of θ that maximizes $\mathcal{Q}(\theta, \theta^i)$

Figure 7 depicts the EM algorithm.

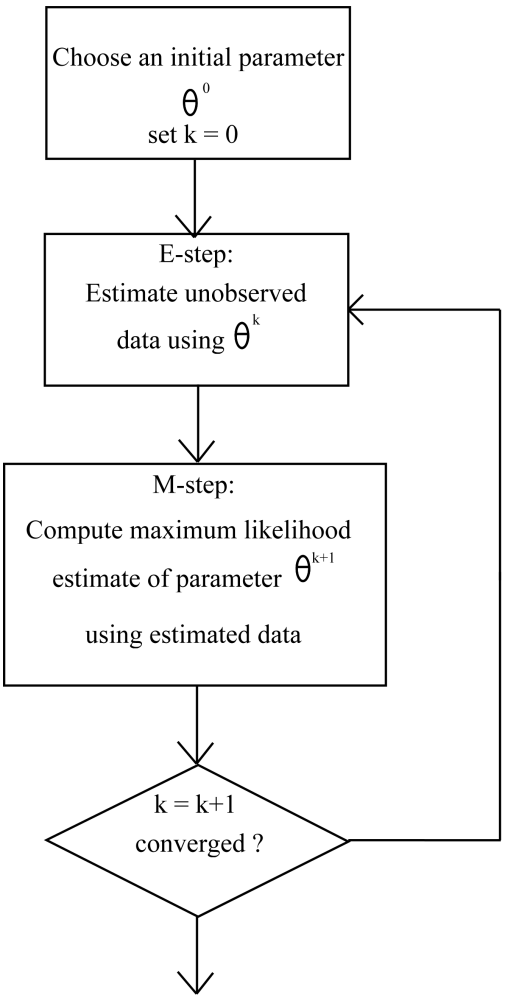

Figure 7: An overview of the EM algorithm

From the relation $\log L(\theta) - \log L(\theta') \geq \mathcal{Q}(\theta;\theta') - \mathcal{Q}(\theta';\theta')$, it can be shown that $\{\theta^i\}_{i\in\mathbb{N}}$ converges to the incomplete-data MLE. In fact the paper published by Dempster,Laird and Rubin didn't prove the convergence correctly (a flaw in the proof was discovered). It is only in 1983 that C. F. Jeff Wu finally presented a correct and rigorous proof. In fact what we have shown is that the EM algorithm does increase the likelihood function of observations but it doesn't guarantee to obtain the maximum likelihood estimators of parameters. Indeed, depending on the initial value θ^0 we may be caught in a local maximum. There are several complex methods to escape local maximum attractions amongst which we can cite the random restart method consisting in trying different initial values.

4.1.2 The Baum-Welch algorithm

In order to derive the Baum-Welch algorithm, we just have to apply the EM algorithm to our Hidden Markov model which is indeed an incomplete data model where 'the missign data' are hidden states $c_1, \cdots, c_n$ occupied by the Markov chain, and the 'complete data' are $s_1, \cdots, s_n, c_1, \cdots, c_n$. Furthermore the parameter θ can be seen as the conjunction $\theta = [\nu, P, G]$. We recall the reader that the Markov chain $\{C_t\}_{t\in\mathbb{N}}$ has a state space $\mathcal{C} = \{1, \cdots, r\}$ of size r and the stochastic process $\{S_t\}_{t\in\mathbb{N}}$ has a state space $\mathcal{S} = \{1, \cdots, m\}$

of size m, and this implies that ν is a vector of size r, P is a r-x-r matrix, and G a r-x-m matrix and finally we have $r^2 + (m+1)r$ parameters to estimate.

The complete likelihood function (see section 2.1) is given by

$$J_{\nu,n}(\theta; c_0^n, s_0^n) = \nu_{c_0} g_{c_0,s_0} \prod_{i=1}^{n} p_{c_{i-1}c_i} g_{c_i s i}$$

and the log-likelihood is therefore

$$\log J_{\nu,n}(c_0^n, s_0^n) = \log \nu_{c_0} + \sum_{i=0}^{n-1} \log p_{c_i c_{i+1}} + \sum_{i=0}^{n} \log g_{c_i s_i}$$

and the intermediate quantity of EM is

$$\begin{aligned}
\mathcal{Q}(\theta;\theta') &= E[\log J_{\nu,n}(C_0^n, s_0^n) \mid S_0^n = s_0^n; \theta'] \\
&= \sum_{c_0,\cdots,c_n} \log J_{\nu,n}(C_0^n, s_0^n) P(C_0^n = c_0^n \mid S_0^n = s_0^n; \theta') \\
&= \sum_{c_0,\cdots,c_n} \log \nu_{c_0}\, P(C_0^n = c_0^n \mid S_0^n = s_0^n; \theta') + \\
&\quad \sum_{c_0,\cdots,c_n} \sum_{i=0}^{n-1} \log p_{c_i c_{i+1}}\, P(C_0^n = c_0^n \mid S_0^n = s_0^n; \theta') + \\
&\quad \sum_{c_0,\cdots,c_n} \sum_{i=0}^{n} \log g_{c_i s_i}\, P(C_0^n = c_0^n \mid S_0^n = s_0^n; \theta') \\
&= \sum_{c_0} \log \nu_{c_0} \sum_{c1,\cdots,c_n} P(C_0^n = c_0^n \mid S_0^n = s_0^n; \theta') + \\
&\quad \sum_{i=0}^{n-1} \sum_{c_i,c_{i+1}} \log p_{c_i c_{i+1}} \sum_{c_0,\cdots,c_{i-1},c_{i+2},\cdots,c_n} P(C_0^n = c_0^n \mid S_0^n = s_0^n; \theta') + \\
&\quad \sum_{i=0}^{n} \sum_{c_i} g_{c_i,s_i} \sum_{c_0,\cdots,c_{i-1},c_{i+1}\cdots,c_n} P(C_0^n = c_0^n \mid S_0^n = s_0^n; \theta') \\
&= \sum_{c_0} \log \nu_{c_0} P(C_0 = c_0 \mid S_0^n = s_0^n; \theta') + \\
&\quad \sum_{i=0}^{n-1} \sum_{c_i,c_{i+1}} \log p_{c_i c_{i+1}} P(C_i = c_i, C_{i+1} = c_{i+1} \mid S_0^n = s_0^n; \theta') + \\
&\quad \sum_{i=0}^{n} \sum_{c_i} g_{c_i,s_i} P(C_i = c_i \mid S_0^n = s_0^n; \theta')
\end{aligned}$$

And we finally obtain

$$\begin{aligned}
\mathcal{Q}(\theta;\theta') &= \sum_{c_0} \log \nu_{c_0} \phi_{0|n}(c_0;\theta') + \sum_{i=0}^{n-1} \sum_{c_i,c_{i+1}} \log p_{c_i c_{i+1}} \phi_{i:i+1|n}(c_i, c_{i+1}; \theta') \\
&\quad + \sum_{i=0}^{n} \sum_{c_i} g_{c_i,s_i} \phi_{i|n}(c_i;\theta')
\end{aligned}$$

As one can see, the intermediate quantity $\mathcal{Q}(\theta;\theta')$ depends only on the computation of the marginal distributions $\phi_{i|n}$ and $\phi_{i,i+1|n}$, which can be computed using the Forward-Backward algorithm described in section 2.3.4. Note that here the argument θ' is of great importance since we want to maximize $\mathcal{Q}(\theta;\theta')$ with respect to θ, that is with respect to ν, P and G and so the ϕ coefficients will remain constant since they do not depend on the same parameters.

Proposition 4.1 *The parameters that maximise $\mathcal{Q}(\theta;\theta')$ are:*
For $1 \leq j \leq r$, $1 \leq k \leq r$ and $1 \leq s \leq m$,

$$\hat{\nu}_j = \frac{\phi_{0|n}(j;\theta')}{\sum_l \phi_{0|n}(l;\theta')}$$

$$\hat{p}_{jk} = \frac{\sum_{i=0}^{n-1} \phi_{i:i+1|n}(j,k;\theta')}{\sum_{i=0}^{n-1} \phi_{i|n}(j;\theta')}$$

$$\hat{g}_{js} = \frac{\sum_{i=0}^{n} \delta_{s_i,s}\phi_{i|n}(j;\theta')}{\sum_{i=0}^{n} \phi_{i|n}(j;\theta')}$$

Proof *To prove this previous proposition we are going to set all the derivatives (w.r.t each parameter) to zero taking into account the probability constraints. In fact this task is relatively easy since $\mathcal{Q}(\theta;\theta')$ is composed of three different terms as follows:*

