BAYESIAN STRUCTURAL INFERENCE AND THE BAUM-WELCH ALGORITHM

DMITRY SHEMETOV

UNIVERSITY OF CALIFORNIA AT DAVIS

Department of Mathematics
dshemetov@ucdavis.edu

Abstract. We perform a preliminary comparative analysis of the Baum Welch (BW) algorithm and the Bayesian Structural Inference (BSI) method presented by Crutchfield and Strelioff [1]. The two methods are compared for accuracy in the context of an initial group of hidden Markov models. The accuracy of the methods is specifically analyzed as a function of varying sample size. Our initial results demonstrate a more accurate performance from BSI method in almost all cases investigated. The investigation is not in any way conclusive, however.

1. Introduction

As ϵ-machines were initially introduced in the course, we worked within the abstract framework of looking at bi-infinite symbol sequence strings and attempting to infer the structure behind these. The concept of ϵ-machines spawned as minimal-state, maximally-predictive hidden Markov models (HMM) that could reproduce a given bi-infinite string, with all of its symbol and sub-word distributions exactly. We then focused on using ϵ-machines to calculate, with closed form expressions, various complexity-measuring quantities of interest, such as structural complexity, entropy rate, excess entropy, etc. In contrast with this usage, this paper narrates an exploration of ϵ-machines in a slightly different direction, with an eye for system modeling and inference.

Naturally, seeing these bi-infinite symbol sequences, we were intrigued by the possibility of using ϵ-machines for a purpose that was briefly mentioned - time-series modeling and analysis. Having some minor exposure to the time-series analysis field, essentially knowing that it dealt with similar questions of detecting structure in bi-infinite, discretely-indexed data strings, we were curious to see howϵ-machine reconstruction methods would compare with standard methods within the field. Cursorily investigating the literature, it appears that there is an unawareness of ϵ-machines in the machine learning or data mining fields. Thus, this investigation hopes to represent an initial attempt at bridging the two areas of study via comparative analysis.

The time series field itself barely needs motivation as an interesting area of study, as much of the data received in science or industry takes the form of data samples from a system performed at regularly-spaced intervals, where analysis is necessary, so we omit that. However, we will comment on the space of algorithms and methods that comprise the field - it is quite a large field, with a multitude of questions being

asked, which makes it hard to describe a generic time series method. This, in turn, makes it quite difficult to come up with a uniform metric or standard with which to compare methods. Setting these broad comparison questions aside, we choose to make progress by reducing to the subfield that focuses on hidden Markov models. This area is concerned with methods for modeling data sets with hidden Markov models and is widely used in data mining and machine learning applications.

For this particular investigation we chose to study the BW algorithm, as it promises to be most accessible for study within this project's time period. There exist readily available implementations of this algorithm, it is sufficiently well known, and generally deals with the same set of problems that ϵ-machines are concerned with.

Roughly, the BW algorithm accepts a model topology (a starting set of parameters for an HMM), a sequence of observed symbols from a source we would like to model, and iteratively produces a HMM with parameters that maximize the likelihood of observing the sequence of observed symbols. The BSI method achieves a similar result, accepting a model topology, a sequence of observables, and produces a probability distribution over the parameter space of HMMs. This probability distribution assigns a likelihood degree to which each HMM parameter configuration could have produced a given observable string.

We proceed to generally describe the results of our comparisons.

2. Summary of Results

Before getting into some technical details about the methods, we describe some basic results that we found. We must first stress that these are very much initial results, with the algorithms being compared in slightly different ways. A deeper investigation is needed.

However, in comparing the two algorithms, we found that the BSI algorithm, in the limited context studied, performed better than the BW algorithm, by producing more accurate approximations. The specific comparison metric we employed was by creating data strings for both algorithms from known models, creating a number of approximations using the data with the methods, and then considering the 2-norm error between the outputted approximation and the true model. The comparisons were made over 6 different HMMs, all with low ($\leq$5 Mealy / 8 Moore) state numbers. These accuracy comparisons were also studied, for each model, as functions of the sample-data size, with the intent of seeing differing convergence rates in the methods. In almost all cases studied, the BSI algorithm possessed an error of almost an order of magnitude below the error of the BW algorithm.

3. Some technical and theoretical details on the algorithms

In this section, we introduce standard technical notation for dealing with HMM's. We assume basic familiarity with Markov chains and hidden Markov models.

3.1. **Hidden Markov Models.**

Broadly speaking, HMM's are Markov chains with a secondary structure imposed, that of an observable symbol set that is emitted in addition to an internal state transition dynamic.

There are two main types of HMM's, Mealy and Moore, distinguished by when the observable symbols are emitted - in the Mealy case, they are emitted in transition between states, while in Moore machines, symbols are emitted while in a

state. There is convenient alternative terminology, which is potentially more explanatory, in which Mealy machines are referred to as being "edge-labeled", while Moore machines are "state-labeled". This distinction will come into play later as the implementations this investigation used involved the separate types, which, in turn, caused some technical problems in the analysis.

3.1.1. *Moore HMM.* Our objects of interest are discrete HMMs, with N hidden states and M possible observed symbols.

-The set of states is denoted $S = \{S_1, ..., S_N\}$, while a state at time t is denoted q_t.

-A starting distribution over these states $\pi = \{\pi_1, ..., \pi_N\}$.

-The hidden state dynamic is determined by the transition matrix $A = \{a_{ij}\}$ of probabilities, where $a_{ij} = P[q_{t+1} = S_j | q_t = S_i]$.

-The set of observable symbols is $V = \{v_1, ..., v_M\}$.

-Finally, each state-symbol pair has an associated probability, considered as the probability of the given state emitting the symbol. This is encoded in a set of functions $B = \{b_i\}$ where $b_i(k) = P[v_k \text{ at time } t | q_t = S_i]$.

