

# Hidden Markov Models

Diljeet Jagpal

-

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Preliminaries</b>	<b>2</b>
2.1	Mathematical Foundations . . . . .	2
2.1.1	Probability Theory . . . . .	2
2.1.2	Conditional Probability . . . . .	2
2.1.3	Stochastic Process . . . . .	3
2.2	Applied Foundations . . . . .	3
<b>3</b>	<b>Standard Markov and Markov Property</b>	<b>4</b>
3.1	History . . . . .	4
3.2	Markov Chain . . . . .	4
3.3	Motivating the Hidden Markov Model . . . . .	8
<b>4</b>	<b>Hidden Markov Model</b>	<b>11</b>
4.1	Definition . . . . .	11
4.2	Using HMM . . . . .	12
4.2.1	Predictive Model . . . . .	12
4.2.2	Three Key Problems . . . . .	14
4.3	Problem 1: Evaluation . . . . .	14
4.3.1	Forward-Backward Algorithm . . . . .	16
4.4	Problem 2: Decoding . . . . .	19
4.4.1	Viterbi Algorithm . . . . .	20
4.5	Problem 3: Learning . . . . .	22
4.5.1	Baum-Welch Algorithm . . . . .	22
4.6	Modified HMM . . . . .	23
4.6.1	GMM . . . . .	23
<b>5</b>	<b>Model Selection / Parameter Estimation</b>	<b>24</b>
5.1	Maximum Likelihood Estimators . . . . .	24
5.2	Approximate Bayesian Computation . . . . .	25
5.3	Kolmogorov–Smirnov test . . . . .	25
<b>6</b>	<b>Replicating Existing Rainfall Models</b>	<b>26</b>
6.1	Coppertwait . . . . .	26
6.2	Grando - HMM Based Model . . . . .	27
6.2.1	Model . . . . .	27
6.2.2	Model Fitting . . . . .	27

6.3	Numerical Results . . . . .	28
6.3.1	Code . . . . .	28
6.3.2	Results - Nine Parameter Estimation . . . . .	28
6.3.3	Results - Three Parameter Estimation . . . . .	29
6.3.4	Adjusted ABC Algorithm . . . . .	31
<b>7</b>	<b>Extending HMM based Rainfall Models</b>	<b>34</b>
7.1	Disk Structure . . . . .	34
7.2	Split monthly . . . . .	35
7.3	High dimensionality . . . . .	35
<b>8</b>	<b>Simple HMM Rainfall Model</b>	<b>36</b>
8.1	Applying Baum Welch to Rainfall data . . . . .	36
8.1.1	Building the Observation Matrix . . . . .	36
8.1.2	Local optimum vs global Optimum . . . . .	36
8.2	Analysis . . . . .	37
8.2.1	Convergence Log . . . . .	37
8.2.2	Selecting the optimum optima . . . . .	38
8.2.3	Results . . . . .	38
<b>9</b>	<b>Generalising the Observation Matrix</b>	<b>40</b>
9.1	Generalising the model . . . . .	40
9.2	Finding a function for Observation Matrix . . . . .	40
9.3	Parameter Estimation . . . . .	40
9.3.1	Methodology . . . . .	41
9.3.2	Results . . . . .	41
<b>10</b>	<b>Results</b>	<b>46</b>
10.1	Our Datasets . . . . .	46
10.2	Results . . . . .	46
10.2.1	CDFs . . . . .	46
10.2.2	Kologmorov-smirnof Tests . . . . .	46
<b>11</b>	<b>Future Research</b>	<b>48</b>

# Preface

Preface

# Chapter 1

## Introduction

The goal of this project is to determine if HMMs are suitable as rain generators.

The first task will be to extend on the work done by Grando. In her testing, she has concluded that HMMs are suitable as rain generators; however, she has used the same data for testing as she used for training. Doing so may, quite likely, have to lead to bias, and thus, we will extend her work by conducting out-of-sample tests.

If possible, we will build the software, so it is user friendly and efficient. With this, we can test data for multiple locations. This will allow us to understand if the result is genuinely significant, at least more than just one location.

# Chapter 2

## Preliminaries

In this section, we will briefly visit the foundations on which we will build throughout this paper. For most, this will be a simple refresher.

### 2.1 Mathematical Foundations

We start with a few key mathematical concepts.