- $T_1(\nu) = \sum_{c_0} \log \nu_{c_0} \phi_{0|n}(c_0;\theta')$
- $T_2(P) = \sum_{i=0}^{n-1} \sum_{c_i,c_{i+1}} \log p_{c_i c_{i+1}} \phi_{i:i+1|n}(c_i,c_{i+1};\theta')$
- $T_3(G) = \sum_{i=0}^{n} \sum_{c_i} g_{c_i,s_i} \phi_{i|n}(c_i;\theta')$

and only one parameter among ν, P and G appear in each of these terms. So maximizing $\mathcal{Q}(\theta;\theta')$ corresponds to maximizing T_1 with respect to ν, T_2 with respect to P and T_3 with respect to G.
Firstly, considering T_1, adding the Lagrange multiplier μ, using the constraint $\sum_l \nu_l = 1$, and setting the derivative to zero we have

$$\frac{\partial}{\partial \nu_j}[T_1(\nu) - \mu \sum_l \nu_l] = \frac{1}{\nu_j}\phi_{0|n}(j;\theta') - \mu = 0$$

So we find

$$\hat{\nu}_j = \frac{1}{\mu}\phi_{0|n}(j;\theta')$$

To remove μ we use the constraint $\sum_l \hat{\nu}_l = 1$ which gives

$$1 = \frac{1}{\mu}\sum_l \phi_{0|n}(l;\theta')$$

$$\mu = \sum_l \phi_{0|n}(l;\theta')$$

Finally we get

$$\hat{\nu}_j = \frac{\phi_{0|n}(j;\theta')}{\sum_l \phi_{0|n}(l;\theta')}$$

Secondly, considering T_2, adding the Lagrange multiplier μ, using the constraints $\sum_l p_{jl} = 1$ (r constraints, one for each different value of j), and setting the derivative to zero we have

$$\frac{\partial}{\partial p_{jk}}[T_2(P) - \mu \sum_l p_{jk}] = \sum_{i=0}^{n-1} \phi_{i:i+1|n}(j,k;\theta') - \mu = 0$$

So we find

$$\hat{p}_{jk} = \frac{1}{\mu} \sum_{i=0}^{n-1} \phi_{i:i+1|n}(j,k;\theta')$$

To remove μ we use the constraint $\sum_k \hat{p}_{jl} = 1$ which gives

$$1 = \frac{1}{\mu} \sum_{i=0}^{n-1} \phi_{i:i+1|n}(j,k;\theta')$$

$$\mu = \sum_{i=0}^{n-1} \phi_{i|n}(j;\theta')$$

Finally we get

$$\hat{q}_{jk} = \frac{\sum_{i=0}^{n-1} \phi_{i:i+1|n}(j,k;\theta')}{\sum_{i=0}^{n-1} \phi_{i|n}(j;\theta')}$$

Secondly, considering T_3, adding the Lagrange multiplier μ, using the constraints $\sum_s g_{js} = 1$ (r constraints, one for each different value of j), and setting the derivative to zero we have

$$\frac{\partial}{\partial g_{js}}[T_3(G) - \mu \sum_s g_{js}] = \sum_{i=0}^{n} \frac{\delta_{s_i,s}}{g_{js}} \phi_{i|n}(j;\theta') - \mu = 0$$

So we find

$$\hat{g}_{js} = \frac{1}{\mu} \sum_{i=0}^{n} \delta_{s_i,s} \phi_{i|n}(j;\theta')$$

To remove μ we use the constraint $\sum_s \hat{g}_{js} = 1$ which gives

$$1 = \frac{1}{\mu} \sum_s \sum_{i=0}^{n-1} \phi_{i:i+1|n}(j,k;\theta')$$

$$\mu = \sum_{i=0}^{n} \phi_{i|n}(j;\theta')$$

Finally we get

$$\hat{g}_{js} = \frac{\sum_{i=0}^{n} \delta_{s_i,s} \phi_{i|n}(j;\theta')}{\sum_{i=0}^{n} \phi_{i|n}(j;\theta')}$$

Note 4.2 We notice that the optimal parameters ν_j, p_{jk} and g_{js} are all expressed as functions of conditional distributions $\phi_{0|n}$, $\phi_{i|n}$ and $\phi_{i:i+1|n}$. However, we can't use them directly because we don't know how to compute them efficiently. Instead we should try to reformulate them using the forward variable $\alpha_{\nu,k}(c)$ and the backward variable $\beta_{k|n}(c)$. In fact for the first two ones, it is very straightforward, because we already saw them in Section 2.3.1,

$$\phi_{0|n}(j;\theta') = P(C_0 = j \mid S_0^n = s_0^n) = \frac{1}{L_{\nu,n}(s_0^n)} \alpha_{\nu,0}(j,s_0) \beta_{0,n}(j,s_1^n)$$

$$\phi_{i|n}(j;\theta') = P(C_i = j \mid S_0^n = s_0^n) = \frac{1}{L_{\nu,n}(s_0^n)}\alpha_{\nu,i}(j, s_0^i)\beta_{i,n}(j, s_{i+1}^n)$$

For $\phi_{i:i+1|n}(c_i, c_{i+1};\theta')$ we must go a bit further

$$\begin{aligned}
\phi_{i:i+1} \mid n(c_i, c_{i+1};\theta') &= P(C_i = c_i, C_{i+1} = c_{i+1} \mid S_0^n = s_0^n) \\
&= \sum_{c_0,\cdots,c_{i-1},c_{i+2},\cdots,c_n} \frac{1}{P(S_0^n = s_0^n)} P(C_0^n = c_0^n, S_0^n = s_0^n) \\
&= \frac{1}{P(S_0^n = s_0^n)} \sum_{c_0,\cdots,c_{i-1},c_{i+2},\cdots,c_n} P(S_0^n = s_0^n \mid C_0^n = c_0^n) P(C_0^n = c_0^n) \\
&= \frac{1}{P(S_0^n = s_0^n)} \sum_{c_0,\cdots,c_{i-1},c_{i+2},\cdots,c_n} P(S_0^i = s_0^i \mid C_0^i = c_0^i) \times \\
&\quad P(S_{i+1} = s_{i+1} \mid C_{i+1} = c_{i+1}) P(S_{i+2}^n = s_{i+2}^n \mid C_{i+1}^n = c_{i+1}^n) \times \\
&\quad \frac{P(C_0^i = c_0^i) p_{c_i,c_{i+1}} P(C_{i+1}^n = c_{i+1}^n)}{P(C_{i+1} = c_{i+1})} \\
&= \frac{1}{L_{\nu,n}} p_{c_i,c_{i+1}} \alpha_{\nu,i}(c_i) g_{c_{i+1},s_{i+1}} \beta_{i+1|n}(c_{i+1}).
\end{aligned}$$

Now replacing the ϕ terms with these new expressions leads us to the famous Baum-Welch (re)-estimates that are given below.

Proposition 4.3 (The Baum-Welch re-estimates) *During a M-step of the EM algorithm, the optimal re-estimates for the HMM parameters are*

$$\hat{\nu}_j = \frac{\alpha_{\nu,0}(j)\beta_{0|n}(j)}{\sum_l \alpha_{\nu,0}(l)\beta_{0|n}(l)}$$

$$\hat{q}_{jk} = \frac{\sum_{i=0}^n \delta_{s_i,s}\alpha_{\nu,i}(j)\beta_{i|n}(j)}{\sum_{i=0}^n \alpha_{\nu,i}(j)\beta_{i|n}(j)}$$

$$\hat{g}_{js} = \frac{p_{jk}\sum_{i=0}^{n-1} \alpha_{\nu,i}(j) g_{k,s_{i+1}} \beta_{i+1|n}(k)}{\sum_{i=0}^{n-1} \alpha_{\nu,i}(j)\beta_{i|n}(j)}$$

We can now implement the Baum-Welch algorithm along with the Forward-Backward algorithm. Besides, to avoid underflow problems we already saw a normalized version of the Forward-Backward algorithm using normalized variables $\overline{\alpha}_{\nu,k}(c)$ and $\overline{\beta}_{k|n}(c)$. Also we should derive a corresponding normalized version of the Baum-Welch algorithm making use of these new variables. This is very easy since one can notice that $\phi_{i|n}(j;\theta') = \overline{\alpha}_{\nu,i}(j)\cdot\overline{\beta}_{i|n}(j)$. Moreover using the last expression of $\phi_{i:i+1|n}(c_i, c_{i+1};\theta')$ we can write

$$\phi_{i:i+1|n}(j,k;\theta') = d_{\nu,i+1}^{-1} p_{jk} g_{k,s_{i+1}} \overline{\alpha}_{\nu,i}(j)\overline{\beta}_{i+1|n}(j).$$

Proposition 4.4 (Normalized Baum-Welch) *During a M-step of the EM algorithm, the optimal re-estimates for the HMM parameters are*

$$\hat{\nu}_j = \frac{\overline{\alpha}_{\nu,0}(j)\overline{\beta}_{0|n}(j)}{\sum_l \overline{\alpha}_{\nu,0}(l)\overline{\beta}_{0|n}(l)}$$

$$\hat{q}_{jk} = \frac{\sum_{i=0}^n \delta_{s_i,s}\overline{\alpha}_{\nu,i}(j)\overline{\beta}_{i|n}(j)}{\sum_{i=0}^n \overline{\alpha}_{\nu,i}(j)\overline{\beta}_{i|n}(j)}$$

$$\hat{g}_{js} = \frac{p_{jk}\sum_{i=0}^{n-1} d_{\nu,i+1}\overline{\alpha}_{\nu,i}(j) g_{k,s_{i+1}} \overline{\beta}_{i+1|n}(k)}{\sum_{i=0}^{n-1} \overline{\alpha}_{\nu,i}(j)\overline{\beta}_{i|n}(j)}$$