A Moore machine is determined by the parameter set $\lambda = \{\pi, S, V, A, B\}$.

3.1.2. *Mealy HMM.* Again, we have N hidden states and M possible observed symbols.

-The set of states is denoted $S = \{S_1, ..., S_N\}$, while a state at time t is denoted q_t.

-A starting distribution over these states $\pi = \{\pi_1, ..., \pi_N\}$.

-The set of observable symbols is $V = \{v_1, ..., v_M\}$.

-*The hidden state dynamic and the emission are handled simultaneously. There is a transition matrix associated with every symbol v_i. This matrix $T^{(v_i)} = \{a_{ij}^{v_i}\}$ represents the probability of transition from state i to state j and emitting the symbol v_i, that is, $a_{ij}^{v_i} = P[v_k \text{ at time t} | q_t = S_i, q_{t+1} = S_j]$.

A Mealy machine is determined by the parameter set $\lambda = \{\pi, S, V, T^{(v_i)}\}$.

3.2. **Baum-Welch.** Now, we describe, in some detail, the internals of the Baum-Welch algorithm. The description that follows is a rough sketch of a larger description that can be found in [2].

Suppose that we are provided a string of observables $O = \{O_1, ..., O_T\}$. Our goal is to find a set of parameters λ^{opt} that maximizes the likelihood of the string, denoted $P(Y|\lambda)$. An iterative scheme has been devised by Baum [et. al] that uses expectation minimization to find the maximum likelihood estimate of the parameters of the HMM. It can be seen akin to a gradient ascent algorithm in the parameter space of HMM's.

To sketch: we have the aforementioned observables Y. Our initial parameter is a randomized guess λ_0. Next the forward-backward algorithm is applied to calculate the following quantities:

-$\alpha_t(i) = P(O_1, O_2, ..., O_t; q_t = S_i | \lambda)$ in the forward pass.

-$\beta_t(i) = P(O_{t+1}, O_{t+2}, ..., O_T | q_t = S_i, \lambda)$ in the backward pass.

After finding these, we are able to find

$$\gamma_t(i) = P(q_t = S_i | O, \lambda) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{n=1}^{N} \alpha_t(i)\beta_t(i)}$$

which has a more intuitive interpretation than α or β. It is the probability of being in state S_i at time t, given the observed sequence and the parameters.

Another important quantity that can be computed from α and β is

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}$$

This is the probability of transitioning from state i to state j at time t. Notice that if we sum over all time

$$\sum_{t=1}^{T}\xi_t(i,j)$$

we obtain the expected number of transitions from state i to state j.

With this in mind, we also note that

$$\sum_{t=1}^{T}\gamma_t(i)$$

is the expected number of transitions out of state i.

Using these quantities, we can now produce:

-a new starting distribution $\bar{\pi} = \gamma_1(i)$ - expected number of times in state i at time $t = 1$

-a new hidden state transition matrix

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

-new symbol emission functions

$$\bar{b}_i(k) = \frac{\sum_{t=1|O_t=v_k}^{T-1}\gamma_t(i)}{\sum_{t=1}^{T-1}\gamma_t(i)}$$

These newly constructed variables form a new parameter family for the model $\lambda' = \{\bar{\pi}, S, V, \bar{A}, \bar{B}\}$. This model has the property that $P(Y|\lambda') \geq P(Y|\lambda)$. Running this process a number of times in succession yields a sequence of HMM parameters that eventually converge on an optima in parameter space. Given that this algorithm uses gradient-ascent, it is prone to becoming stuck in local optima. To prevent this, it is generally seeded with a number of initial guesses λ_0 which are iterated through and the best scoring model is chosen.

3.3. **Bayesian Structural Inference.** We now roughly describe the BSI method, but in less detail than the above BW algorithm. For a full treatment, we defer to [1], and for some worked examples, to [3].

The goal of BSI is similar to the BW algorithm - given some observed data O and an assumed model M_i (the network topology), we want to discover the unknown parameters λ_i and the starting distribution σ_i for the model M_i that are likely to have produced O. Where BSI distinguishes from BW, is that instead of looking for a specific λ to converge on, we look to calculate the following distribution over parameters - $P(\lambda_i|O, M_i)$. With sufficient data O, the probability distribution should cluster around a small set of likely machines, while also providing error and confidence bars for the model parameters inferred.

Note that, crucially, this method is restricted to the **unifilar** subclass of HMM's. Unifilarity is a specific property for a Mealy HMM that a state-symbol pair uniquely determines the next internal state. This translates to a nice subclass of HMM's that have less potential for ambiguity than general HMM's.

A main conceptual point in BSI is to decompose the goal distribution via Bayes' theorem:

$$P(\lambda_i|O, M_i) = \frac{P(O|\lambda_i, M_i)P(\lambda_i|M_i)}{P(O|M_i)}$$

We comment on the newly spawned terms:

-The likelihood, or the likelihood of the data given the model and the parameters:

$$P(O|\lambda_i, M_i)$$

Note: this quantity is analogous to what the BW algorithm seeks to maximize $P(O|\lambda)$. We are not entirely sure about the theoretical relationship this creates between the two algorithms, but it seems to be worth noting that the methods seem very related and the exact theoretical differences should be explored.

-The prior, or the likelihood estimate without the data D:

$$P(\lambda_i|M_i)$$

-The evidence, or the model comparison term:

$$P(O|M_i) = \sum_i P(O|\lambda_i, M_i)P(\lambda_i|M_i)$$

For a detailed explanation of the calculation of these quantities, we refer the reader to [BSIHP].