#### 2.1.1 Probability Theory

To discuss any probabilistic ideas, we must first understand general probability theory. We will start by defining a probability space.

**Definition 2.1.** Probability Space

A probability space is defined by  $(\Omega, \mathcal{F}, \mathbb{P})$ .  $\Omega$  is the non-empty set of all possible outcomes, such that all events  $\omega \in \Omega$ .  $\mathbb{P}$  is a probability measure, a function  $\mathbb{P}(A)$  that maps event  $A$  to a number within  $[0,1]$  based on the likelihood of the event.  $\mathcal{F}$  is a  $\sigma$ -algebra on  $\Omega$  if

1.  $\Omega \in \mathcal{F}$
2.  $A \in \mathcal{F}$  implies  $A^c \in \mathcal{F}$
3. if  $A_1, A_2, A_3, \dots$  are in  $\mathcal{F}$  then so is  $A_1 \cup A_2 \cup A_3 \dots$

#### 2.1.2 Conditional Probability

Sometimes we require the probability of an event assuming another event has occurred. In such situations, we require a conditional probability. Given two events  $A$  and  $B$ , the probability of event  $A$  occurring conditioned on event  $B$ :

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}, \quad \forall A \in \mathcal{F} \quad (2.1)$$

From 2.1 and the fact that for dependent events  $\mathbb{P}(A \cap B) = \mathbb{P}(B \cap A)$  we can see that:

$$\mathbb{P}(A \cap B) = \mathbb{P}(A|B)\mathbb{P}(B) = \mathbb{P}(B|A)\mathbb{P}(A), \forall A, B \in \mathcal{F} \quad (2.2)$$

Substituting 2.2 into 2.1 we get the famous Bayes Theorem.

**Theorem 2.2.** *Bayes' Theorem*

For dependent events  $A$  and  $B$  with probability space  $(\Omega, \mathcal{F}, \mathbb{P})$ , where  $\mathbb{P}(B) \neq 0$ ,

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(B|A)\mathbb{P}(A)}{\mathbb{P}(B)}, \forall A \in \mathcal{F} \quad (2.3)$$

### 2.1.3 Stochastic Process

To be able to define a Markov model, of any kind, we must first define a stochastic process.

**Definition 2.3.** Stochastic Process

Given an ordered set  $T$  and probability space  $(\Omega, \mathcal{F}, \mathbb{P})$  a stochastic process is a collection of random variables  $X = \{X_t; t \in T\}$ . Based on  $t \in T$  and  $\omega \in \Omega$  we get a numerical realization of the process. For simplicity, this may be viewed as a function;  $X_t(\omega)$ .

## 2.2 Applied Foundations

# Chapter 3

## Standard Markov and Markov Property

### 3.1 History

Andrei Markov discovered the Markov model while analyzing the relationship between consecutive letters from the text in the Russian novel "Eugene Onegin". A translation can be found here Markov, n.d. With a two-state model (states Vowel and Consonant) he proved that the probability of letters being in a particular state is not independent. Given the current state, he could probabilistically predict the next. With various probabilities to and from each state, this chain of states formed the foundation of the Markov Chain.

### 3.2 Markov Chain

A Markov chain is a network of connected states. At any given time the model is said to be in a particular state. At a regular discrete interval, the model can change states depending on the probabilities. To define a Markov chain, we must first address the Markov property GRIMMETT et al., 2020.

**Definition 3.1.** Markov Property

Let  $\{X_t ; t \in \mathbb{N}_0\}$  denote a stochastic process 2.3, where  $t$  represents time. The process has the Markov property if and only if,

$$\mathbb{P}\{X_{n+1} = i_{n+1} | X_n = i_n, X_{n-1} = i_{n-1}, \dots, X_0 = i_0\} = \mathbb{P}\{X_{n+1} = i_{n+1} | X_n = i_n\} \quad (3.1)$$

We can now define a Markov chain.

**Definition 3.2.** Markov Chain

A stochastic process  $\{X_t ; t > 0\}$  is a Markov Chain if and only if it satisfies the Markov property 3.1.

To store the sequence of states a Markov chain has been through, we use the set  $Q = \{q_t ; t \in \mathbb{N}_0\}$ , where  $q_t$  represents the state at time  $t$ . We will use this notation throughout the paper.