4.2 Continuous-time approach

Section 2 emphasizes the necessity to adopt a different method for the treatment of the instruction streams arriving at the Flash memory chip. We saw that 5-ms bins don't capture correctly the low level behaviour of write instructions (see the comparison of autocorrelation functions in figure 2) and 1-ms bins create a lot of noise (many empty bins). This is why we decided to consider the instructions arriving at the Flash memory as a continuous-time Hidden Markov model. Therefore, we will consider the continuous-time bivariate stochastic process $\{C(t), S(t)\}_{t\geq 0}$ where $\{C(t)\}_{t\geq 0}$ is a continuous-time Markov chain with state-space $\mathcal{C} = \{1, 2, \ldots, r\}$ representing the hidden state at a given time and $\{S(t)\}_{t\geq 0}$ is the stochastic process of observations with state-space $\mathcal{S} = \{1, 2, \ldots, m\}$. Moreover we will also denote by $\{T_i\}_{i\in\mathbb{N}}$ the arrival times of observations satisfying $T_1 < T_2 < \cdots < T_n < \cdots$. One underlying assumption that will be made is that the embedded bivariate stochastic process at arrival times is a HMM. The aim of this section is to evaluate the relevant parameters which are the initial distribution of the CTMC ν, the G-matrix and the infinitesimal generator of the underlying Markov process Q. To achieve this, we need to define a new likelihood function.

4.2.1 Likelihood function

We have to keep in mind that we have at our disposal a time-stamped trace and that somehow we want to apply the Baum-Welch algorithm or more exactly the EM algorithm to compute estimates. Therefore, for the rest of the project, we will consider $\{T_i\}_{i\in\mathbb{N}}$=$\{t_i\}_{i\in\mathbb{N}}$ where $\{t_i\}_{i\in\mathbb{N}}$ is a fixed sequence of observation times. Moreover the state space of observations will be $\mathcal{S} = \{1, \ldots, m\}$ with $m = 2$ representing the two instructions READ and WRITE. Hence, intuitive definitions for the likelihood function of observations and the complete date likelihood are, respectively,

$$L(s_0^n) = P[\{S(t_i) = s_i\}_{i=0,1,\ldots,n}]$$

$$L(s_0^n, c_0^n) = P[\{S(t_i) = s_i, C(t_i) = c_i\}_{i=0,1,\ldots,n}].$$

One should now express the full likelihood function according to the relevant parameters we want to determine. Here is one way to do this

$$\begin{aligned} L(s_0^n, c_0^n) &= P[\{S(t_i) = s_i\}_{i=0,1,\ldots,n} \mid \{C(t_i) = c_i\}_{i=0,1,\ldots,n}] \times \\ &\quad P[\{C(t_i) = c_i\}_{i=0,1,\ldots,n}] \end{aligned}$$

The new likelihood function is a product of two probabilities. The first probability can be worked out similarly as the as Section 2:

$$\begin{aligned} P[\{S(t_i) = s_i\}_{i=0,1,\ldots,n} \mid \{C(t_i) = c_i\}_{i=0,1,\ldots,n}] &= \prod_{i=0}^{n} P[S(t_i) = s_i \mid C(t_i) = c_i] \\ &= \prod_{i=0}^{n} g_{c_i,s_i} \end{aligned}$$

Concerning the second factor, using the Markov property of $\{C(t), S(t)\}_{t\geq 0}$, we write

$$\begin{aligned} P[\{C(t_i)=c_i\}_{i=0,1,\dots,n}] &= P[C(t_n)=c_n \mid C(t_{n-1})=c_{n-1},\dots,C(t_0)=c_0] \\ &\quad P[C(t_{n-1})=c_{n-1} \mid C(t_{n-2})=c_{n-2},\dots,C(t_0)=c_0]\dots \\ &\quad P[C(t_1)=c_1=c_1 \mid C(t_0)=c_0]P[C(t_0=c_0] \\ &= P[C(t_0)=c_0]\cdot\prod_{i=0}^{n-1} P[C(t_{i+1})=c_{i+1} \mid C(t_i)=c_i] \end{aligned}$$

Using the notation introduced in section 3.2.1, and denoting $P[C(t_0)=c_0]=\nu'_{c_0}$ and $d_k=t_{k+1}-t_k$, we can further write

$$P[\{C(t_i)=c_i\}_{i=0,1,\dots,n}] = \nu'_{c_0}\prod_{i=0}^{n-1} P_{d_k}(c_i,c_{i+1})$$

We recall the reader that $\boldsymbol{P_t}=(P_t(i,j))_{(i,j)\in\mathcal{C}\times\mathcal{S}}$ and each entry (i,j) of this matrix is the conditional probability

$$P_t(i,j)=P[C(t)=j \mid C(0)=i]=P[C(t+u)=j \mid C(u)=i] \qquad \text{for all } u\geq 0$$

Finally we can write the complete data likelihood function as

$$L(s_0^n,c_0^n)=\nu'_{c_0}\prod_{i=0}^{n} g_{c_i,s_i}\prod_{i=0}^{n-1} P_{d_k}(c_i,c_{i+1})$$

We can see there are many similarities with the discrete-case complete data likelihood. The first factor ν'_{c_0} is the probability that the hidden state at time t_0 is c_0. If $t_0=0$ then we have the identity $\nu'_{c_0}=\nu_{c_0}$ but this would mean that we have necessarily an observation at the origin time which is not a trivial assumption and it turns out that this is not the case for our trace-stamped trace. However, still referring to section 3.2.1, we can have an easy relation between them which is $\nu'=\nu\boldsymbol{P_{t_0}}$. But to make an analogy with the discrete case we will work with ν' instead of ν and use the previous relation at the very end to obtain ν. The other difference with the discrete-case is the product $\prod_{i=0}^{n-1} P_{d_k}(c_i,c_{i+1})$ because in the discrete case we had the product $\prod_{i=0}^{n-1} p_{c_i,c_{i+1}}$ which makes explicitly appear the relevant parameters, namely the transition matrix P. In continuous time it is a little bit more complex and we have to work harder to deal with $\prod_{i=0}^{n-1} P_{d_k}(c_i,c_{i+1})$. In fact I thought about two solutions, the first one is an approximated solution which is not applicable if we don't know more informations about the Markov process whereas the second solution is an exact one.

4.2.2 First solution

The infinitesimal generator matrix Q is deeply related to the behaviour of the Markov chain for short period of time. Indeed in Section 3.2 we saw that $\lim_{h\to 0}\frac{P[X(t+h)=j|X(t)=i]}{h}=q_{ij}$. So my first idea was to find some sort of iterative of recursive method to change the scale of the problem until a certain extent. This is why I used a binary chop method to calculate

the product $\prod_{i=0}^{n-1} P_{d_k}(c_i, c_{i+1})$ in the following way

$$\begin{aligned}
P_{d_k}(c_k, c_{k+1}) &= \sum_{c' \in \mathcal{C}} P[C(t_{k+1} = c_{k+1}, C(\frac{t_{k+1}+t_k}{2}) = c' \mid C(t_k) = c_k] \\
&= \sum_{c' \in \mathcal{C}} P[C(t_{k+1} = c_{k+1} \mid C(\frac{t_{k+1}+t_k}{2}) = c', C(t_k) = c_k] \cdot \\
&\quad \frac{P[C(\frac{t_{k+1}+t_k}{2}) = c', C(t_k) = c_k]}{P[C(t_k) = c_k]} \\
&= \sum_{c' \in \mathcal{C}} P[C(t_{k+1} = c_{k+1} \mid C(\frac{t_{k+1}+t_k}{2}) = c'] \cdot P[C(\frac{t_{k+1}+t_k}{2}) = c' \mid C(t_k) = c_k] \\
&= \sum_{c' \in \mathcal{C}} P_{d_k/2}(c_k, c') \cdot P_{d_k/2}(c', c_{k+1})
\end{aligned}$$

We reiterate the same process for $p^{d_k/2}_{c_k,c'}$ and $p^{d_k/2}_{c',c_{k+1}}$ until d_k becomes small enough (threshold that we need to establish) such that we can use the first order approximation :

$$P_{d_k}(c_k, c_{k+1}) = q_{c_k,c_{k+1}} \cdot d_k + o(d_k) \quad if \quad c_k \neq c_{k+1}$$

$$P_{d_k}(c_k, c_k) = \int_{d_k}^{\infty} q_{c_k,c_k} e^{-q_{c_k,c_k} t} \, \mathrm{d}t$$

For instance, if we assume that these previous approximations are acceptable for $d_k \leq \epsilon$ (ϵ being a fixed threshold), then we will have to reiterate the process described above $\log_2(d_k/\epsilon) + 1$ times.