4. Some technical details of the method comparison

4.1. ***Scikit-Learn* and *CMPy*.** To implement this analysis, we used the Python library *Scikit-Learn* (SL) for its BW implementation. The BSI method used was implemented in the *CMPy* package. In order to make the comparisons, there were some differences in the methods that needed to be bridged.

4.1.1. *Moore versus Mealy HMM's.* As previously mentioned, the issue of Moore versus Mealy HMM's arose. In particular, the BW algorithm implemented in SL was based on handling Moore machines, while the BSI method operates on Mealy HMM's. Naturally, we investigated the question of converting between the two types. In short, conversion methods exist but they are not canonical (there exist multiple methods by which to convert), nor are they symmetric (switching Mealy -> Moore is not at all similar to switching Moore -> Mealy), nor do they preserve the state-topology structure (often, one must create new states and expand the state size).

For the sake of some initial work, we coded a Moore -> Mealy converter, but quickly realized that the method was generating non-unifilar Mealy HMM's, something BSI could not work with. Further attempts at coding a converter that preserves unifilarity took too long and were abandoned. Creating a Mealy -> Moore conversion algorithm met the same fate.

Finalizing these investigations, we chose to manually convert a select library of Mealy machines into Moore machines and compare within that domain, leaving the question of performance on the whole space of HMMs to later investigations.

4.1.2. *Iteration and sample sizes.* We now mention the parameter values chosen when running the comparisons.

Due to the gradient-ascent nature of the BW algorithm, it needed to be seeded with a uniform sampling of starting conditions in parameter space, run multiple times, after which we selected the highest and lowest scoring models. The seed number used was 25. The number of iteration steps used for the BW algorithm was 25. We realize that these numbers are quite low, but our BW implementation ran very slowly otherwise.

Since the BSI algorithm produces are probability distribution over parameter space, we generated a sample of 200 models in the resulting distribution and selected the highest and lowest scoring models.

5. Results of analysis

5.1. **Accuracy as a function of sampling window size.** The methods were compared on a few different metrics.

First, we generated the true models that we sampled the data from. Then, we fed this data into each individual algorithm and chose the best and worst models in the sample, as outlined above. We then scored these models by combining the squared error of all the matrices together with an L^2 norm. The L^2 norm was used because the norm produces an averaging effect on the error and combines all the differences into a single, rounded error value, while magnifying especially large differences.

Specifically, the scoring went as follows:

-In the Moore HMM case, taking true model parameters to be $\lambda^{Tr} = \{\pi^{Tr}, A^{Tr}, B^{Tr}\}$ and the approximate model to be $\lambda^{Ap} = \{\pi^{Ap}, A^{Ap}, B^{Ap}\}$, the score of the approximate model is calculated as follows:

$$\|\lambda^{Tr} - \lambda^{Ap}\| = (\|A^{Tr} - A^{Ap}\|_2^2 + \|B^{Tr} - B^{Ap}\|_2^2 + \|\pi^{Tr} - \pi^{Ap}\|_2^2)^{\frac{1}{2}}$$

where $\|\cdot\|_2$ denotes the L^2 norm for vectors/matrices (sum of the squares).

-In the Mealy HMM case, with $\lambda^{Tr} = \{\pi^{Tr}, (T^{(v_k)})^{Tr}\}$ being the true model and $\lambda^{Ap} = \{\pi^{Ap}, (T^{(v_k)})^{Ap}\}$ being the approximate model, the score was calculated via:

$$\|\lambda^{Tr} - \lambda^{Ap}\| = (\sum_{k=1}^{M} \|(T^{(v_k)})^{Tr} - (T^{(v_k)})^{Ap}\|_2^2 + \|\pi^{Tr} - \pi^{Ap}\|_2^2)^{\frac{1}{2}}$$

The comparisons were made in the context of reproducing each of the following HMM. Given that we knew the Mealy versions of each process, we then generated a Moore version, using our own algorithm. We tested to make sure that each of the machine pairs produced the same symbol sequence distributions.

We then used the Mealy version of the process to generate a sequence of symbols, which was then fed into the BSI method, which we then scored against its approximation of the Mealy model. The BW algorithm was then applied to its Moore counterpart and was scored against its reproduction of that model.

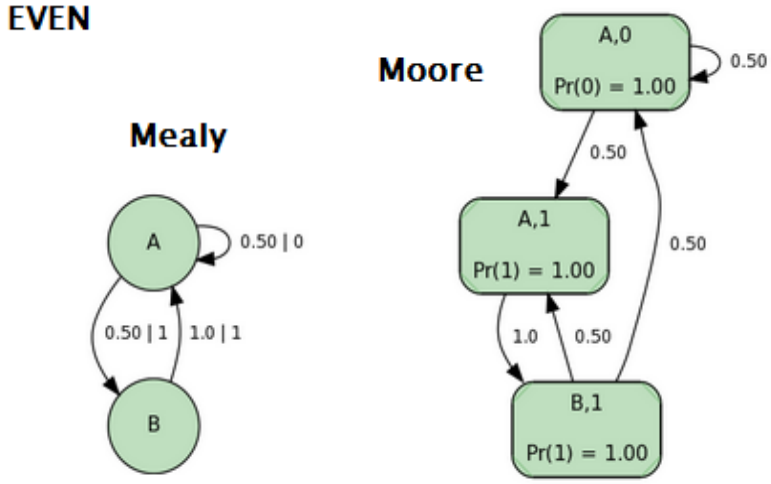
5.2. **Plots.** The structure of this section is as follows:

For each process, we present an HMM diagram of the models and then plot the behavior of the best-case and worst-case approximations, as functions of sampling size, for each method. The best-case approximate is in blue, while the worst-case approximate is in green.