**Example 3.3.** Given a Markov Model with states  $S = \{S_1, S_2, S_3\}$ , if the model starts at  $S_2$  and then goes to  $S_3$  and then back to  $S_2$  the state sequence  $Q$  will be  $Q = \{q_1 = S_2, q_2 = S_3, q_3 = S_2\}$ .



From the Markov property, we can see that the only thing that influences  $q_t$  is  $q_{t-1}$ . We can use this to make predictions for  $q_{t+1}$  based on the outward transition probabilities from state  $q_t$ . By calculating all outward transition probabilities from the state at  $q_t$ , we can construct a probability measure that we may use to predict future states.

i.e.

Given a Markov chain with  $N$  states including  $i$  and  $j$  and discrete time  $t \in \mathbb{N}_0$ , we can use

$$\mathbb{P}(q_t = S_j | q_{t-1} = S_i)_{1 \leq i, j \leq N} \quad (3.2)$$

as a probability measure to help predict future states.

These probabilities can vary with time, but this can make the model quite complicated. Thus, we usually assume the probabilities are constant. These unique Markov models are called time-homogenous.

**Definition 3.4.** Time homogenous

Let  $\{X_t ; t \in \mathbb{N}_0\}$  denote a stochastic process 2.3, where  $t$  represents discrete time, and  $p(i, j)$  represent the transition probability from state  $i$  to state  $j$ . If

$$\mathbb{P}\{X_n = j | X_{n-1} = i\} = p(i, j), \forall n \in \mathbb{N}_0 \quad (3.3)$$

then the process is time-homogenous.

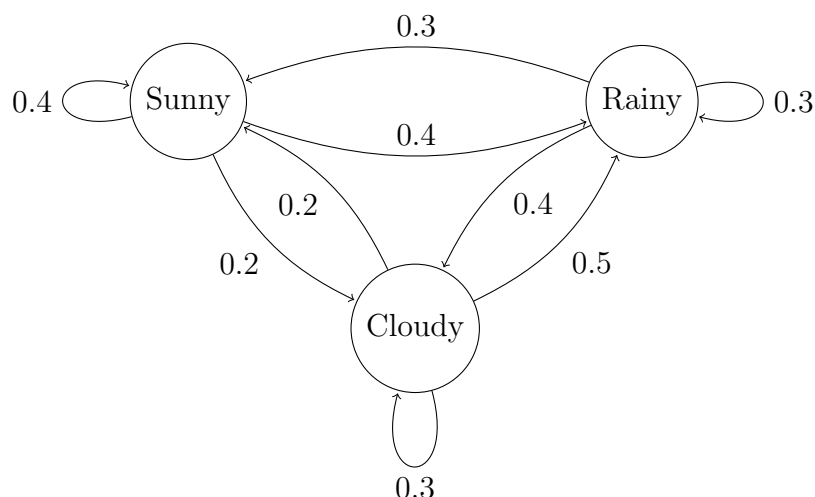
For a discrete Markov model with  $N$  states, there are  $N^2$  total transitions, where  $p(i, j) = 0$  represents an impossible transition. We must store each of these probabilities. Given a time-homogenous Markov chain, we can create a 2-dimensional  $N \times N$  matrix of transition probabilities  $p$ . Unique Markov chains have unique transition matrices. These matrices can be defined as below:

$$p = \{p(i, j) = \mathbb{P}\{X_n = j | X_{n-1} = i\}\}_{1 \leq i, j \leq N} \quad (3.4)$$

All  $p$  matrices have some essential properties. The first is that all values within  $p$  must be within  $[0, 1]$  which comes naturally as all values are probabilities and thus by definition, it must lie within  $[0, 1]$ . The second is that all rows, columns or both form stochastic vectors. If it is the rows, then the matrix is defined as a right-stochastic matrix; if it is the columns, it is called a left-stochastic matrix.

To demonstrate, we will present an example where the states represent the weather.

**Example 3.5.** Let  $\{X_t; t \in \mathbb{N}_0\}$  denote a Markov Chain, with state-space  $S = \{\text{rainy, sunny, cloudy}\}$ , where  $t$  represents the number of days from the start. Since all states can eventually reach all other states, we say this model is ergodic. When all states within a model connect to all others, the model is always ergodic.



Arrows indicated the transition between states and adjacent values correspond to the probability of this transition.

