An HMM Implementation for Change Detection in Semiconductor Manufacturing
ECSE 6810: Probabilistic Graphical Models
Andrés Vargas

1 Introduction

Currently, there are billions of smartphones in existence around the world. Each of these smartphones houses within it a number of chips that drive the apps that allow us to text, surf the web, count calories, play games, and do thousands of other things. Each of these chips is composed of number of very thin (on the nano meter scale) wafers that need to be manufactured very precisely in order for the chip to be able to carry out its function. Thus, it is very important that the companies that manufacture these wafers do so in a way that minimizes the probability that a wafer will be defective.

The wafers are manufactured in bulk via an automated process that occurs inside special chambers. Changes in the state of the chamber will affect the quality of the resultant wafers, and so we would like to notify the supervising engineers whenever the chamber state has changed so that proactive maintenance can be performed on the ailed chamber. The problem is that "chamber state" is a loosely defined concept. How can we hope do detect changes in something that we don't even know how to define? The solution comes from the fact that each chamber is equipped with sensors that collect signals in real time. It doesn't seem at all far fetched to assume that these signals are correlated with the state of the chamber. Thus, although we cannot directly observe the state of the chamber, we can use the emitted signals to estimate the probability that, at any given time, the state of the chamber will be different from its state at the previous time.

Figure 1 shows a bayesian network representation of the proposed schema. Bayesian networks of this form are called hidden markov models, or HMM's. There are harpoons above the names of the observed variables in Figure 1 to emphasize the fact that they are vectors, since there are multiple signals emitted at each time t. However, in this particular situation, the emissions at each time slice have an interesting semantic meaning: they are the parameters of a 2-dimensional damped harmonic oscillator, and we assume they are statistically independent. Under the independence assumption, the actual schema is more precisely represented by Figure 2. This assumption also significantly reduces the complexity of parameter learning because we only need to learn D variances for each hidden state, as opposed to the $\frac{D(D-1)}{2}$ covariances that would be required without it, where D is the number of sensor measurements emitted at each time slice.

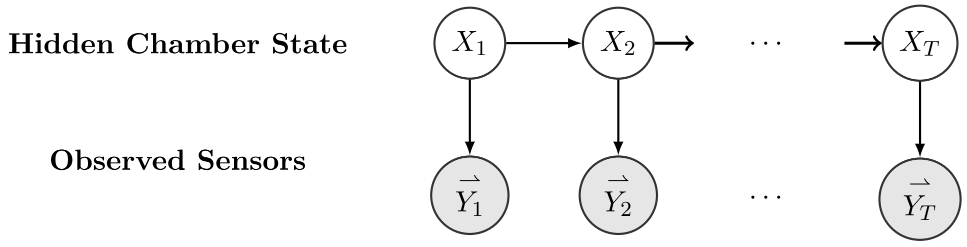

Figure 1: HMM schema without independence assumption

Given N change points for some chamber, I will learn the parameters $\lambda = \{\pi, A, B\}$ for a hidden markov model, where $A \in \mathbb{R}^{N\times N}$ is the transition CPT, $B \in \mathbb{R}^{T\times N}$ is the emission probabilities, and $\pi \in \mathbb{R}^N$ is the prior probabilities. The parameters will be learned using a real-life semiconductor manufacturing data set, and I will then evaluate the model on the same data set. The hope is that the learned model will be able to correctly identify points in time when a transition between two distinct hidden states occurs. The algorithm will be implemented in R.

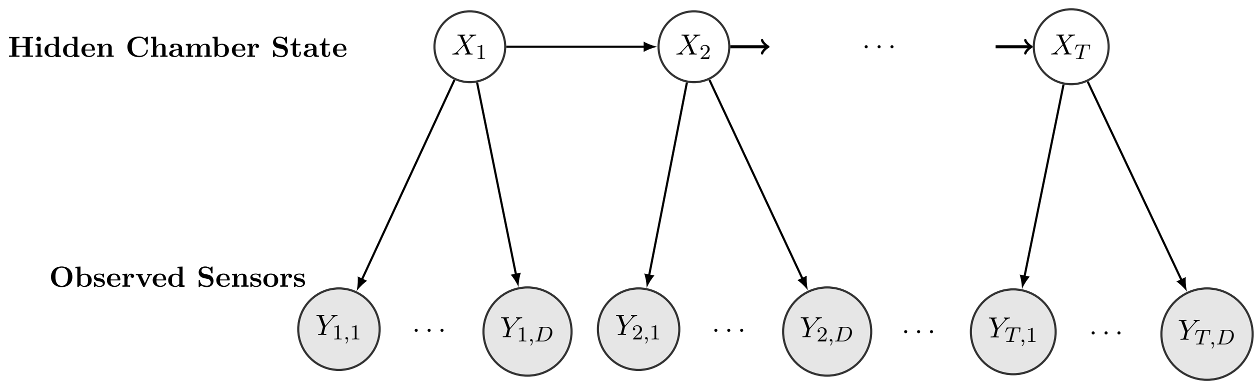

Figure 2: HMM schema with independent emissions at each time slice

2 Theory

In this section, I discuss the parameter learning process, followed by a discussion on how to use the learned parameters for change detection. Lastly, I discuss a method for dealing with underflow.