There are three main problems that make this solution inapplicable in practice. The first is one concerns the fact that we don't have any expression for $o(d_k)$ and so we can't quantify the error we make. The second is that it is not obvious how to set the value of the threshold ϵ since it is definitely process-dependent (the slower is the Markov chain, the greater is ϵ). The third problem is that this method is very greedy computationally speaking and so it does not worth implementing it.

4.2.3 Second solution

The second solution is based on the result $\boldsymbol{P_t} = e^{tQ}$ and so providing that we can express explicitly every entry of e^{tQ} according to $\{q_{ij}\}$ we will be able to apply the EM algorithm to derive the MLEs for the parameters. Here again there are three different possibilities to express e^{tQ} that we shall describe.

The Sylvester method. This result states that e^{tQ} is a polynomial of tQ of degree $r-1$. That is to say we have

$$e^{tQ} = \sum_{i=0}^{r-1} \alpha_i(t) Q^i.$$

This result can be proven using the Cayley-Hamilton theorem stating that Q is a root of its characteristic polynomial $P(X) = \sum_{i=0}^{n} a_i X^i$ and therefore we have

$$P(Q) = 0 \iff \sum_{i=0}^{n} a_i Q^i = 0$$

And so we can express the nth power of Q as a function of the first $n-1$ powers (knowing that $a_n \neq 0$):

$$Q^n = -\frac{1}{a_n} \sum_{i=0}^{n-1} a_i Q^i$$

By induction, we can therefore show that the pth power of Q can also be expressed with only the first $n-1$ ones and since e^{tQ} is a mere linear combination of powers of Q ($e^{tQ} = \sum_{n=0}^{\infty} \frac{t^n}{n!} Q^n$) it is also a linear combination of the first $n-1$ powers of Q and this concludes the proof. The fundamental problem is the determination of the coefficients $\alpha_i(t)$ which seems to be infeasible in our context.

Diagonalization method. From now on and for more simplicity we will assume that there are only two hidden states. The diagonalization method is based on the following result: if $D = diag(\lambda_1, \ldots, \lambda_n)$, then $D^k = diag(\lambda_1^k, \ldots, \lambda_n^k)$ and consequently $e^D = diag(e^{\lambda_1}, \ldots, e^{\lambda_n})$. This means if we manage to diagonalize Q we will be able to reach our objective. Let us begin the diagonalization method. Since there are only two hidden states, then there exist two parameters $a, b > 0$ such that $Q = \begin{pmatrix} -a & a \\ b & -b \end{pmatrix}$. We first need to calculate the eigenvalues of Q:

$$det(\lambda I - Q) = \begin{vmatrix} \lambda + a & -a \\ -b & \lambda + b \end{vmatrix} = \lambda(\lambda + a + b).$$

We find out two eigenvalues $\lambda_1 = 0$ and $\lambda_2 = -(a+b)$. At this point we are still not sure that Q is diagonalizable. To find out the eigenvectors associated with the eigenvalue $\lambda_1 = 0$, we must solve the following system of equations

$$\begin{cases} -ax + ay = 0 \\ bx - by = 0 \end{cases} \iff x = y$$

All the eigenvectors associated with $\lambda_1 = 0$ are of the form (x, x), in particular $(1, 1)$ belongs to this set. To find out the eigenvectors associated with the eigenvalue $\lambda_2 = -(a+b)$, we must solve the following system of equations

$$\begin{cases} -ax + ay = -(a+b)y \\ bx - by = -(a+b)y \end{cases} \iff y = -\frac{b}{a}x$$

All the eigenvectors associated with $\lambda_1 = 0$ are of the form $(x, -\frac{b}{a}x)$, in particular $(a, -b)$ belongs to this set and at this stage we can say that Q is diagonalizable since $(a, -b)$ and $(1, 1)$ are not collinear. Let M be the matrix with these eigenvectors as its columns

$$M = \begin{pmatrix} 1 & a \\ 1 & -b \end{pmatrix}$$

We can finally write the diagonalization equation

$$Q = M \begin{pmatrix} 0 & 0 \\ 0 & -(a+b) \end{pmatrix} M^{-1}$$

We compute M^{-1} using the relation $M^{-1} = \frac{1}{|M|}(adjugate(M))^T = \frac{-1}{a+b} \begin{pmatrix} -b & -a \\ -1 & 1 \end{pmatrix}$. Finally by writing

$$tQ = -\frac{1}{a+b} \begin{pmatrix} 1 & a \\ 1 & -b \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & -(a+b)t \end{pmatrix} \begin{pmatrix} -b & -a \\ -1 & 1 \end{pmatrix}$$

So

$$e^{tQ} = -\frac{1}{a+b} \begin{pmatrix} 1 & a \\ 1 & -b \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & e^{-(a+b)t} \end{pmatrix} \begin{pmatrix} -b & -a \\ -1 & 1 \end{pmatrix}$$

$$\iff e^{tQ} = \frac{1}{a+b} \begin{pmatrix} b + ae^{-(a+b)t} & a - ae^{-(a+b)t} \\ b - be^{-(a+b)t} & a + be^{-(a+b)t} \end{pmatrix}$$

Laplace transform. We can confirm the previous result using the Laplace transform. The Laplace transform is a widely used transform and has many applications, in particular in signal processing and probability theory. In a way it can be considered as a generalization of the Fourier transform. Sometimes a problem may appear to be unsolvable in the temporal domain but by switching it into the Laplace domain and switching it back into the temporal domain we can considerably reduce the difficulty of the problem so that it becomes feasible. If $f(t)$ is a function, its Laplace transform denoted by $F(s)$, where s is a complex number, is given by the integral transform (so it is linear)

$$F(s) = \mathcal{L}(f(t)) = \int_0^\infty e^{-st} f(t)\, dt$$

The Laplace transform is bijective for the majority of usual functions, and the inverse Laplace transform is denoted $\mathcal{L}^{-1}$ and satisfies

$$\mathcal{L}^{-1}(F(s)) = f(t)$$

The Laplace transform of a temporal matrix is the matrix composed of all the Laplace transforms of its entries. We can therefore write that $\mathcal{L}(e^{tQ}) = \int_0^\infty e^{-st} e^{tQ}\, dt$.

Proposition 4.5 *The following identity holds*

$$e^{-st} e^{tQ} = e^{-stI+tQ}$$

Proof *Starting from the right-hand side we write*

$$\begin{aligned}
e^{-stI+tQ} &= \sum_{n=0}^{\infty} \frac{(-stI + tQ)^n}{n!} \\
&= \sum_{n=0}^{\infty} \sum_{k=0}^{n} \frac{1}{n!} \binom{n}{k} (-st)^k I^k\, (tQ)^{n-k} \\
&= \sum_{n=0}^{\infty} \sum_{k=0}^{n} \frac{(-st)^k}{k!} \frac{(tQ)^{n-k}}{(n-k)!} \\
&= \sum_{k=0}^{\infty} \sum_{n=k}^{\infty} \frac{(-st)^k}{k!} \frac{(tQ)^{n-k}}{(n-k)!} \\
&= \sum_{k=0}^{\infty} \sum_{m=0}^{\infty} \frac{(-st)^k}{k!} \frac{(tQ)^{m}}{m!} \\
&= \sum_{n=0}^{\infty} \frac{(-st)^n}{n!} \sum_{m=0}^{\infty} \frac{(tQ)^{m}}{m!} \\
&= e^{-st} e^{tQ}
\end{aligned}$$

Proposition 4.6 *for all real number t and all matrix Q we have:*

$$\mathcal{L}(e^{tQ}) = (sI - Q)^{-1}$$

Proof *The proof is straightforward*

$$\mathcal{L}(e^{tQ}) = \int_0^\infty e^{-stI+tQ}\, dt = (Q - sI)^{-1} \left[e^{-(Q-sI)t}\right]_0^\infty = (sI - Q)^{-1}$$

Let us calculate $e^tQ = \mathcal{L}^{-1}((sI-Q)^{-1}) = \begin{pmatrix} P_t(1,1) & P_t(1,2) \\ P_t(2,1) & P_t(2,2) \end{pmatrix}$.