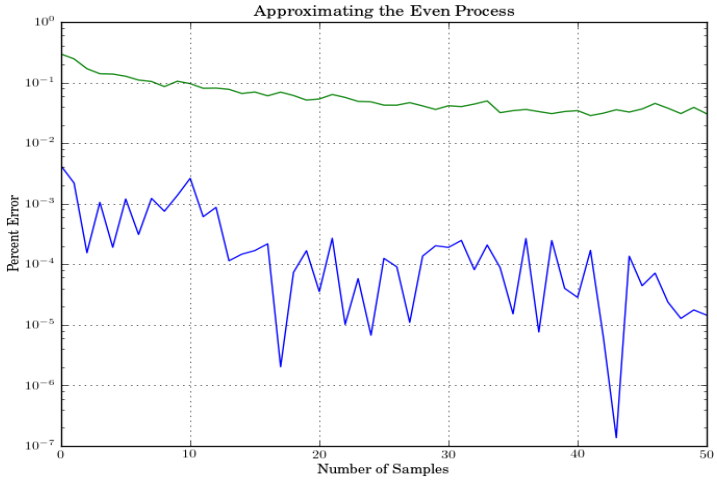
We were able to study the performance of the BSI method for samples ranging from 100 - 5000, while the BW algorithm was only studied 100 - 1000, due to the BW algorithm requiring more computational resources. It should be noted, that the algorithms were compared with respect to their respective versions of the models.

Each plot presents the percent error as a function of the sample size. The BSI plots have their y-scale in the log format (except for the RRXOR process).

5.2.1. *The Even Process.* A diagram of the even process:

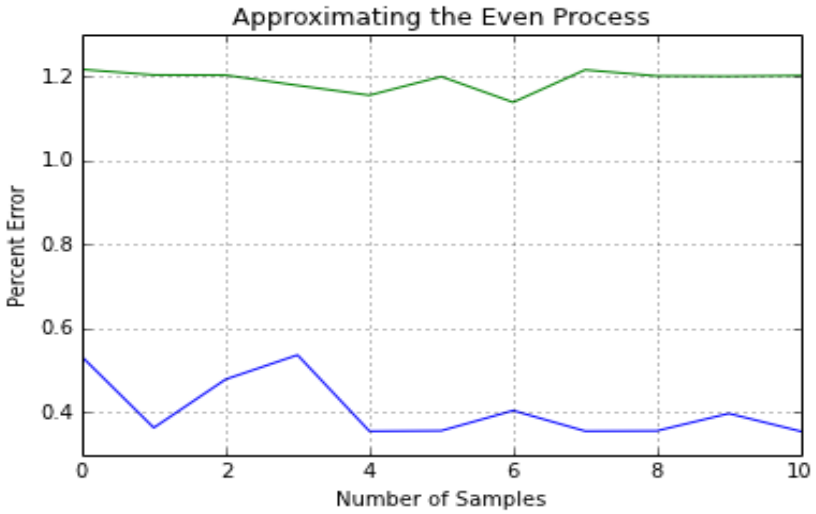


BSI



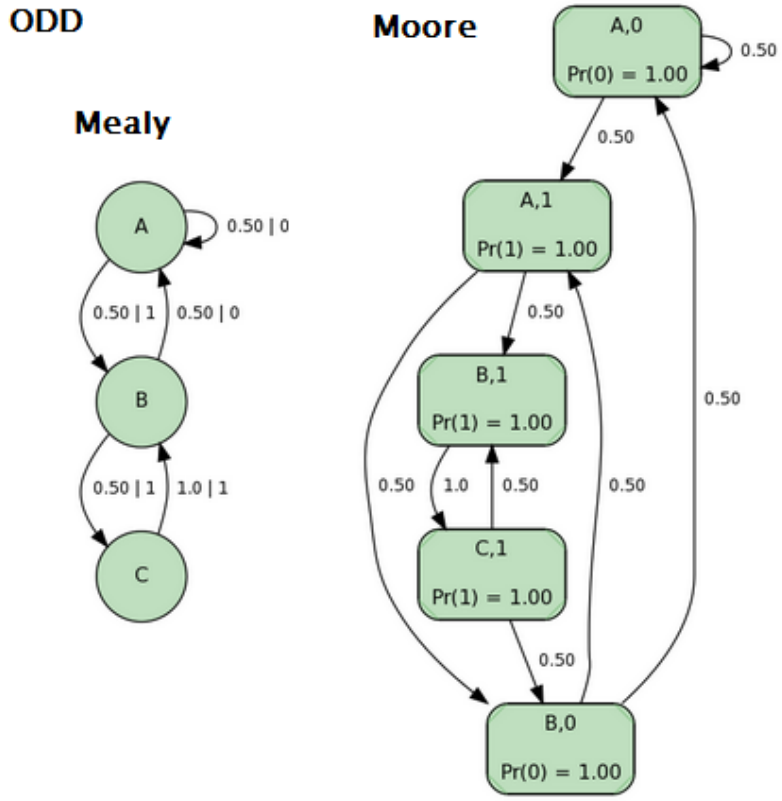
Something of a noisy-linear trend can be seen, suggesting an exponential decay in the error.