Using 3.5 we can create a matrix  $p$ . To make this clear, we first create a table with our states labelled for rows and columns, where the  $p(i, j)$  is the cell corresponding to row  $i$  and column  $j$ .

x	Sunny	Rainy	Cloudy
Sunny	0.4	0.4	0.2
Rainy	0.3	0.3	0.4
Cloudy	0.2	0.5	0.3

This content of this table forms the matrix  $p$ .

$$p = \begin{bmatrix} 0.4 & 0.4 & 0.2 \\ 0.3 & 0.3 & 0.4 \\ 0.2 & 0.5 & 0.3 \end{bmatrix} \quad (3.5)$$

Now that we have a Markov Model with its  $p$ , we must reflect on its potential uses. Some natural questions one may ask are:

1. Given at time  $t$  the state was  $S_i$ , what is the most likely state time  $t + 1$ ?
2. What is the probability of getting a particular state sequence  $O$ ?
3. What is the probability of staying within a state for  $d$  time steps?

This first problem we will address using the matrix  $p$ . The motivation for creating the matrix  $p$  was to build a matrix where  $p_{ij}$  contains the probability of moving from state  $i$  to state  $j$ . Thus, we look at the current row and find the maximum probability and its corresponding  $j$ .

**Example 3.6.** Let us refer back to 3.5 and assume the current state is Cloudy. We can see that  $0.2 < 0.3 < 0.5$  and that 0.5 corresponds to Rainy. Thus our the most likely next state would be Rainy.

Note: If the current state were sunny, we would have two maximums of 0.4. In such a case, the process is equally likely to go to either.

The second problem provides a state sequence  $Q = \{q_t, q_{t+1}, q_{t+2} \dots\}$  and asks what the probability of this occurring is, i.e.  $\mathbb{P}(Q|model)$ . This can be simplified quite easily as shown below.

$$\mathbb{P}(Q|model) = \mathbb{P}(\{q_t, q_{t+1}, q_{t+2} \dots\}|model) \quad (3.6)$$

$$= \mathbb{P}(q_t)\mathbb{P}(q_{t+1}|q_t)\mathbb{P}(q_{t+2}|q_{t+1})\dots \quad (3.7)$$

$$= \mathbb{P}(q_t)p(q_t, q_{t+1})p(q_{t+1}, q_{t+2})\dots \quad (3.8)$$

**Example 3.7.** We can now use 3.6 to demonstrate the probability of a state sequence from 3.5. Let  $Q = \{\text{Sunny, Sunny, Cloudy, Rainy}\}$ . Given that we start from Sunny:

$$\mathbb{P}(Q|model) = \mathbb{P}(\{\text{Sunny, Sunny, Cloudy, Rainy}\}|model) \quad (3.9)$$

$$= \mathbb{P}(\text{Sunny})\mathbb{P}(\text{Sunny}|\text{Sunny})\mathbb{P}(\text{Cloudy}|\text{Sunny})\mathbb{P}(\text{Rainy}|\text{Cloudy}) \quad (3.10)$$

$$= 1 * 0.4 * 0.2 * 0.5 \quad (3.11)$$

$$= 0.04 \quad (3.12)$$

The third problem asks how long is the model likely to stay in any given state. Assume that the model stays in state  $S_i$  for  $d$  days. We can create a state sequence for this,  $Q = \{q_t = S_i, q_{t+1} = S_i, \dots, q_{t+d-1} = S_i, q_{t+d} \neq S_i\}$ . Using the state sequence probability calculation from before we can compute the following equation:

$$\mathbb{P}(Q|model) = p(i, i)^{d-1}(1 - p(i, i)) = p_i(d) \quad (3.13)$$

We label this  $p_i(d)$  to represent the discrete probability density function of the duration  $d$  in state  $i$ . From this we can calculate the expected stay in any particular state. This is done using the following formula:

$$\bar{d}_i = \sum_{d=1}^{\infty} dp_i(d) \quad (3.14)$$

$$= \frac{1}{1 - p(i, i)} \quad (3.15)$$

**Example 3.8.** Referring back to 3.5, suppose we would like to see how many days in a row we expect it to be sunny. We see that  $p(\text{Sunny, Sunny}) = 0.4$ . Now we can use 3.14 to find,

$$d_{\text{Sunny}}^- = \sum_{d=1}^{\infty} dp_{\text{Sunny}}(d) \quad (3.16)$$

$$= \frac{1}{1 - p(\text{Sunny, Sunny})} \quad (3.17)$$

$$= \frac{1}{1 - 0.4} \quad (3.18)$$

$$= 1.67 \quad (3.19)$$

Thus we expect it to remain sunny for 1.67 days. Since we are dealing with discrete data, it is more appropriate to round up to 2 days.