We denote by $A = sI - Q = \begin{pmatrix} s+a & -a \\ -b & s+b \end{pmatrix}$. In order to take the inverse of A we are going to use the formula

$$A^{-1} = \frac{1}{det(A)} C^T$$

where C is the adjugate matrix (that is the matrix of cofactors). First of all we calculate the determinant

$$det(A) = \mid A \mid = (s+a)(s+b) - ab = s(s+a+b)$$

Then we can directly calculate the matrix of cofactors C which is very straightforward :

$$C = \begin{pmatrix} s+b & b \\ a & s+a \end{pmatrix}$$

Finally we can express the inverse of A as follows

$$A^{-1} = \frac{1}{s(s+a+b)} \begin{pmatrix} s+b & a \\ b & s+a \end{pmatrix}$$

Now we can take the inverse Laplace transform, element by element. Let us begin with the first term of the previous matrix, namely $(A^{-1})_{1,1}$:

$$\mathcal{L}^{-1}((A^{-1})_{1,1}) = \mathcal{L}^{-1}\left(\frac{s+b}{s(s+a+b)}\right)$$

To make use of the table given the most classical inverse Laplace transform, we are going to use partial fraction decomposition on the previous rational fraction.

$$\frac{s+b}{s(s+a+b)} = \frac{\frac{b}{a+b}}{s} + \frac{\frac{a}{a+b}}{s+a+b}$$

Thus we end up with

$$\mathcal{L}^{-1}\left(\frac{s+b}{s(s+a+b)}\right) = \frac{1}{a+b}[b + ae^{-(a+b)t}]\mathcal{U}(t)$$

$\mathcal{U}(t)$ being the Heaviside function which basically equals 0 if $t \leq 0$ and 1 if $t > 0$. Since we are only dealing with positive values of t (we will apply it for $t = d_k = t_{k+1} - t_k > 0$), we will omit it in the remainder of the calculations. By adopting the same procedure to calculate all other inverse Laplace transforms, we obtain this final matrix :

$$e^{tQ} = \frac{1}{a+b} \begin{pmatrix} b + ae^{-(a+b)t} & a - ae^{-(a+b)t} \\ b - be^{-(a+b)t} & a + be^{-(a+b)t} \end{pmatrix}$$

We can check that this is the same matrix we obtained with the diagonalization method. The following proposition is another way to check our result.

Proposition 4.7 *Let A and B be two square matrices of size n with A having each of its rows summing to 0, then the product AB will also have each of its rows summing to 0. In particular for all $p \in \mathbb{N}^*$, the rows of A^p sum to 0.*

Proof *Let $C = AB$ with $C = (c_{ij})_{i,j\in\{1,\dots,n\}}$. The result for the sum of the ith row of C is given by*

$$\sum_{j=1}^{n} c_{ij} = \sum_{j=1}^{n}\sum_{k=1}^{n} a_{ik}b_{kj} = \sum_{k=1}^{n}\sum_{j=1}^{n} a_{ik}b_{kj} = \underbrace{\sum_{k=1}^{n} a_{ik}}_{0}\sum_{j=1}^{n} b_{kj} = 0 \tag{5}$$

Thus $e^{tQ} = (I + tQ + \frac{t^2}{2}Q^2 + \dots)$ must have its rows summing up to 1 which can be easily checked in our case.

4.2.4 Modifications to the Forward-Backward algorithm

Now that we can fully and explicitly express the likelihood function with the parameters $\theta = [\nu', Q, G]$, one would be tempted to copy the Baum-Welch algorithm from the discrete case. However we must keep in mind that the Baum-Welch algorithm uses the Forward and Backward measures and if when we recall the associated recursions

$$\alpha_{\nu,k}(c) = \sum_{c'} \alpha_{\nu,k-1}(c')\, p_{c',c}\, g_{c,s_k}$$

$$\beta_{k|n}(c) = \sum_{c'} p_{c,c'}\, g_{c',s_{k+1}} \beta_{k+1|n}(c')$$

we notice that P, the one-step transition matrix of the discrete Markov chain, appears in it. It would seem that we can't use them for the continuous-time version of the Baum-Welch algorithm. But in fact we can obtain a similar result by bringing some minor modifications. Indeed to make use of the results we have already found out in section 2, we can notice that considering the following transforms

$$\text{event} \quad C_k = c_k \quad \to \text{event} \quad C(t_k) = c_k$$

$$\text{event} \quad S_k = s_k \quad \to \text{event} \quad S(t_k) = s_k$$

we can define our own new ϕ's coefficients, our own forward and backward variables as follows

$$\phi_{k:l|n}(c_k, \dots, l) = P(C(t_k) = c_k, \dots, C(t_l) = c_l \mid S(t_0) = s_0, \dots, S(t_n) = s_n)$$

$$\alpha_k(c) = P(S(t_0) = s_0, \dots, S(t_k) = s_k, C(t_k) = c_k)$$

$$\beta_k(c) = P(S(t_0) = s_0, \dots, S(t_k) = s_k \mid C(t_k) = c_k).$$

So the transforms suggested above allow us to transform the parameter p_{c_{k-1},c_k} into $P_{d_{k-1}}(c_{k-1}, c_k)$ and finally by analogy, we can derive our recursions:

$$\alpha_{\nu,k}(c) = \sum_{c'} \alpha_{\nu,k-1}(c')\, P_{d_{k-1}}(c', c)\, g_{c,s_k}$$

$$\beta_{k|n}(c) = \sum_{c'} P_{d_k}(c, c')\, g_{c',s_{k+1}} \beta_{k+1|n}(c')$$

4.2.5 The continuous-time version of the Baum-Welch algorithm

Now let's get back to the expression of the likelihood

$$L(\{s_0^n\},\{c_0^n\};\theta) = \nu'_{c_0}\prod_{i=0}^{n} g_{c_i,s_i}\prod_{k=0}^{n-1} P_{d_k}(c_k,c_{k+1})$$

From the previous expression, we can then compute the log-likelihood :

$$Log(L(\{s_0^n\},\{c_0^n\};\theta)) = log(P(C(t_0)=c_0))\sum_{i=0}^{n} log(g_{c_i,s_i}) + \sum_{k=0}^{n-1} log(P_{d_k}(c_k,c_{k+1}))$$

When calculating the intermediate quantity $Q(\theta;\theta')$ of EM we have

$$\begin{aligned} Q(\theta;\theta') &= E_\theta[logL(\{s_0^n\},\{c_0^n\}) \mid S(t_0)=s_0,\ldots,S(t_n)=s_n] \\ &= \sum_{c_0} log(P(C(t_0)=c_0))\phi_0(c_0;\theta') + \sum_{i=0}^{n}\sum_{c_i} log(g_{c_i,s_i})\phi_i(c_i;\theta') \\ &\quad + \sum_{k=0}^{n-1}\sum_{c_k,c_{k+1}} log(P_{d_k}(c_k,c_{k+1}))\phi_{k,k+1}(c_k,c_{k+1};\theta') \end{aligned}$$

We can here again use T_1, T_2 and T_3 to denote the three distinct sums in $Q(\theta;\theta')$. Note that we have the exact same first two terms as in the discrete case and thus the re-estimates for ν' and G have exactly the same expression as ν and G for the discrete case. Let's focus on the third term T_3 and considering the two hidden states we write :

$$\begin{aligned} T_3 &= \sum_{i=0}^{n-1}\sum_{k,l} log(P_{d_i}(k,l))\phi_{i,i+1}(k,l;\theta') \\ &= \sum_{i=0}^{n-1} \phi_{i,i+1}(1,1)log(P_{d_i}(1,1)) + \phi_{i,i+1}(1,2)log(P_{d_i}(1,2)) + \phi_{i,i+1}(2,1)log(P_{d_i}(2,1)) \\ &\quad + \phi_{i,i+1}(2,2)log(P_{d_i}(2,2)) \end{aligned}$$

During a maximisation step of the EM algorithm we will have to find the optimal values for a and b such that T_3 is maximum. We therefore need to solve, for each iteration of the algorithm, the following system of constrained equations :

$$\begin{cases} \partial T_3/\partial a = 0 \\ \partial T_3/\partial b = 0 \\ a > 0 \\ b > 0 \end{cases}$$

Therefore the first thing we must do is to calculate the first derivatives of $P_t(i,j)$ with respect to a and b. Here are the results I found :

$$\begin{cases} \partial P_t(1,1)/\partial a = [1 - ta - \frac{a}{a+b}]\frac{e^{-(a+b)t}}{a+b} - \frac{b}{(a+b)^2} \\ \partial P_t(1,1)/\partial b = [-\frac{a}{a+b} - ta]\frac{e^{-(a+b)t}}{a+b} - \frac{b}{(a+b)^2} + \frac{1}{a+b} \end{cases}$$

$$\begin{cases} \partial P_t(2,2)/\partial a = [-\frac{b}{a+b} - tb]\frac{e^{-(a+b)t}}{a+b} - \frac{a}{(a+b)^2} + \frac{1}{a+b} \\ \partial P_t(2,2)/\partial b = [1 - tb - \frac{b}{a+b}]\frac{e^{-(a+b)t}}{a+b} - \frac{a}{(a+b)^2} \end{cases}$$

$$\begin{cases} \partial P_t(1,2)/\partial a = [ta + \frac{a}{a+b} - 1]\frac{e^{-(a+b)t}}{a+b} - \frac{a}{(a+b)^2} + \frac{1}{(a+b)} \\ \partial P_t(1,2)/\partial b = [\frac{a}{a+b} + ta]\frac{e^{-(a+b)t}}{a+b} - \frac{a}{(a+b)^2} \end{cases}$$

$$\begin{cases} \partial P_t(2,1)/\partial a = [tb + \frac{b}{a+b}]\frac{e^{-(a+b)t}}{a+b} - \frac{b}{(a+b)^2} \\ \partial P_t(2,1)/\partial b = [tb + \frac{b}{a+b} - 1]\frac{e^{-(a+b)t}}{a+b} - \frac{b}{(a+b)^2} + \frac{1}{(a+b)} \end{cases}$$