BW



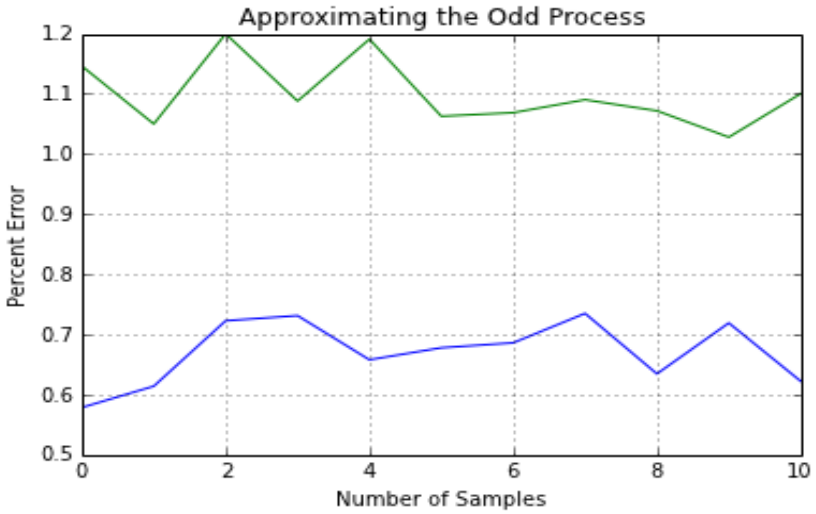
Notice that the scale on this is not log-plot scale.

5.2.2. *The Odd process.* A diagram of the odd process:

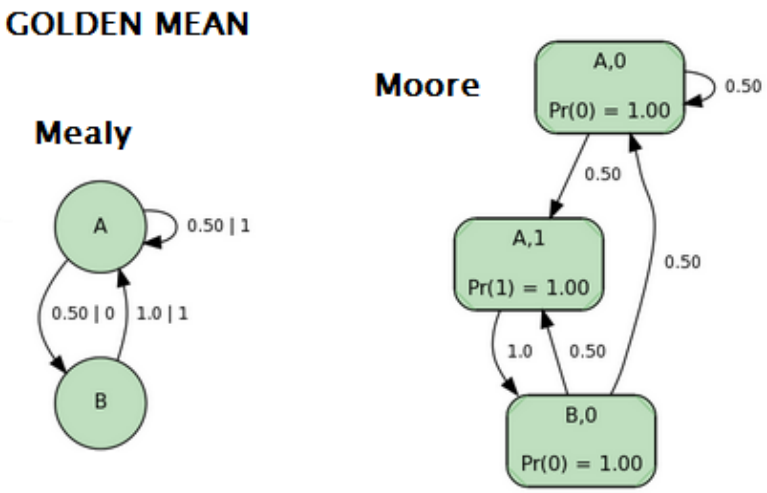


BSI

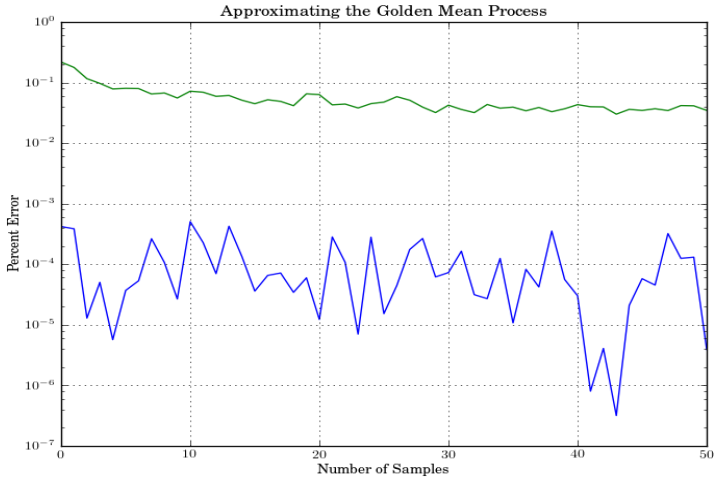
BW



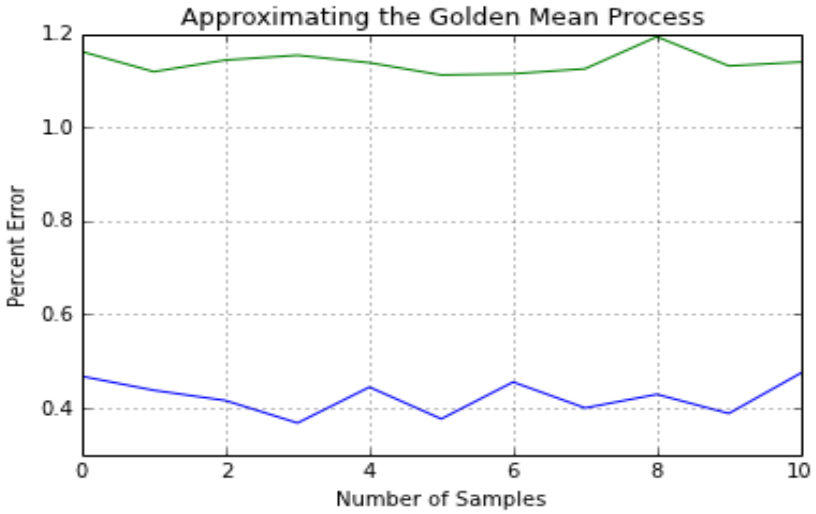
5.2.3. *The Golden Mean process.* A diagram of the golden mean process:


GOLDEN MEAN
Mealy
A
B
0.50 | 1
0.50 | 0
1.0 | 1
Moore
A,0
Pr(0) = 1.00
0.50
0.50
A,1
Pr(1) = 1.00
0.50
1.0
0.50
B,0
Pr(0) = 1.00