2.1 Parameter Learning

The Baum-Welch algorithm, which is a type of EM algorithm, is used to learn the parameters of hidden markov models. Section 2.1.1 derives the updates at each iteration and section 2.1.2 discusses termination conditions.

2.1.1 Re-estimating Parameters

Let $\boldsymbol{Y} = (\boldsymbol{Y}_1, ..., \boldsymbol{Y}_T)$ be the training data. Henceforth, I will write $\mathbb{P}(\boldsymbol{Y}_{t+1} = \boldsymbol{y}_{t+1})$ as $\mathbb{P}(\boldsymbol{y}_{t+1})$ for notational convenience. Let $\xi_t(i,j)$ be the probability of being in state i at time t and state j at time $t+1$, given the model λ. We can use the factorization property of bayesian networks to write

$$\begin{aligned}
\xi_t(i,j) = \mathbb{P}(X_t = i, X_{t+1} = j | \boldsymbol{Y}, \lambda) &= \frac{\mathbb{P}(X_t = i, X_{t+1} = j, \boldsymbol{Y} | \lambda)}{\sum_{i,j} \mathbb{P}(X_t = i, X_{t+1} = j, \boldsymbol{Y} | \lambda)} \\
&= \frac{\mathbb{P}(\boldsymbol{y}_1, ..., \boldsymbol{y}_t, X_t = i|\lambda)\mathbb{P}(X_{t+1} = j|X_t = i)\mathbb{P}(\boldsymbol{y}_{t+1}|X_{t+1})\mathbb{P}(\boldsymbol{y}_{t+2}, ..., \boldsymbol{y}_T|X_{t+1} = j, \lambda)}{\sum_{i,j} \mathbb{P}(\boldsymbol{y}_1, ..., \boldsymbol{y}_t, X_t = i|\lambda)\mathbb{P}(X_{t+1} = j|X_t = i)\mathbb{P}(\boldsymbol{y}_{t+1}|X_{t+1})\mathbb{P}(\boldsymbol{y}_{t+2}, ..., \boldsymbol{y}_T|X_{t+1} = j, \lambda)} \\
&= \frac{\alpha_t(i) a_{ij} b_j(\boldsymbol{y}_{t+1}) \beta_{t+1}(j)}{\sum_{i,j} \alpha_t(i) a_{ij} b_j(\boldsymbol{y}_{t+1}) \beta_{t+1}(j)} \ ,
\end{aligned}$$

where $\alpha_t(i) = \mathbb{P}(\boldsymbol{y}_1, ..., \boldsymbol{y}_t, X_t = i|\lambda)$ and $\beta_{t+1}(j) = \mathbb{P}(\boldsymbol{y}_{t+2}, ..., \boldsymbol{y}_T|X_{t+1} = j, \lambda)$ can be computed recursively as follows:

$$\alpha_{t+1}(j) = \sum_{i=1}^{N} \alpha_t(i)\mathbb{P}(X_{t+1} = j|X_t = i)\mathbb{P}(\boldsymbol{y}_{t+1}|X_{t+1} = j) = \sum_{i=1}^{N} \alpha_t(i) a_{ij} b_j(\boldsymbol{y}_{t+1}) \ ,$$

$$\beta_t(i) = \sum_{j=1}^{N} \mathbb{P}(X_{t+1} = j|X_t = i)\mathbb{P}(\boldsymbol{y}_{t+1}|X_{t+1} = j)\beta_{t+1}(j) = \sum_{j=1}^{N} a_{ij} b_j(\boldsymbol{y}_{t+1}) \beta_{t+1}(j) \ .$$

Here, a_{ij} is $\mathbb{P}(X_{t+1} = j|X_t = i)$, and $b_j(\boldsymbol{y}_{t+1})$ is $\mathbb{P}(\boldsymbol{y}_{t+1}|X_{t+1} = j)$. We initialize $\alpha_1(i) = \pi_i b_i(\boldsymbol{y}_1)$ and $\beta_T(i) = 1$ for each hidden state i. Note, in the summations, i and j go from 1 to N, where N is the number of hidden states.

Next, define $\gamma_t(i) = \mathbb{P}(X_t = i|\boldsymbol{Y}, \lambda)$. In other words, $\gamma_t(i)$ is the probability of being in state i at time t, given the observation sequence and λ. Thus, we can marginalize $\xi_t(i,j)$ over j to obtain $\gamma_t(i)$:

$$\gamma_t(i) = \sum_j \xi_t(i,j) \ .$$

Notice that we can write the expected number of transitions from state i, over all samples and given the observations and λ, as

$$E[\sum_{t=1}^{T-1} \mathbb{I}\{X_t = i|\boldsymbol{Y},\lambda\}] = \sum_{t=1}^{T-1} E[\mathbb{I}\{X_t = i|\boldsymbol{Y},\lambda\}] = \sum_{t=1}^{T-1} \mathbb{P}(X_t = i|\boldsymbol{Y}^m,\lambda) = \sum_{t=1}^{T-1} \gamma_t(i) \ .$$

In a similar fashion, we can write the expected number of transitions from state i to state j as $\sum_{t=1}^{T-1} \xi_t(i,j)$. We can use this intuition to come up with estimators for π_i and a_{ij}. π_i can be estimated by determining the number of times, out of all samples, that we expect to be in state i, i.e.

$$\hat{\pi}_i = \gamma_1(i) \ .$$

a_{ij} can be estimated by determining the number of times we expect to transition from state i to state j, divided by the number of times we expect to transition from state i, i.e.