Finally we can fully express the previous system of constrained equations with a and b, thus we obtain these huge equations :

- With respect to a

$$\begin{aligned} \sum_{i=0}^{n-1} \phi_{i,i+1}(1,1) * \frac{[1 - ta - \frac{a}{a+b}]\frac{e^{-(a+b)t}}{a+b} - \frac{b}{(a+b)^2}}{\frac{1}{a+b}(b + ae^{-(a+b)t})} \\ + \phi_{i,i+1}(1,2) * \frac{[ta + \frac{a}{a+b} - 1]\frac{e^{-(a+b)t}}{a+b} - \frac{a}{(a+b)^2} + \frac{1}{(a+b)}}{\frac{1}{a+b}(a - ae^{-(a+b)t})} \\ + \phi_{i,i+1}(2,1) * \frac{[tb + \frac{b}{a+b}]\frac{e^{-(a+b)t}}{a+b} - \frac{b}{(a+b)^2}}{\frac{1}{a+b}(b - be^{-(a+b)t})} \\ + \phi_{i,i+1}(2,2) * \frac{[-\frac{b}{a+b} - tb]\frac{e^{-(a+b)t}}{a+b} - \frac{a}{(a+b)^2} + \frac{1}{a+b}}{\frac{1}{a+b}(a + be^{-(a+b)t})} \\ = 0 \end{aligned} \tag{6}$$

- with respect to b

$$\begin{aligned} \sum_{i=0}^{n-1} \phi_{i,i+1}(1,1) * \frac{[-\frac{a}{a+b} - ta]\frac{e^{-(a+b)t}}{a+b} - \frac{b}{(a+b)^2} + \frac{1}{a+b}}{\frac{1}{a+b}(b + ae^{-(a+b)t})} \\ + \phi_{i,i+1}(1,2) * \frac{[\frac{a}{a+b} + ta]\frac{e^{-(a+b)t}}{a+b} - \frac{a}{(a+b)^2}}{\frac{1}{a+b}(a - ae^{-(a+b)t})} \\ + \phi_{i,i+1}(2,1) * \frac{[tb + \frac{b}{a+b} - 1]\frac{e^{-(a+b)t}}{a+b} - \frac{b}{(a+b)^2} + \frac{1}{(a+b)}}{\frac{1}{a+b}(b - be^{-(a+b)t})} \\ + \phi_{i,i+1}(2,2) * \frac{[1 - tb - \frac{b}{a+b}]\frac{e^{-(a+b)t}}{a+b} - \frac{a}{(a+b)^2}}{\frac{1}{a+b}(a + be^{-(a+b)t})} \\ = 0 \end{aligned} \tag{7}$$

Using the analogy presented in the previous subsection, we can express $\phi_{i,i+1}(c_i, c_{i+1})$ as

$$\phi_{i,i+1}(c_i, c_{i+1}) = \frac{1}{L(s_0^n)} P_{d_i}(c_i, c_{i+1}) \alpha_i(c_i) g_{c_{i+1}, s_{i+1}} \beta_{i+1}(c_{i+1}).$$

Note that the computation of $\phi_{i,i+1}(c_i, c_{i+1})$ requires the computation of $L(s_0^n)$ which is intractable, but in fact we don't need to because in the previous equations we can simplify them and it only remains the numerators $P_{d_i}(c_i, c_{i+1}) \alpha_i(c_i) g_{c_{i+1}, s_{i+1}} \beta_{i+1}(c_{i+1})$. Of course, this is unsolvable by hand and so we need a mathematical Software as MATLAB or Mathematica to solve this system of equations.

Proposition 4.8 *During a M-step, the optimal re-estimates for parameters ν', G, and Q with two hidden states are*

$$\hat{\nu'}(j) = \frac{\alpha_0(j)\beta_0(j)}{\sum_l \alpha_0(l)\beta_0(l)}$$

$$\hat{g}_{js} = \frac{\sum_{i=0}^{n} \delta_{s_i,s}\alpha_i(j)\beta_i(j)}{\sum_{i=0}^{n} \alpha_i(j)\beta_i(j)}$$

$$(\hat{a};\hat{b}) = \textit{solution of} \begin{cases} (6) \\ (7) \\ a > 0 \\ b > 0 \end{cases} \quad \textit{with} \quad Q = \begin{pmatrix} -\hat{a} & \hat{a} \\ \hat{b} & -\hat{b} \end{pmatrix}$$

Finally when the optimal ν' is found, we can compute the vector ν using the formula:

$$\hat{\nu} = \hat{\nu'}\boldsymbol{P_{t_0}}^{-1} = \nu' e^{-t_0\hat{Q}}$$

4.2.6 Generalization to n states

So far we have only considered 2 hidden states and so Q was a square matrix of size 2. If we now assume that there are exactly r states, for $r \geq 3$, can we adopt the previous demonstration? It turns out that it is not so simple. Indeed, if we now assume r hidden states, then Q is a r-by-r matrix with exactly $r^2 - r = r(r-1)$ parameters to determine (recall that the diagonal elements q_{ii} are equal to $-\sum_{1\leq j\leq r, j\neq i} q_{ij}$), hence $r(r-1)$ equations (one equation for each derivative). The main problem we are confronted to when assuming r states is that we would have to find the roots of a r-order polynomial in both methods presented earlier on, namely the diagonalization and the Laplace transform methods. Indeed, concerning the diagonalization method, we need to find the eigenvalues of Q and therefore this means finding the roots of the characteristic polynomial $det(\lambda I - Q)$ which is a r-order polynomial. On the other hand, concerning the Laplace transform, we need to find the roots of $det(sI - Q)$ to be able to apply the partial-fraction decomposition. In either case we can't find the roots of a r-order polynomial and moreover we would have to find a symbolic expression of the roots according to the parameters q_{ij} which is a very complicated problem. The least we can do is to introduce the roots $a_1, \ldots, a_r$ of the polynomial we are considering, and by expanding the polynomial $(X - a_1)\ldots(X - a_r)$, we can identify each term w.r.t q_{ij}'s and finally we can formulate a system of r equations. This is still a difficult problem to solve it but I invite the reader to consult a paper from my supervisor [33] dealing with a similar problem with simpler assumptions that he solved numerically using MATHEMATICA.

4.3 Implementation and results

As said earlier, we are going to use a time-stamped trace that I got from my supervisor Peter Harrison. As seen in section 2, before using this trace, his group first processed this data trace by splitting the time axis into 5-millisecond bins so that they could use the HMM results derived in section 2 for the discrete case. However I don't need to pre-process it as my theory takes the continuous time into account. This time-stamped trace is basically just composed of two columns, the first one representing the observation times (in microseconds) and the second one representing the observation types (reads (r) or writes (w)). To apply my previous algorithm I just modified this trace such that a read instruction (r) correspond to a '1' and a write instruction (w) corresponds to a '2'. My final data traces (for the first 10 points) looks like:

Observation instants (μs)	Instruction type
1	0
1514	2
2133	1
2142	1
2744	1
2748	2
2749	1
3727	1
3771	2

First we need to set our initial conditions for our parameters ν', Q and G. However, the initial conditions depend on the contextual information that we have about the concerned process. For instance, assume the initial hidden state is determined by a biased coin and we know that 'heads' are more likely to appear than 'tails', then clearly we should take this prior information into account in our model by setting our initial distribution such that the probability of the state associated with 'heads' is greater than the other state, for example $\nu = (3/4 \quad 1/4)$. Unfortunately I don't have such contextual information and this is the reason why I decided to apply an equiprobable distribution:

$$\nu' = (0.5 \quad 0.5) \qquad G = \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix}$$

The case of Q is a bit harder since there are no "usual" values for elements of a infinitesimal generator matrix. Unlike the discrete case where the entries of the one-step transition matrix P are bounded since they represent a probability, the elements of the Q-matrix are largely unscaled and are process-dependent. For instance a small value for a means that the first state c_0 is more likely to hold for a long time since it represents the parameter of the exponential distribution of a holding time in state c_0. Besides, in order to successfully implement the EM algorithm, we need to establish a STOP condition and in my algorithm I chose to stop the successive iterations when two successive estimates of G are distant by no more than $\epsilon = 10^{-5}$, that is we stop the while-loop at the nth iteration when $\|G_{n-1} - G_n\| < \epsilon$. My algorithm is attached as Appendix. Since a and b are unscaled I decided to try different initial values for a and b and the results are presented in the below table