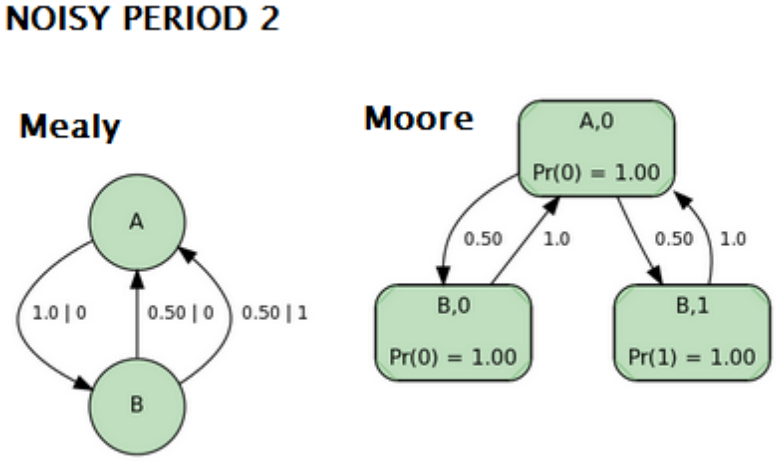
BSI


Approximating the Golden Mean Process
Percent Error
Number of Samples
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
0
10
20
30
40
50

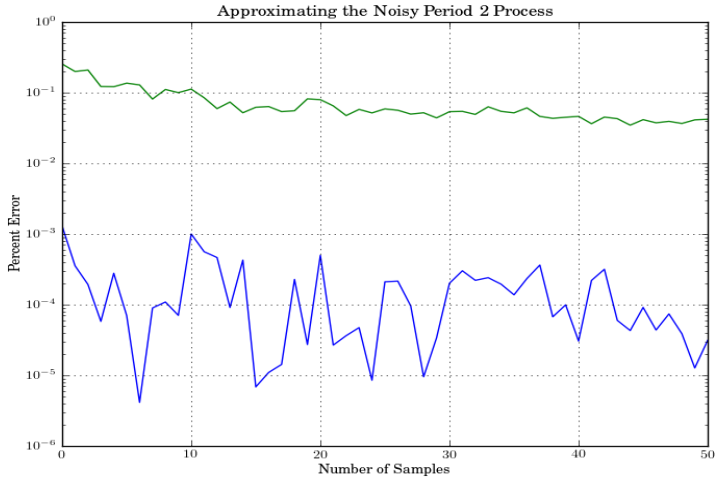
BW



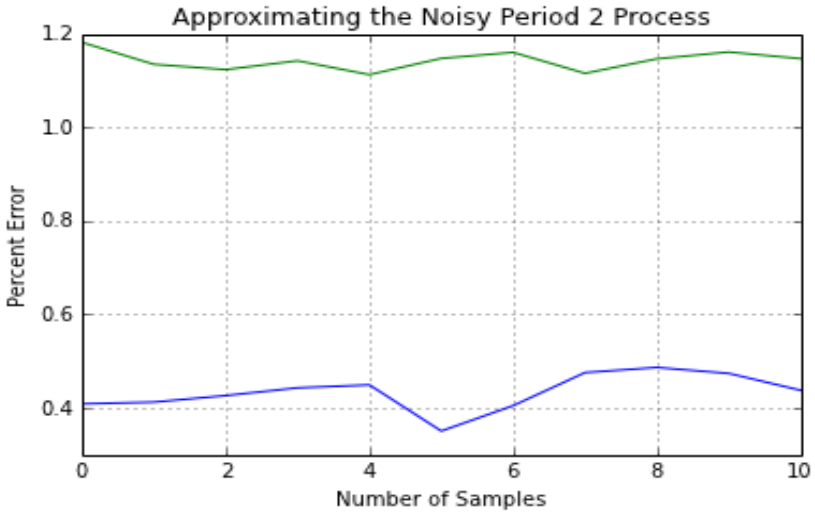
5.2.4. *The Noisy Period 2 process.* A diagram of the noisy period 2 process:



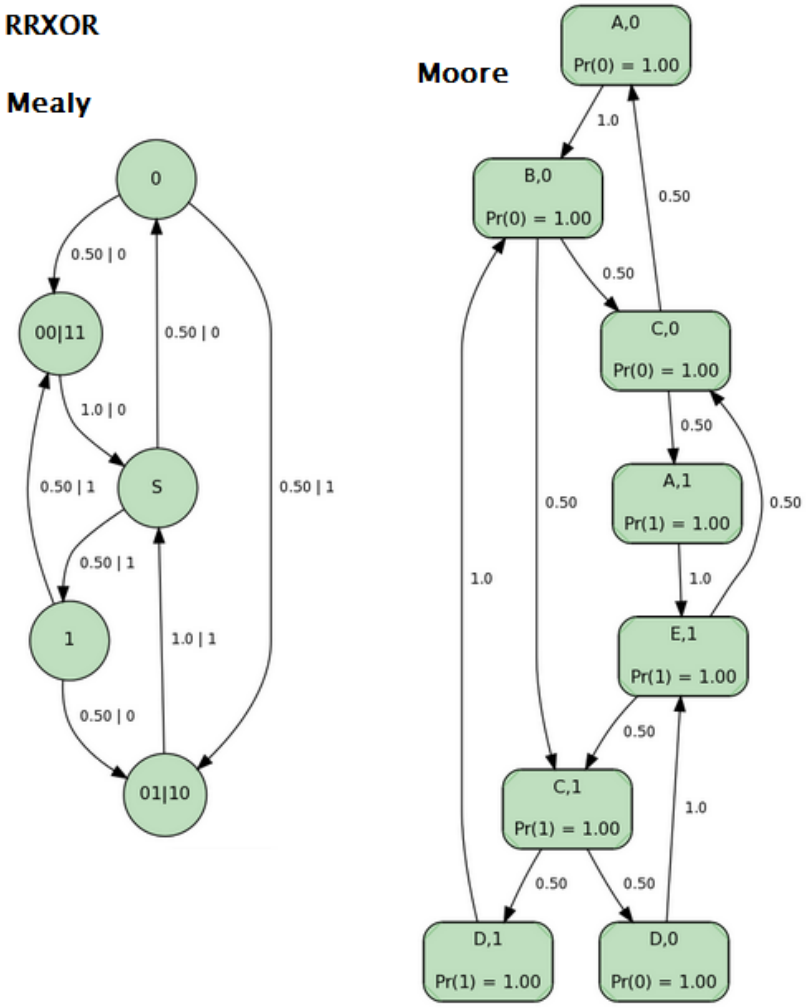
BSI



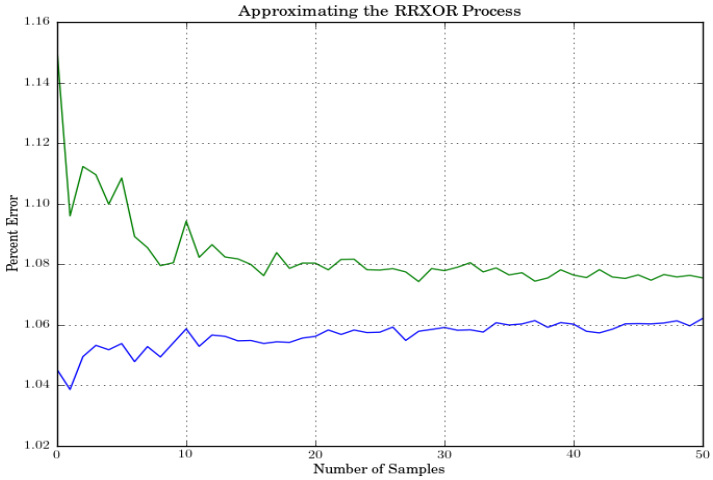
BW



5.2.5. *The RRXOR process.* A diagram of the RRXOR process:


RRXOR
Mealy
0
0.50 | 0
00|11
0.50 | 0
1.0 | 0
0.50 | 1
S
0.50 | 1
0.50 | 1
1
1.0 | 1
0.50 | 0
01|10
Moore
A,0
Pr(0) = 1.00
1.0
B,0
Pr(0) = 1.00
0.50
0.50
C,0
Pr(0) = 1.00
0.50
A,1
Pr(1) = 1.00
0.50
0.50
1.0
1.0
E,1
Pr(1) = 1.00
0.50
C,1
Pr(1) = 1.00
1.0
0.50
0.50
D,1
Pr(1) = 1.00
D,0
Pr(0) = 1.00

BSI

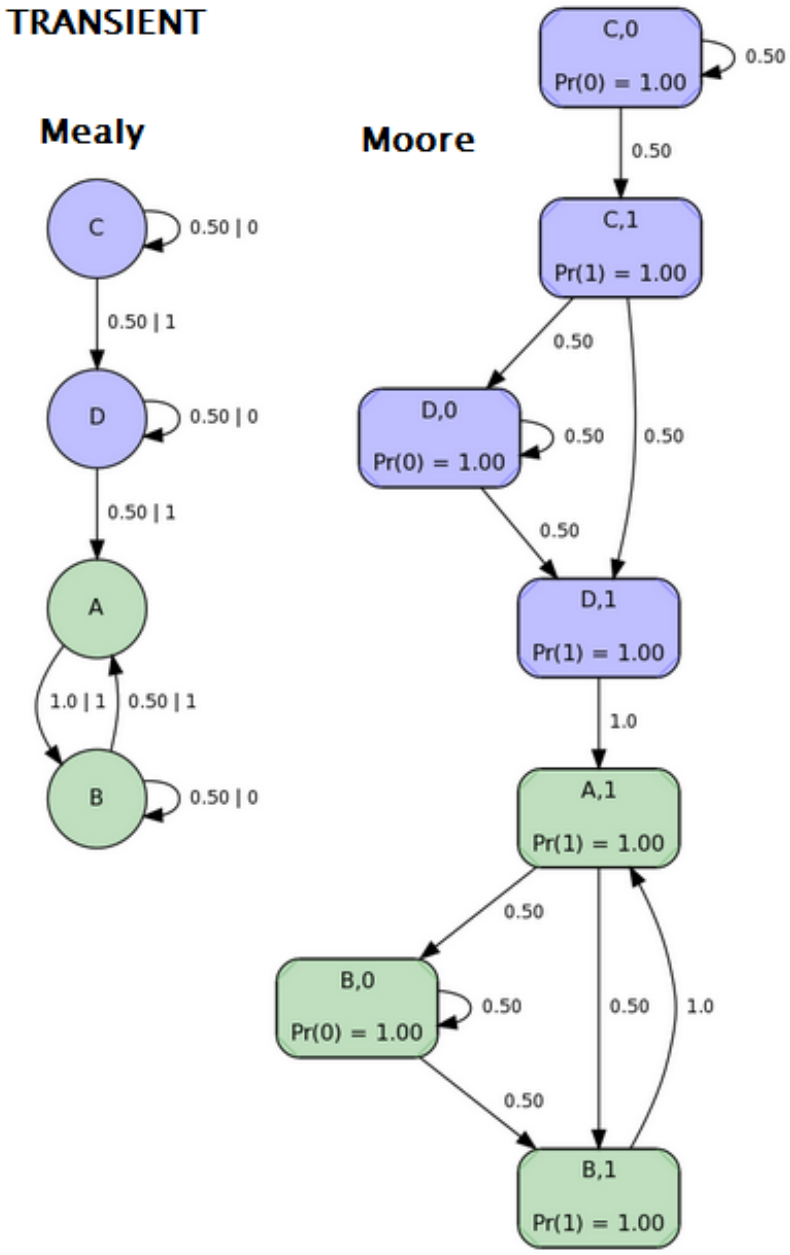


This is probably the strangest case - the method began to trend upwards in its error as we added more data. A potential overfitting phenomenon?