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \ .$$

Estimating B is more complicated because the observations are continuous and multidimensional. Assume (perhaps brashly) that $\mathbb{P}(\boldsymbol{Y}_t|X_t = j) \sim \mathcal{N}(\mu_j, \Sigma_j)$. In other words, we assume that the probability of the observations at each time t, given the hidden state at t, is given by a multivariate gaussian distribution.

The updates are for the mean and covariance matrix are

$$\hat{\boldsymbol{\mu}}_j = \frac{\sum_{t=1}^{T} \gamma_t(j) \cdot \boldsymbol{Y}_t}{\sum_{t=1}^{T} \gamma_t(j)} \ ,$$

$$\hat{\boldsymbol{\Sigma}}_j = \frac{\sum_{t=1}^{T} \gamma_t(j) \cdot (\boldsymbol{Y}_t - \boldsymbol{\mu}_j)(\boldsymbol{Y}_t - \boldsymbol{\mu}_j)^*}{\sum_{t=1}^{T} \gamma_t(j)} \ , \forall j \in \{1,...,N\} \ .$$

The asterisk in the equation for $\hat{\boldsymbol{\Sigma}}_j$ stands for the transpose. We see that the numerator in the update for the mean is the weighted sum of the observations, with the each weight being the probability of being in state j for the respective sample and time. The purpose of the denominator is merely to standardize the sum in the numerator. If there were only one hidden state, then N would be 1 and we would have $\gamma_t(1) = 1 \ , \forall t$. We would would thus only need to estimate one mean, and the estimate would be $\hat{\boldsymbol{\mu}} = \frac{\sum_{t=1}^{T} \boldsymbol{Y}_t}{T}$, which is nothing but the maximum likelihood estimate for the mean. This same intuition can be used to explain the estimate for the covariance matrix. Liporace provides a derivation of the update equations for multivariate emissions in [2], and Rabiner provides a lagrange multiplier derivation of the update equations for a_{ij} and π_i in [1].

Recall from section 1 that we assumed that the signals at each time slice are independent. This means that, in our situation, $\boldsymbol{\Sigma}_j$ is a diagonal matrix with the elements of the diagonal representing the variances of the corresponding sensors. Thus, we need only estimate D variances, and the updates become

$$\hat{\sigma}_{dj}^2 = \frac{\sum_{t=1}^{T} \gamma_t(j)(Y_{dj} - \mu_{dj})^2}{\sum_{t=1}^{T} \gamma_t(j)}, \ \forall d \in \{1,...,D\} \ , \forall j \in \{1,...,N\} \ .$$

2.1.2 Termination Conditions

The algorithm terminates when $LLnew - LLold < \epsilon$, where $LLnew$ is the most current log-likelihood and $LLold$ is the immediately previous log-likelihood. The log-likelihood can be written as

$$\log \mathbb{P}(\boldsymbol{Y}|\lambda) = \log \sum_{j=1}^{N} \mathbb{P}(\boldsymbol{y}_{1:T}, X_T = j|\lambda) = \log \sum_{j=1}^{N} \alpha_T(j) \ .$$

Thus, we can reuse the $\alpha_T(i)$ from the two most recent iterations to determine whether or not the algorithm should terminate.

2.2 Change Detection and Model Evaluation

Once the parameters λ^* have been learned, we want to evaluate the extent to which they can detect changes in the hidden state. The probability that the hidden state changes between time t and $t+1$ is

$$\sum_{i \neq j} \mathbb{P}(X_t = i, X_{t+1} = j|\boldsymbol{Y}, \lambda^*) = \sum_{i \neq j} \xi_t(i,j) \ .$$

To evaluate the performance of the model, I sorted the times by their probabilities of coinciding with a state change, and then used visualizations to compare the most probable change times with the actual change times.

2.3 Underflow

Using the equations from section 2.1.1 leads to underflow, especially when T is large and observations are multivariate (as so happens to be the case with this project), because then the recursion will require the product of a large number of very small probabilities. To combat this issue, scale both $\alpha_t(i)$ and $\beta_t(i)$ by a quantity c_t to obtain $\bar{\alpha}_t(i)$ and $\bar{\beta}_t(i)$, and then store c_t, for all $t \in \{1, ..., T\}$. At the end of the recursion, the stored scaling factors c_t can be used to recover the parameter updates, log-likelihoods, and change detection probabilities. This scaling procedure was originally proposed by Rabiner[1], but I found the paper by Shen[2] exceedingly useful in clearing up some of the technicalities in Rabiners original paper. First, let

$$\ddot{\alpha}_1(i) = \alpha_1(i) \ , \forall i \in \{1, ..., N\} \ .$$

Then, define

$$c_1 = \frac{1}{\sum_{i=1}^{N} \ddot{\alpha}_1(i)}$$

and

$$\bar{\alpha}_1(i) = c_1 \ddot{\alpha}_1(i) \ .$$

We update as follows:

$$\ddot{\alpha}_{t+1}(j) = \sum_{i=1}^{N} \bar{\alpha}_t(i) a_{ij} b_j(\boldsymbol{Y}_{t+1}) \ ,$$

$$c_{t+1} = \frac{1}{\sum_{i=1}^{N} \ddot{\alpha}_{t+1}(i)} \ ,$$

$$\bar{\alpha}_{t+1}(j) = c_{t+1} \ddot{\alpha}_{t+1}(j).$$

Note that this update scheme is recursive because

$$\bar{\alpha}_{t+1}(j) = c_{t+1} \ddot{\alpha}_{t+1}(j) = \frac{\ddot{\alpha}_{t+1}(j)}{\sum_{j=1}^{N} \ddot{\alpha}_{t+1}(j)} = \frac{\sum_{i=1}^{N} \bar{\alpha}_t(i) a_{ij} b_j(\boldsymbol{Y}_{t+1})}{\sum_{j=1}^{N} \sum_{i=1}^{N} \bar{\alpha}_t(i) a_{ij} b_j(\boldsymbol{Y}_{t+1})} \ .$$

It is worth reiterating that c_t should be stored at each recursive layer t. We can show that $\bar{\alpha}_t(i) = \left(\prod_{\tau=1}^{t} c_\tau\right) \alpha_t(i)$ by induction. For the base case (t=1), we have $\bar{\alpha}_1(i) = c_1\ddot{\alpha}_1(i) = \left(\prod_{\tau=1}^{1} c_\tau\right) \alpha_1(i)$. For the inductive step, assume that

$$\bar{\alpha}_k(j) = \left(\prod_{\tau=1}^{k} c_\tau\right) \alpha_k(j)$$

for some integer $k > 1$. From here it follows that

$$\begin{aligned}
\bar{\alpha}_{k+1}(j) &= c_{k+1}\ddot{\alpha}_{k+1}(j) \\
&= c_{k+1} \sum_{i=1}^{N} \bar{\alpha}_k(i) a_{ij} b_j(\boldsymbol{Y}_{k+1}) \\
&= c_{k+1} \sum_{i=1}^{N} \left(\prod_{\tau=1}^{k} c_\tau\right) \alpha_k(i) a_{ij} b_j(\boldsymbol{Y}_{k+1}) \\
&= \left(\prod_{\tau=1}^{k+1} c_\tau\right) \sum_{i=1}^{N} \alpha_k(i) a_{ij} b_j(\boldsymbol{Y}_{k+1}) \\
&= \left(\prod_{\tau=1}^{k+1} c_\tau\right) \alpha_{k+1} \ .
\end{aligned}$$

For the backward process, use the same scaling factors c_t that were computed in the forward process. Initialize $\ddot{\beta}_T(i) = 1$ and $\bar{\beta}_T(i) = c_T\ddot{\beta}_T(i)$. The recursive updates are

$$\ddot{\beta}_t(i) = \sum_{j=1}^{N} a_{ij} b_j(Y_{t+1}) \bar{\beta}_{t+1}(j) \ ,$$

$$\bar{\beta}_t(i) = c_t \ddot{\beta}_t(i) \ .$$

It can be shown that $\bar{\beta}_t(i) = \left(\prod_{s=t}^{T} c_s\right) \beta_t(i)$ via a similar inductive proof as the one for $\bar{\alpha}_t(i)$. Now, let $\boldsymbol{C}_t = \prod_{\tau=1}^{t} c_\tau$ and $\boldsymbol{D}_t = \prod_{s=t}^{T} c_s$. Note that $\boldsymbol{C}_t\boldsymbol{D}_{t+1} = \boldsymbol{C}_T$, $\boldsymbol{C}_t\boldsymbol{D}_t = \boldsymbol{C}_T c_t$, $\boldsymbol{C}_T = \boldsymbol{D}_1$, $\bar{\alpha}_t(i)/\boldsymbol{C}_t = \alpha_t(i)$, and $\bar{\beta}_t(i)/\boldsymbol{D}_t = \beta_t(i)$, $\forall t \in \{1, ..., T\}$, $\forall i \in \{1, ..., N\}$. Also observe that

$$\begin{aligned}
\gamma_t(i) = \sum_j \xi_t(i,j) = \frac{\sum_j \mathbb{P}(X_t = i, X_{t+1} = j, \boldsymbol{Y})}{\mathbb{P}(\boldsymbol{Y})} &= \frac{\mathbb{P}(X_t = i, \boldsymbol{Y})}{\mathbb{P}(\boldsymbol{Y})} = \frac{\mathbb{P}(\boldsymbol{Y}|X_t = i)\mathbb{P}(X_t = i)}{\mathbb{P}(\boldsymbol{Y})} \\
= \frac{\mathbb{P}(\boldsymbol{y}_{1:t} \ , \boldsymbol{y}_{(t+1):T}|X_t = i)\mathbb{P}(X_t = i)}{\mathbb{P}(\boldsymbol{Y})} &= \frac{\mathbb{P}(\boldsymbol{y}_{1:t}|X_t = i)\mathbb{P}(\boldsymbol{y}_{(t+1):T}|X_t = i)\mathbb{P}(X_t = i)}{\mathbb{P}(\boldsymbol{Y})} \\
= \frac{\mathbb{P}(\boldsymbol{y}_{1:t}, X_t = i)\mathbb{P}(\boldsymbol{y}_{(t+1):T}|X_t = i)}{\mathbb{P}(\boldsymbol{Y})} &= \frac{\alpha_t(i)\beta_t(i)}{\mathbb{P}(\boldsymbol{Y})} \ .
\end{aligned}$$