The Markov model we have been discussing so far is observable as we can observe its events. Not all Markov models, however, are observable.

### 3.3 Motivating the Hidden Markov Model

Sometimes we do not observe our states but only see the effect of a state change. To help develop this idea, we borrow an example from Rabiner and Juang, 1986.

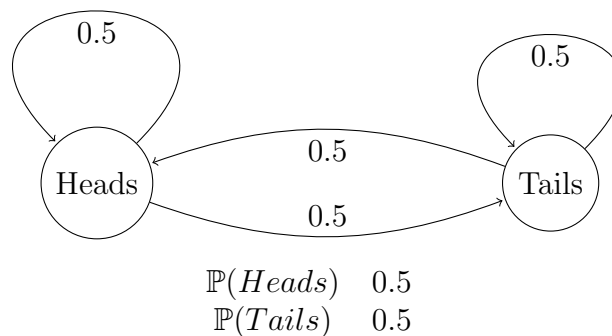
**Example 3.9.** Suppose someone is hiding behind a barrier or curtain, where we cannot see what they are doing. This person is doing some experiment with flipping coins and shouting heads or tails at regular intervals. We do not know:

- i How many coins there are.
- ii If the coin/s is/are fair.

Since the problem is quite vague, we must experiment with various models and see which fits best the data. Since the only thing we can observe is the outcome, heads or tails, we will refer to this as our observation. Let us start with the simplest.

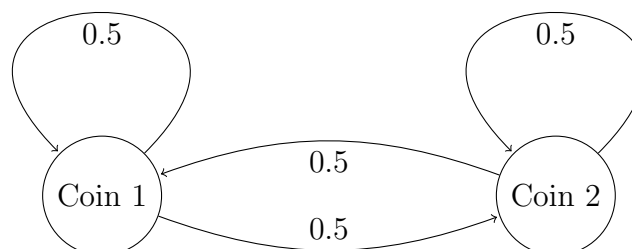
#### 1. 1-Fair Coin

In this model we have two states, heads and tails. There is a 0.5 probability for the model to change state and equally for staying in the same. The simplicity of this model comes at the cost of many assumptions that may not necessarily hold.



#### 2. 2-Fair Coins

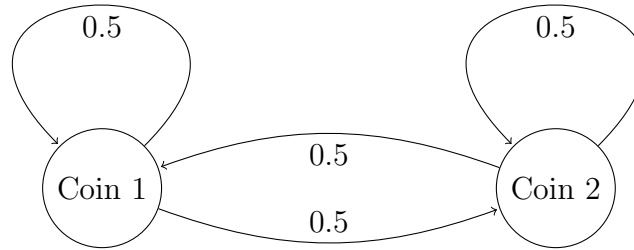
This model also has two states, but this time they represent two different coins. Both can produce heads and tails. Thus we do not know which produced the observation. The fact that they are both fair coins means that as an external observer, we will not see a difference between this model and the 1-Fair Coin. However, it is clear that the observations are now independent of the hidden states transitions, as they are equally likely regardless of the state. In the 1-Fair Coin model, we can determine perfectly which state the model is in from the observation, but here this is no longer possible.



	$\mathbb{P}(Coin1)$	$\mathbb{P}(Coin2)$
$\mathbb{P}(Heads)$	0.5	0.5
$\mathbb{P}(Tails)$	0.5	0.5