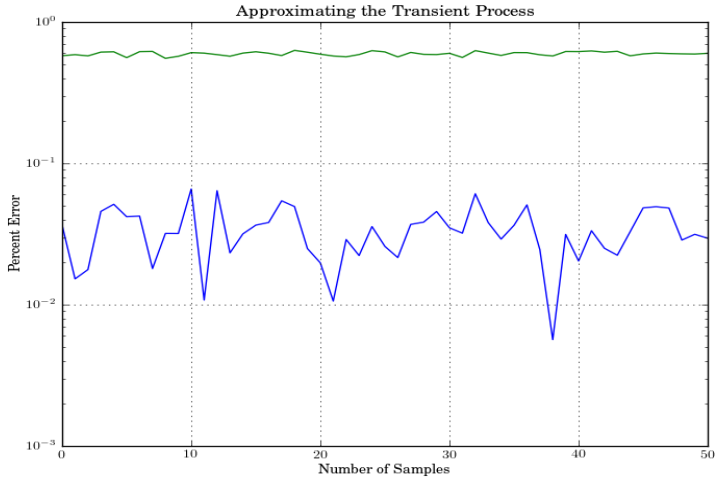
BW

An unfixed bug in the code prevented us from generating a plot for this case.

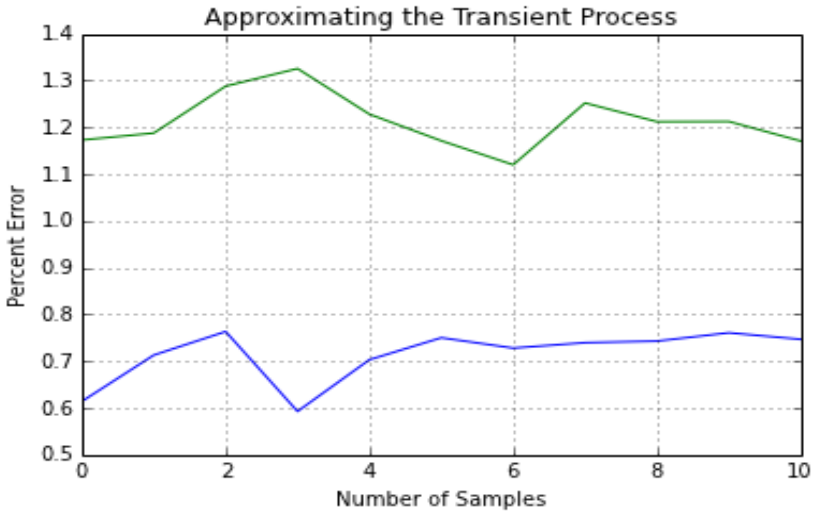
5.2.6. *The Transient process.* A diagram of the transient process:


TRANSIENT
Mealy
C
0.50 | 0
0.50 | 1
D
0.50 | 0
0.50 | 1
A
1.0 | 1
0.50 | 1
B
0.50 | 0
Moore
C,0
Pr(0) = 1.00
0.50
0.50
C,1
Pr(1) = 1.00
0.50
D,0
Pr(0) = 1.00
0.50
0.50
0.50
D,1
Pr(1) = 1.00
1.0
A,1
Pr(1) = 1.00
0.50
B,0
Pr(0) = 1.00
0.50
0.50
1.0
0.50
B,1
Pr(1) = 1.00

BSI



BW



6. Conclusion

Our initial experiments have found that the BSI algorithm obtains a consistently lower error than the BW algorithm, while having much shorter run-times. However, we stress that these are highly, highly rough and prelimenary results, obtained by comparing the two algorithms on a slightly uneven footing.

In order to make a fairer comparison, we would need to:

-Settle the Moore-Mealy conversion issue. In this analysis, the BW algorithm was at a disadvantage, since it had to deal with larger-state models, thereby increasing its parameter search space and increasing room for error.

-Investigate further the error-norm issue - it is possible that the norm employed may somehow be favoring the BSI method.

-Explore the broader HMM space and see how the two algorithms perform in on models with higher numbers of states, connections, and alphabets.

-Run the BW algorithm with more iterations, greater seeding, and study the effects of larger data sample sizes. We do not believe that seed=25 and iterations=25 is enough to do this algorithm justice, but we did not have the computational resources to use larger parameters. For similar reasons, we only explored 1000 observable samples for the BW algorithm, compared to the 5000 for the BSI algorithm.

Further investigative questions arose that we did not have a chance to explore:

-How are the algorithms affected by non-trivial topologies of networks? That is to say, since the BW algorithm generally assumes a perfectly connected model graph and to simulate the lack of a connection, it assigns a very low probability to an edge that should not exists. How can this point be addressed by the BW algorithm, given that the BSI method allows for considering just exactly the set of models where a particular connection does not exist and produce its distribution over a subset of HMM's.

-Explore specifically the performance of the algorithms in the low-data regime. It would be beneficial to know which algorithm performs best when the data available is limited.

-Bonus: we investigated musical representation of HMMs, as an aside into audio data representation and an application to generative music composition. We implemented a simple interpreter of HMM observed sequences as note sequences and used *PySynth* to generate pitch sequences, to see if the sequences the two algorithms inferred sounded different. The generated audio library is available on request via email. The authors are currently actively experimenting with this.

References

[1] C. C. Strelioff and J. P. Crutchfield, "Bayesian Structural Inference for Hidden Processes", Physical Review E 89 (2014) 042119.

[2] Lawrence R. Rabiner (February 1989). "A tutorial on Hidden Markov Models and selected applications in speech recognition".

[3] C. C. Strelioff, "Lecture 33: Bayesian Inference for ϵ-machines".