We can now write the parameter updates in terms of the newly defined quantities of this section:

$$\hat{\pi}_i = \gamma_1(i) = \frac{\alpha_1(i)\beta_1(i)}{\mathbb{P}(\boldsymbol{Y}|\lambda)} = \frac{\alpha_1(i)\beta_1(i)}{\sum_{i=1}^N \alpha_T(i)} = \frac{(\bar{\alpha}_1(i)/\boldsymbol{C}_1)(\bar{\beta}_1(i)/\boldsymbol{D}_1)}{\sum_{i=1}^N \bar{\alpha}_T(i)/\boldsymbol{C}_T} = \frac{\boldsymbol{C}_T}{c_1\boldsymbol{D}_1} \cdot \frac{\bar{\alpha}_1(i)\bar{\beta}_1(i)}{\sum_{i=1}^N \bar{\alpha}_T(i)} = \frac{1}{c_1} \cdot \frac{\bar{\alpha}_1(i)\bar{\beta}_1(i)}{\sum_{i=1}^N \bar{\alpha}_T(i)} \ ,$$

$$\begin{aligned}\hat{a}_{ij} &= \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} = \frac{\left(\sum_{t=1}^{T-1} \alpha_t(i)a_{ij}b_j(\boldsymbol{Y}_{t+1})\beta_{t+1}(j)\right)/\mathbb{P}(\boldsymbol{Y}|\lambda)}{\left(\sum_{t=1}^{T-1} \alpha_t(i)\beta_t(i)\right)/\mathbb{P}(\boldsymbol{Y}|\lambda)} = \frac{\sum_{t=1}^{T-1} \alpha_t(i)a_{ij}b_j(\boldsymbol{Y}_{t+1})\beta_{t+1}(j)}{\sum_{t=1}^{T-1} \alpha_t(i)\beta_t(i)} \\ &= \frac{\sum_{t=1}^{T-1}(\bar{\alpha}_t(i)a_{ij}b_j(\boldsymbol{Y}_{t+1})\bar{\beta}_{t+1}(j)/(\boldsymbol{C}_t\boldsymbol{D}_{t+1}))}{\sum_{t=1}^{T-1}(\bar{\alpha}_t(i)\bar{\beta}_t(i)/(\boldsymbol{C}_t\boldsymbol{D}_t))} = \frac{\left(\sum_{t=1}^{T-1} \bar{\alpha}_t(i)a_{ij}b_j(\boldsymbol{Y}_{t+1})\bar{\beta}_{t+1}(j)\right)/\boldsymbol{C}_T}{\left(\sum_{t=1}^{T-1}(\bar{\alpha}_t(i)\bar{\beta}_t(i)/c_t)\right)/\boldsymbol{C}_T} \\ &= \frac{\sum_{t=1}^{T-1} \bar{\alpha}_t(i)a_{ij}b_j(\boldsymbol{Y}_{t+1})\bar{\beta}_{t+1}(j)}{\sum_{t=1}^{T-1}(\bar{\alpha}_t(i)\bar{\beta}_t(i)/c_t)} \ ,\end{aligned}$$

$$\hat{\boldsymbol{\mu}}_j = \frac{\sum_{t=1}^T \gamma_t(j)\cdot\boldsymbol{Y}_t}{\sum_{t=1}^T \gamma_t(j)} = \frac{\sum_{t=1}^T \alpha_t(j)\beta_t(j)\cdot\boldsymbol{Y}_t/\mathbb{P}(\boldsymbol{Y}|\lambda)}{\sum_{t=1}^T \alpha_t(j)\beta_t(j)/\mathbb{P}(\boldsymbol{Y}|\lambda)} = \frac{\sum_{t=1}^T \alpha_t(j)\beta_t(j)\cdot\boldsymbol{Y}_t}{\sum_{t=1}^T \alpha_t(j)\beta_t(j)} = \frac{\sum_{t=1}^T(\bar{\alpha}_t(j)\bar{\beta}_t(j)\cdot\boldsymbol{Y}_t/c_t)}{\sum_{t=1}^T(\bar{\alpha}_t(j)\bar{\beta}_t(j)/c_t)} \ ,$$

$$\hat{\boldsymbol{\Sigma}}_j = \frac{\sum_{t=1}^T \gamma_t(j)\cdot(\boldsymbol{Y}_t-\boldsymbol{\mu}_j)(\boldsymbol{Y}_t-\boldsymbol{\mu}_j)^*}{\sum_{t=1}^T \gamma_t(j)} = \frac{\sum_{t=1}^T(\bar{\alpha}_t(j)\bar{\beta}_t(j)\cdot(\boldsymbol{Y}_t-\boldsymbol{\mu}_j)(\boldsymbol{Y}_t-\boldsymbol{\mu}_j)^*/c_t)}{\sum_{t=1}^T(\bar{\alpha}_t(j)\bar{\beta}_t(j)/c_t)} \ .$$