### 3. 2-Biased Coins

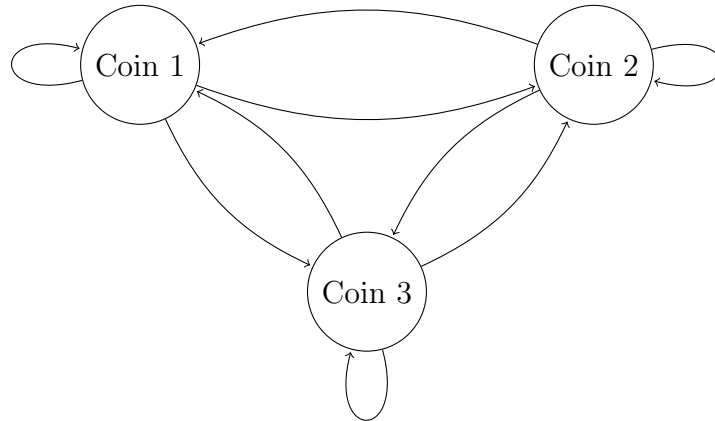
Although this model is similar to 2-Fair Coins, the change in  $\mathbb{P}(Heads)$  and  $\mathbb{P}(Tails)$  for each state has a large impact on the observation probabilities. If the model is in state one heads is more likely and if the state is in state two tails is more likely. As an observer we can now say if we see tails, it was most likely from state two and if we see heads, it was most likely from state one. The change between these two states must be an unrelated probability, such as a third coin or another source of randomness.



	$\mathbb{P}(Coin1)$	$\mathbb{P}(Coin2)$
$\mathbb{P}(Heads)$	0.75	0.25
$\mathbb{P}(Tails)$	0.25	0.75

### 4. 3-Biased Coins

Similar to the 2-Biased Coins model, the probabilities for heads and tails vary with the three states.



	$\mathbb{P}(Coin1)$	$\mathbb{P}(Coin2)$	$\mathbb{P}(Coin3)$
$\mathbb{P}(Heads)$	0.6	0.25	0.45
$\mathbb{P}(Tails)$	0.4	0.75	0.55

After contemplating which model is best, an important conclusion one may make, is that it is quite challenging to decide on the number of states without a priori information. We must ensure that there are enough states, such that the model does not overgeneralize but also not so many that it requires too much data to train. Naturally, one would assume

choosing the larger number of states is more suitable as they can take the shape of a model with fewer states, while it is not for the opposite. However, larger models require much more data to be statistically reliable, as there are too many unknown variables. Below we show this for our models and where the unknowns come from (-s being from the state transition probabilities and -O being from the observation probabilities).

Model	Number of Unknowns	From
1-Fair Coin	0	-
2-Fair Coins	1	1-S
2-Biased Coins	4	2-S 2-O
3-Biased Coins	9	6-S 3-O

Hence, the best approach is to base the model state size on the amount of available data, which is not always guaranteed to give reliable results but can sometimes be the only option. For example, when limited by data.

One may notice that the first model has 0 unknowns. It has this because each state links with only one observation. Thus, as an observer, we can assert what state the model is in from an observation. Thus, the states are no longer hidden, and the model is simply a standard Markov model.

Another essential detail the avid reader may have noticed is in the case of 2-Biased Coins and 3-Biased Coins it is possible to make a reasonable guess of what state the model is in, i.e. which coin is flipped, based on the observation. Furthermore, the statistical properties of predictions generated using 1-Fair Coin and 2-Fair Coins should be identical but for 2-Biased Coins and 3-Biased Coins be somewhat unique. Due to this, unless the underlying system is entirely fair, like in one and two, it should be possible to determine which model best fits the data and determine the model.

We now arrive at the core idea behind hidden Markov models. Although we cannot directly observe the model or its properties, we can create a model that fits the data the best through sufficient observation data.

# Chapter 4

## Hidden Markov Model

### 4.1 Definition

A hidden Markov model is a doubly stochastic process, where the underlying process is markovian and hidden Rabiner and Juang, 1986, which comes from the fact that there are two stochastic processes, one determining the transition between states and one determining the output observation. Braum and his colleagues developed this in Leonard E Baum, Eagon, et al., 1967 and Leonard E. Baum and Petrie, 1966.

To define a hidden Markov model, we need five things.

1. N

- i N is the number of hidden states. Usually based on something in the real world but sometimes can be unknown, as in 3.9.
- ii The states are usually ergodic, i.e. from any given state, we can eventually reach another, but this is unnecessary.
- iii The states are from the state space  $S = \{S_1, S_2, \dots, S_N\}$ .

2. M

- i M is the number of observable outputs.
- ii These make a discrete alphabet of observations called  $V = \{v_1, v_2, \dots, v_M\}$ .