	$b=1$	$b=50$	$b=100$
$a=1$	$\hat{\nu}=\begin{pmatrix}1.0000 & 0.0000\end{pmatrix}$ $\hat{G}=\begin{pmatrix}0.9672 & 0.0328\\ 0.0027 & 0.9973\end{pmatrix}$ $\hat{Q}=\begin{pmatrix}-0.2627 & 0.2627\\ 0.2790 & -0.2790\end{pmatrix}$ $Nits=85$	$\hat{\nu}=\begin{pmatrix}0.0005 & 0.9995\end{pmatrix}$ $\hat{G}=\begin{pmatrix}0.5162 & 0.4838\\ 0.9989 & 0.0011\end{pmatrix}$ $\hat{Q}=\begin{pmatrix}-1.5468 & 1.5468\\ 80.5316 & -80.5316\end{pmatrix}$ $Nits=24$	$\hat{\nu}=\begin{pmatrix}0.0054 & 0.9946\end{pmatrix}$ $\hat{G}=\begin{pmatrix}0.5207 & 0.4793\\ 0.9996 & 0.0004\end{pmatrix}$ $\hat{Q}=\begin{pmatrix}-1.1734 & 1.1734\\ 121.7691 & -121.7691\end{pmatrix}$ $Nits=17$
$a=50$	$\hat{\nu}=\begin{pmatrix}1.0000 & 0.0000\end{pmatrix}$ $\hat{G}=\begin{pmatrix}0.9990 & 0.0010\\ 0.5159 & 0.4841\end{pmatrix}$ $\hat{Q}=\begin{pmatrix}-69.7626 & 69.7626\\ 1.3781 & -1.3781\end{pmatrix}$ $Nits=15$	$\hat{\nu}=\begin{pmatrix}1.0000 & 0.0000\end{pmatrix}$ $\hat{G}=\begin{pmatrix}0.9754 & 0.0246\\ 0.2298 & 0.7702\end{pmatrix}$ $\hat{Q}=\begin{pmatrix}-176.6577 & 176.6577\\ 116.3151 & -116.3151\end{pmatrix}$ $Nits=173$	$\hat{\nu}=\begin{pmatrix}1.0000 & 0.0000\end{pmatrix}$ $\hat{G}=\begin{pmatrix}0.9673 & 0.0327\\ 0.1008 & 0.8992\end{pmatrix}$ $\hat{Q}=\begin{pmatrix}-189.1350 & 189.1350\\ 182.7590 & -182.7590\end{pmatrix}$ $Nits=245$
$a=100$	$\hat{\nu}=\begin{pmatrix}0.9994 & 0.0006\end{pmatrix}$ $\hat{G}=\begin{pmatrix}0.9995 & 0.0005\\ 0.5206 & 0.4794\end{pmatrix}$ $\hat{Q}=\begin{pmatrix}-106.3692 & 106.3692\\ 1.0556 & -1.0556\end{pmatrix}$ $Nits=10$	$\hat{\nu}=\begin{pmatrix}1.0000 & 0.0000\end{pmatrix}$ $\hat{G}=\begin{pmatrix}0.9811 & 0.0189\\ 0.3420 & 0.6580\end{pmatrix}$ $\hat{Q}=\begin{pmatrix}-194.6988 & 194.6988\\ 78.4430 & -78.4430\end{pmatrix}$ $Nits=117$	$\hat{\nu}=\begin{pmatrix}1.0000 & 0.0000\end{pmatrix}$ $\hat{G}=\begin{pmatrix}0.9746 & 0.0254\\ 0.2155 & 0.7845\end{pmatrix}$ $\hat{Q}=\begin{pmatrix}-201.5140 & 201.5140\\ 139.3565 & -139.3565\end{pmatrix}$ $Nits=176$

In this table we have displayed the parameters we want to estimate along with the number of iterations $Nits$ before convergence. From this table we can notice that we know what the initial state is with a probability greater than 0.95 in every case. Also we remark an important dependence between the initial values of a and b, and the final estimates we get. The rate of convergence $Nits$ is also affected by the initial parameters because changing the initial values of a and b may make us move to another initial point of $T_3(a,b)$ where the gradient is higher and therefore the convergence is longer. Furthermore when

we change the initial values of a and b, we also affect the initial values of the forward and backward variables and therefore we change all values of all estimates: this is the reason why even we change the value of a or b by a small amount, we may end up with totally different estimates.

Note 4.9 In section 3.3, we also assumed the doubly stochastic process to be a Markov-modulated Poisson process. The current rate of this MMPP is determined by $\{C(t), t \geq 0\}$, in other words there are only two rates λ_0 and λ_1 (respectively associated with hidden states c_0 and c_1). Besides, we know that an inter-arrival time of a MMPP follows the *hyperexponential random distribution* $f(t) = \sum_{i=1}^{r} \pi_i \lambda_i e^{-\lambda_i t}$, with $\boldsymbol{\pi} = (\pi_1 \quad \pi_2)$ the steady-state vector of $\{C(t), t \geq 0\}$. Moreover we know that we can obtain $\boldsymbol{\pi}$ from the Q-matrix solving the constrained equations:

$$\begin{cases} \boldsymbol{\pi} Q = 0 \\ \boldsymbol{\pi} \cdot \mathbf{1} = 1 \end{cases}$$

Once we obtain $\boldsymbol{\pi}$, we can apply another EM algorithm to determine the MLEs for λ_0 and λ_1. But we can also proceed otherwise and use an approximation to determine λ_0 and λ_1. Indeed, let's pick out $\hat{G}$ and $\hat{Q}$ form the central cell of the previous table (case when initial parameters are a=50 and b=50):

$$\hat{G} = \begin{pmatrix} 0.9754 & 0.0246 \\ 0.2298 & 0.7702 \end{pmatrix} \qquad \hat{Q} = \begin{pmatrix} -176.6577 & 176.6577 \\ 116.3151 & -116.3151 \end{pmatrix}$$

By solving the previous system of equations we find $\boldsymbol{\pi} = (0.3970 \quad 0.6030)$. Here is the approximation: if we denote T the total time of the trace, n_r the number of reads and n_w the number of writes, then we can approximate the number of reads par unit of time n_r/T and the number of writes per unit of time n_w/T using:

$$\begin{cases} \frac{n_r}{T} = \sum_{j=1}^{2} \pi_j g_{j,1} \lambda_j \\ \frac{n_w}{T} = \sum_{j=1}^{2} \pi_j g_{j,2} \lambda_j \end{cases}$$

This is a simple system of equations for which we find $\boldsymbol{\lambda} = (\lambda_0 = 0.0013 \quad \lambda_1 = 0.0012)$.

5 Conclusion and future work

In this paper we have described a new methodology to adapt the use of Hidden Markov models to continuous-time processes. We have firstly introduced HMMs as an efficient tool allowing us to represent real-life system faithfully and parsimoniously. Then we have discussed the three canonical issues related to HMMs and we have shown how to solve them in a tractable way: this gave birth to the three well known algorithm that are the Forward-Backward algorithm, the Viterbi algorithm and the Baum-Welch algorithm. In the second part we have discussed a successful application of HMMs to Flash memory partly carried out by my supervisor Peter Harrison. Although the results were surprisingly satisfying, especially when comparing simple metrics as mean count and its standard deviation, Peter wanted to improve the matching between the ACFs. From this point we have considered a continuous-time bivariate stochastic process playing the role of a Hidden Markov model at arrival times and we needed to make some further assumption: this process was assumed to be a Markov-modulated Poisson process. The next natural step was to derive the new parameters of this bivariate process, and in order to reach such a goal, we have adapted the discrete-case theory to continuous time. In particular we have found a new way to compute

the parameter re-estimates in a very simple context where $\{C(t)\}$ switched between two states only over the time.

We can extend this work in many directions but one of the most imperative one is to generate new traces using the model parameters we have found. Then we could compare the raw trace with the newly generated one using some specific metrics as the mean count of reads and writes, the number of consecutive reads and writes, etc. We can also focus on taking into account more than 2 hidden states but this seems to be a harder problem. Finally we could create a model with more observation states using the sizes of instructions and for instance we could create a 4-state observation space with large-sized/small-sized reads/writes.

References

[1] P. G. Harrison, S. K. Leyton, N. M. Patel and S. Zertal, *Storage Workload Modelling by Hidden Markov Models: Application to Flash Memory*, 2010.

[2] Susanna Wau Men Au-Yeung, P. G. Harrison and William J. Knottenbelt, *A queueing Network Model of Patient Flow in an Accident and Emergency Department*, 2006.

[3] S. K. Leyton, *Statistical Inference in Hidden Markov Models*, 2008.

[4] L. R. Rabiner, *A tutorial on hidden Markov models and selected applications in speech recognition*, Proc. IEEE, 77:257-285, 1989.

[5] Sean R. Eddy, *Hidden Markov models* - protein structures, Current Opinion in Structural Biology, 1996.

[6] Olivier Cappe, Eric Moulines, and Tobias Ryden. *Inference in Hidden Markov Models.* Springer Series in Statistics. Springer, 2005.

[7] Iain L. MacDonald and Walter Zucchini. *Hidden Markov and other Models for discrete-valued Time Series.*Monographs on Statistics and Applied Probability. Chapman & Hall/CRC, 1997.

[8] Albert, P.S. *A two-state Markov mixture model for a time series of epileptic seizure counts.* Biometrics **47**, 1371-1381, 1991.