The log-likelihood can also be recovered:

$$\begin{aligned}\log\mathbb{P}(\boldsymbol{Y}|\lambda) &= \log\sum_{j=1}^N \mathbb{P}(\boldsymbol{y}_{1:T}, X_T = j) = \log\sum_{j=1}^N \alpha_T(j) = \log\frac{\sum_{j=1}^N \bar{\alpha}_T(j)}{\boldsymbol{C}_T} = \log\frac{\sum_{j=1}^N \bar{\alpha}_T(j)}{\prod_{\tau=1}^T c_\tau} \\ &= \log\sum_{j=1}^N \bar{\alpha}_T(j) - \sum_{\tau=1}^T \log c_\tau \ .\end{aligned}$$

The last equality in the equation immediately above is key because it uses the properties of the $\log(\cdot)$ function to recover the log-likelihood without having to multiply together all the scaling factors. If not for the glorious properties of the logarithm, then all would be lost, since the whole purpose of this scaling procedure was to avoid the underflow introduced by multiplying together minuscule quantities such as those contained in the scaling factors.

Lastly, the probability that the hidden state changes between t and $t+1$ can be recovered via

$$\begin{aligned}\sum_{i\neq j}\mathbb{P}(X_t = i, X_{t+1} = j|\boldsymbol{Y},\lambda^*) &= \sum_{i\neq j}\xi_t(i,j) = \frac{\sum_{i\neq j}\alpha_t(i)a_{ij}b_j(\boldsymbol{Y}_{t+1})\beta_{t+1}(j)}{\mathbb{P}(\boldsymbol{Y}|\lambda^*)} = \frac{\sum_{i\neq j}\bar{\alpha}_t(i)a_{ij}b_j(\boldsymbol{Y}_{t+1})\bar{\beta}_{t+1}(j)/(\boldsymbol{C}_t\boldsymbol{D}_{t+1})}{\sum_{j=1}^N \alpha_T(j)} \\ &= \frac{\sum_{i\neq j}\bar{\alpha}_t(i)a_{ij}b_j(\boldsymbol{Y}_{t+1})\bar{\beta}_{t+1}(j)}{\boldsymbol{C}_T\sum_{j=1}^N \bar{\alpha}_T(j)/\boldsymbol{C}_T} = \frac{\sum_{i\neq j}\bar{\alpha}_t(i)a_{ij}b_j(\boldsymbol{Y}_{t+1})\bar{\beta}_{t+1}(j)}{\sum_{j=1}^N \bar{\alpha}_T(j)} \ .\end{aligned}$$

A pseudo-code for an implementation of the Baum-Welch algorithm for learning parameters of a HMM with continuous multivariate emissions, using scaling, can be found in appendix A.

3 Experimental Results

I implemented the Baum-Welch algorithm for continuous emissions, with scaling, on three different datasets, where each dataset corresponds to a different chamber. I will henceforth refer to these three chambers as A,

B, and C. The Baum-Welch algorithm assumes that we know the number of hidden states N. The data for each chamber includes the number of changes that occurred. We can simply add 1 to the number of changes to obtain the the parameter N. Table 1 lists the number of hidden states for each chamber.

Chamber	A	B	C
No. Hidden States	2	2	3

Table 1: Number of Hidden States by Chamber

For each chamber, I sorted the times in decreasing order by the probability (determined by the algorithm) of a hidden state change occurring. Define this ordering as $t_1^*, t_2^*, ..., t_{T_c-1}^*$, where T_c is the length of the emission sequence for chamber c. Recall that the emissions at each time slice represent the parameters of a 2-dimensional damped harmonic oscillator. Thus, for N hidden states, we can visualize the algorithm performance by plotting the data in two dimensions and changing the marker color at the time points $t_1^*, t_2^*, ..., t_{N-1}^*$. As can be seen in Figures 3 and 4, the algorithm seems to have correctly identified points where a state change occurs in chambers A and B. For chamber C, which required two change points , changing the marker color at t_1^* and t_2^* resulted in figure 5; a mediocre result. However, if we instead choose t_1^* and t_3^* as the change points, we get Figure 6; a much better result.