3. A

- i A is the state transition matrix.
- ii This is the same as the  $p$  matrix 3.4.
- iii  $A = \{a_{ij}\}$
- iv  $a_{ij} = \{p(i, j) = \mathbb{P}\{X_n = j | X_{n-1} = i\}\}$

4. B

- i B is the observation probability matrix.
- ii  $B = \{b_j(k)\}$
- iii  $b_j(k) = \mathbb{P}(V_k \text{ at } t | q_t = S_j)_{1 \leq j \leq N, 1 \leq k \leq M}$

## 5. $\pi$

- i  $\pi$  is a vector containing all initial state probabilities.
- ii  $\pi = \{\pi_i\}$
- iii  $\pi_i = \mathbb{P}(q_1 = S_i) \text{ for } 1 \leq i \leq N$

We can now combine the above to provide a formal definition of hidden markov models.

### Definition 4.1. Hidden Markov Model

A Hidden Markov Model is a 5-tuple  $\{N, M, A, B, \pi\}$  that is used to represent a doubly stochastic process where the hidden process is markovian.

Before we continue, we will also provide some notation used for the rest of the paper.

- i We will continue to use  $Q$  3.3 to represent the Markov model's state sequence, i.e. the hidden process.
- ii We will use  $O = \{O_1, O_2, \dots, O_T\}$ ,  $O_i \in V$  to represent the observation sequence.
- iii we will occasionally represent the hidden markov model as  $\{N, M, \lambda\}$  where  $\lambda = \{A, B, \pi\}$

## 4.2 Using HMM

As with any mathematical model, we can use HMMs to predict future observations. As with simple Markov models 3.2, we can create probability measures, but this time we must use conditioning.

### 4.2.1 Predictive Model

We can use HMM to generate a sequence of potential observations. Given a a hidden markov model, lets call it  $H = \{N, M, A, B, \pi\}$ , to generate a sequence of  $T$  observations  $O$  we do the following steps:

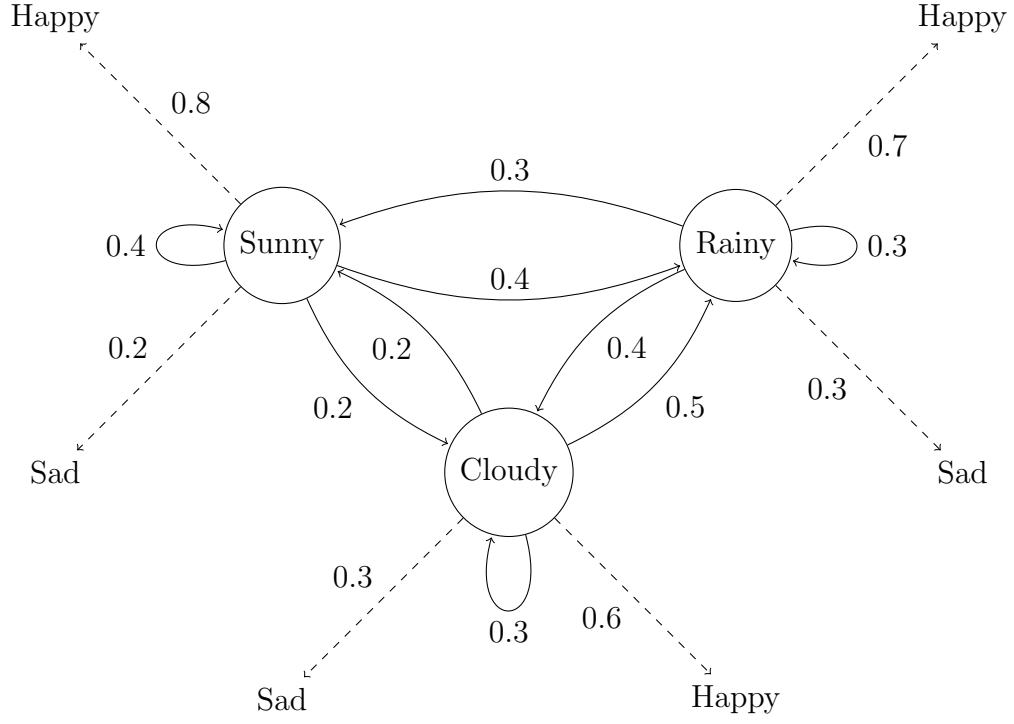
1. Using  $\pi$  as a probability measure, set  $t = 1$  and randomly select a state as the first  $q_1 = s_i$ .
2. Using  $b_i(k)$  as a probability measure, randomly select the observation  $O_t = v_k$ .
3. Using  $a_{ij}$  as a probability measures, set  $t = t + 1$  and randomly select a the next state  $q_{t+1} = s_j$ .
4. Repeat steps 2 and 3 until  $t = T$

To demonstrate this method, we will use an adapted version of 3.5. This version will have the Markov chain from before as the underlying hidden process and another process with state-space  $\{\text{Happy}, \text{Sad}\}$ , which we can observe.