[9] Albert, P.S. *A Markov model for sequences of ordinal data from a relapsing* Biometrics **50**, 51-60, 1991.

[10] J. Grandell, *Doubly stochastic Poisson Processes.* Springer, Berlin, 1976.

[11] H.Heffes and D.M. Lucantoni, *A Markov modulated characterization of packetized voice and data traffic and related statistical multiplexer performance,* IEEE J. Selected Areas Comm. **4**(6), 856-868, 1986.

[12] J.J. Hunter, *On the moments of Markov renewal Poisson processes,* Adv. Appl. Probab. **1**, 188-210, 1969.

[13] J.F.C. Kingman, *On doubly stochastic Poisson processes,* Roy. Cambridge Phil. Soc. **60**, 923-960, 1964.

[14] D.M. Lucantoni and V. Ramaswami, *Efficient Algorithms for solving the non-linear matrix equations arising in phase type queues,* Comm. Statist. Stochastic Models **1**, 29-52, 1985.

[15] K.S. Meier, *A fitiing algorithm for Markov-modulated Poisson processes having two arrival rates,* European J. Oper. Res. **29**, 370-377, 1987.

[16] A. Azzalini, *Maximum likelihood estimation of order m for stationary stochastic processes.* Biometrika **70**, 381-387, 1983.

[17] A. Azzalini and A.W. Bowman, *A look at some data on the Old Faithful geyser.* Appl. Satist. **39**, 357-365, 1990.

[18] P. Baldi, Y. Chauvin, T. Haunkapiller and M.A. McClure, emphHidden Markov models of biological primary sequence information. PROC. Nat. Acad. Sci. U.S.A. **91**, 1059-1063, 1994.

[19] F.G. Ball and J.A. Rice *Statistical Inference for Stochastic Processes.* Academic Press, London, 1992.

[20] L.E. Baum, *An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes.* PROC. Third Symposium on Inequalities, ed. O. Shisha. Academic Press, New York, 1-8, 1972.

[21] L.E. Baum and T. Petrie, *Statistical inference for probabilistic functions of finite state Markov chains.* Ann. Math. Statist. **37**, 1554-1563, 1966.

[22] P. Billingsley, *Statistical methods in Markov chains.* Ann. Math. Statist. **32**, 12-40, 1961.

[23] S. Bisgaard and L.E. Travis, *Existence and uniqueness of the solution of the likelihood equations for binary Markov chains.* Statist. Prob. Letters **12**, 29-35, 1991.

[24] C. Chatfield, *Statistical inference regarding Markov chain models.* Appl. Statist. **22**, 7-20, 1973.

[25] D.R. Cox and P.A.W Lewis, *The Statistical Analysis of Series of Events.* Methuen, London, 1966.

[26] D.R. Cox and H.D. Miller, *The Theory of Stochastic Processes.* Chapman&Hall, London, 1965.

[27] D.R. Cox and E.J. Snell, *Analysis of Binary Data.* Second edition, Chapman&Hall, London, 1989.

[28] C. Moler and C. van Loan, *nineteen dubious ways to compute the exponential matrix.* SIAM Rev. **20**, 801-836, 1978.

[29] C.C. Paige, G.P.H. Styan and P.G. Wachter, *Computation of the stationary distribution of a Markov chain.* J. Statist. Comput. Simulation **4**, 173-186, 1975.

[30] R. Pyke, *Markov renewal processes: definitions and preliminary properties.* Ann. Math. Statist/ **32**, 1231-1242, 1961.

[31] V. Ramaswami, *Stable recursion for the steady state vector for Markov chains of $M/G/1$ type.* Comm. Statist. Stochastic Models **4**, 183-188, 1988.

[32] B. Sengupta, *Markov processes whose steady state distribution is matrix-exponential with an application to the GI/PH/1 queue.* Adv. Appl. Probab. **21**, 159-180, 1986.

[33] A.J. Field and P.G. Harrison, *Busy Periods in Fluid Queues with Multiple Emptying Input States.* Journal of Applied Probability, 2010, *to appear.*

Appendix

```
function res = BaumWelch(x0)

N=100;                                              %Number of points
S = textread('ProcessedTrace.txt');
S = S(1:N,1:2);
Times = S(:,1);
Obs = S(:,2);

%%%%%% initialization of parameters %%%%%%%%%%%%%%
a=x0(1);
b=x0(2);
Q = [[-a a];[b -b]];
nu = [1/2 1/2];
G = [nu ; nu];
alpha = ones(N,2);
beta = alpha;
phi = zeros(N-1,4);
Gmem = 0;
D=zeros(N,1);
options = optimset('LargeScale','off','MaxFunEvals', 5000);

%%%%%% Baum-Welch algorithm
Nits = 0;

while (norm(G-Gmem)>10^(-5) || Nits == 0)   %%%% STOP condition

      Gmem = G

    %*********** Update the alphas *************%

    alpha(1,:) = [nu(1)*G(1,Obs(1)) nu(2)*G(2,Obs(1))];
    alpha(1,:) = alpha(1,:)/sum(alpha(1,:));
    for t= 2:N
        E = expm((Times(t)-Times(t-1))*Q);
        for j=1:2
            alpha(t,j) = G(j,Obs(t))*E(1,j)*alpha(t-1,1) + G(j,Obs(t))*E(2,j)*alpha(t-
        end
        D(t) = sum(alpha(t,:));
        alpha(t,:) = alpha(t,:)/D(t);
    end

     %*********** Update the betas *************%

    for t=1:N-1
        t1 = N-t;
            E = expm((Times(t1+1)-Times(t1))*Q);
        for j=1:2
```

```
        beta(t1,j) = G(1,Obs(t1+1))*E(j,1)*beta(t1+1,1) + G(2,Obs(t1+1))*E(j,2)*bet
    end
    beta(t1,:) = beta(t1,:)/D(t1+1);
end
beta;

 %*********** Update the phis *************%

for t=1:N-1
    E = expm((Times(t+1)-Times(t))*Q);
    phi(t,1) = E(1,1)*alpha(t,1)*G(1,Obs(t+1))*beta(t+1,1)/D(t+1);
    phi(t,2) = E(1,2)*alpha(t,1)*G(2,Obs(t+1))*beta(t+1,2)/D(t+1);
    phi(t,3) = E(2,1)*alpha(t,2)*G(1,Obs(t+1))*beta(t+1,1)/D(t+1);
    phi(t,4) = E(2,2)*alpha(t,2)*G(2,Obs(t+1))*beta(t+1,2)/D(t+1);
end
phi;

%\\\\\\\\\\\\\\\\\\\\\\\\Update the parameters ////////////////////////%

%***************Updating the initial distribution nu ******************%

nu(1) = alpha(1,1)*beta(1,1) / (alpha(1,1)*beta(1,1) + alpha(1,2)*beta(1,2)) ;
nu(2) = alpha(1,2)*beta(1,2) / (alpha(1,1)*beta(1,1) + alpha(1,2)*beta(1,2)) ;

%*************** Updating the G matrix ****************************%

for i=1:2
    for j = 1:2
        temp1 = 0;
        temp2 = 0;
        for k = 1:N
            if (Obs(k)) == j
               temp1 = temp1 + alpha(k,i)*beta(k,i);
            end
               temp2 = temp2 + alpha(k,i)*beta(k,i);
        end
        G(i,j) = temp1/temp2;
    end
end

%*************** Updating the Generator matrix Q******************%


s = fsolve(@myfun, [a b], options)
a=s(1);
b=s(2);
Q = [[-a a];[b -b]]
```

```
    Nits = Nits + 1

end

    %**************** Definition of Equations*******************%
    function F = myfun(x)
        F1 = 0;
        F2 = 0;
        for i=1:N-1
            d = Times(i+1)-Times(i);
            e = exp(-(x(1)+x(2))*d);
            F1 = F1 + phi(i,1) * ((1-d*x(1)- x(1)/(x(1)+x(2)))*e - x(2)/(x(1)+x(2)))/
                            phi(i,2) * ((d*x(1)-1+ x(1)/(x(1)+x(2)))*e - x(1)/(x(1)+x
                            phi(i,3)*((d*x(2)+ x(2)/(x(1)+x(2)))*e - x(2)/(x(1)+x(2)
                            phi(i,4)*((-d*x(2)- x(2)/(x(1)+x(2)))*e - x(1)/(x(1)+x(2
            F2 = F2 + phi(i,1) * ((-x(1)/(x(1)+x(2))-d*x(1))*e - x(2)/(x(1)+x(2))+1)/
                            phi(i,2) * ((d*x(1)+ x(1)/(x(1)+x(2)))*e - x(1)/(x(1)+x(
                            phi(i,3)*((d*x(2)+ x(2)/(x(1)+x(2))-1)*e - x(2)/(x(1)+x(
                            phi(i,4)*((1-x(2)/(x(1)+x(2))-d*x(2))*e - x(1)/(x(1)+x(2

        end
        F=[F1 F2];

    end
end
```