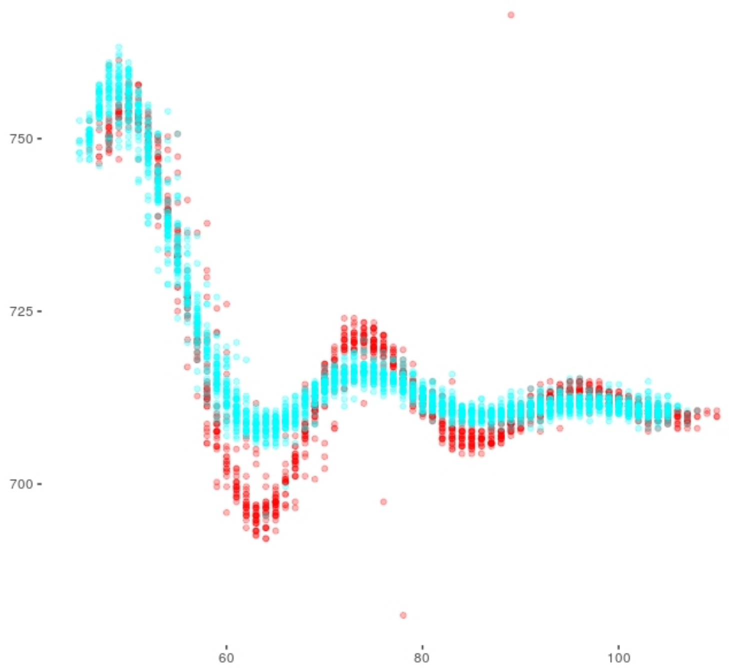

Figure 3: Visualization of Algorithm Performance on Chamber A

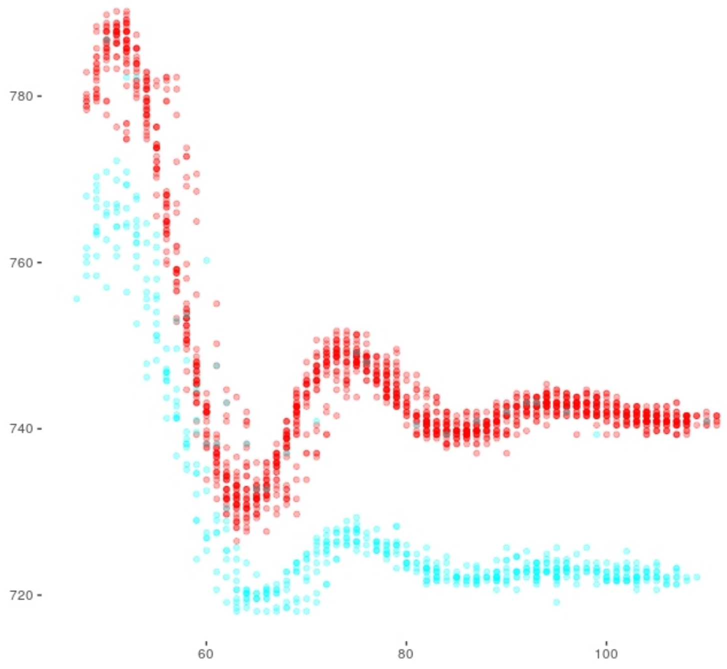

Figure 4: Visualization of Algorithm Performance on Chamber B

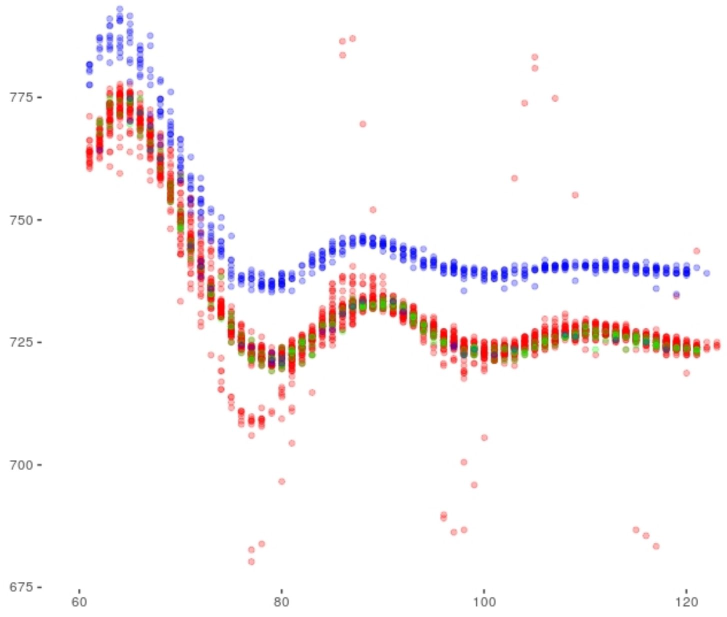

Figure 5: Visualization of Algorithm Performance on Chamber C, Changing Colors at t_1^* and t_2^*

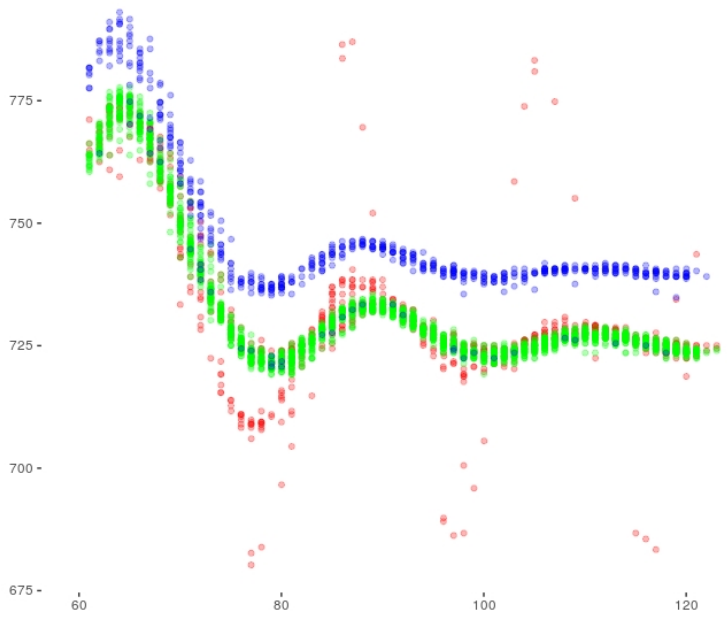

Figure 6: Visualization of Algorithm Performance on Chamber C, Changing Colors at t_1^* and t_3^*

A plausible reason for the significant difference between Figures 5 and 6 becomes readily apparent when we extract the time stamps for t_1^*, t_2^*, and t_3^* from the data for chamber C. As can be seen in Table 2, t_1^* and t_2^* occurred within five days of each other, whereas t_3^* occurred over a month before t_1^* and t_2^*. Therefore, it appears that constraining change points to be at least a certain amount of time τ apart could improve algorithm performance.

t_1^*	t_2^*	t_3^*
2015-06-23 13:10:54.457	2015-06-19 05:28:40.284	2015-05-05 22:03:53.491

Table 2: Time Stamps of t_1^*, t_2^*, and t_3^* in Chamber C

4 Conclusion

This project has presented the theory behind the Baum-Welch algorithm for continuous multivariate emissions, with scaling to avoid underflow. Then, I implemented the algorithm on a real-life data set. By far, the hardest part of the project was dealing with underflow. In fact, I didn't realize that this would be a problem until I had already written the entire algorithm in its original form. While mathematically correct, the algorithm without scaling is nearly useless in practice because, more often than not, time series' are long and emissions are multivariate. Specifically, the number of probabilities that need to be multiplied together increases linearly with the length of the time series, and these same probabilities decrease *exponentially* with the dimensionality of the emissions (here we see another consequence of the curse of dimensionality)! Even with scaling, it is possible to run into underflow if T and/or D are large enough. In cases such as these, we can run the algorithm in logarithmic space, and then exponentiate at the end to obtain the probabilities. When we only seek the *argmax* of either the posterior or the likelihood, which occurs quite often in practice, the monotonicity of the logarithm function means that exponentiation upon algorithm termination is not needed, and so we can run the entire algorithm without ever explicitly seeing a probability!

I was hoping to be able to include a full derivation of the update equations for continuous multivariate emissions from [2], but time did not allow for this. Nevertheless, the proof was mathematically appealing. In short, the proof uses a theorem proved by Fan to represent $\mathbb{P}(\boldsymbol{Y}, \boldsymbol{s})$ as an expected value of over some measure space $\boldsymbol{G}$, where $\boldsymbol{s}$ is some realized sequence of emissions. Thus, optimizing the marginalization over $\boldsymbol{s}$ of the new representation is equivalent to optimizing $\mathbb{P}(\boldsymbol{Y})$. Liporace uses the method of Lagrange to carry out the optimization.

For future work, the algorithm can be given the capability of determining how many hidden states there are with an infinite hidden markov model. Also, the damped harmonic oscillator defined by each emission corresponds to data generated by an actual signal: for this project, I used temperature. In addition to temperature, the dataset contains measurements from 16 other sensors, each of which defines its own parametric curve over time. An interesting extension of my work would be to apply it over all the sensors contained in my dataset; not just temperature. One way of doing this would be to employ a multivariate gaussian tensor representation. Lastly, the algorithm would perform better in theory without the independence assumption, although this would likely lead to an unacceptable jump in computation time.

5 Appendix A: Baum-Welch algorithm for Learning HMM Parameters for Continuous Multivariate Independent Emissions

```
procedure BaumWelch(sequence of length T of D-dimensional observations $\boldsymbol{Y}$, number of hidden states $N$, tolerance $\epsilon > 0$)
    Initialize $\pi$ to an $N$-vector of uniform random values.
    Initialze $A$ to an $N \times N$ matrix of uniform random values
    Initialize $\boldsymbol{\mu}_j$ to the $j^{th}$ center returned by the kmeans algorithm applied to $\boldsymbol{Y}$, $\forall j \in \{1, ..., N\}$
    Initialize $\sigma_{dj}$ to 1, $\forall (d, j) \in \{1, ..., D\} \times \{1, ..., N\}$
    $\boldsymbol{\sigma}_j \leftarrow [\sigma_{dj}]_{d=1}^{D}$ , $\forall j \in \{1, ..., N\}$
    Initialize $B$ to a $T \times N$ matrix with $b_{tj} = \mathbb{P}(\boldsymbol{Y}_t = \boldsymbol{y}_t | X_t = j)$, where $\boldsymbol{Y}_t | X_t = j \sim \mathcal{N}(\boldsymbol{\mu}_j, \boldsymbol{\sigma}_j)$
    $LLold \leftarrow -999999$
    $LLnew \leftarrow$ loglikelihood$(\pi, A, B)$
    while $LLnew - LLold > \epsilon$ do
        $LLold \leftarrow LLnew$
        Update $\pi$, $A$, and $B$ according the scaled update equations in section 2.3
        $LLnew \leftarrow$ loglikelihood$(\pi, A, B)$
    end while
    return $\pi$, $A$, $B$
```

References

[1] L. Rabiner; *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*

[2] L.A. Liporace; *Maximum Likelihood Estimation for Multivariate Observations of Markov Sources*

[3] D. Shen; *Some Mathematics for HMM*

[4] T. M. Luong, V. Perduca, and G. Nuel; *Hidden Markov Model Applications for Change Point Analysis*