**Example 4.2.** Suppose Alice is hidden away from the world and has no access to information regarding the weather. She meets Bob every day and knows how weather affects his mood. For simplicity, assume Bob only has two moods, happy and sad. Given matrix A, B and vector  $\pi$  can she predict his mood for three consecutive days?

From context we can deduce the following: enumerate





The solid lines represent hidden probabilities, and the dashed represent observable.

$N = 3$ , number of hidden states

$M = 2$ , number of possible observations

$$A = \begin{bmatrix} 0.4 & 0.4 & 0.2 \\ 0.3 & 0.3 & 0.4 \\ 0.2 & 0.5 & 0.3 \end{bmatrix} \quad (4.1)$$

$$B = \begin{bmatrix} 0.8 & 0.2 \\ 0.7 & 0.3 \\ 0.6 & 0.4 \end{bmatrix} \quad (4.2)$$

$$\pi = \begin{bmatrix} 0.5 \\ 0.3 \\ 0.2 \end{bmatrix} \quad (4.3)$$

enumerate

We can now use 4.2.1 to create an observation sequence  $O = \{o_1, o_2, o_3\}$ . We will require multiple random variables generated using various probability measures. We will use these through python using the "rand.py" file. We will use a uniform random variable. We will label this r.v. We select the state that corresponds to the region on the cumulative probability distribution that the random variable lies on.

- i We generate a r.v.  $= 0.0058$ . Using  $\pi$  as the probability distribution, we select Sunny. We can now set  $q_1 = s_1$ .
- ii We generate a r.v.  $= 0.1947$ . Using  $b_1(k)$  as the probability distribution we select Happy as the observation. We can now set  $o_1 = v_1$ .
- iii we generate a r.v.  $= 0.7168$ . Using  $a_{1j}$  as the probability distribution we select Rainy as the next state. We now set  $q_2 = s_2$ .
- iv We generate a r.v.  $= 0.1060$ . Using  $b_2(k)$  as the probability distribution we select Happy as the observation. We can now set  $o_2 = v_1$ .
- v we generate a r.v.  $= 0.8977$ . Using  $a_{2j}$  as the probability distribution we select Cloudy as the next state. We now set  $q_3 = s_3$ .
- vi We generate a r.v.  $= 0.1369$ . Using  $b_3(k)$  as the probability distribution we select Happy as the observation. We can now set  $o_2 = v_1$ .

Finally, we can look back on our prediction  $O$  and see that it is equal to  $\{v_1, v_1, v_1\}$ , i.e. we predict three consecutive Happy days.

### 4.2.2 Three Key Problems

There are many interesting questions one may pose regarding the HMM but there are three famous ones, highlighted in Rabiner and Juang, 1986 which we will focus on.

1. Evaluation

Given model  $H = \{N, M, A, B, \pi\}$  what is the probability that it generated the sequence of observations  $O = \{o_1, o_2, \dots, o_T\}$ ? i.e.  $\mathbb{P}(O | H)$

2. Decoding

What sequence of states  $Q = \{q_1, q_2, \dots, q_3\}$  best explains a sequence of observations  $O = \{o_1, o_2, \dots, o_T\}$ ?

3. Learning

Given a set of observation  $O = \{o_1, o_2, \dots, o_T\}$ , how can we learn the model  $H = \{N, M, A, B, \pi\}$  that would generate them?

In the coming sections, we will be motivating uses and developing solutions for each problem.

## 4.3 Problem 1: Evaluation

Lets start by addressing question 1. Informally, we are looking for the probability that a given model generated a sequence of observations, i.e.  $\mathbb{P}(O|\lambda)$ .

This probability has many useful applications. For example, we may have multiple potential models  $\lambda_i$  and cannot decide which one is the most suitable. In this case, we can now calculate this probability for each  $\lambda_i$  and the largest.

To find this probability, we must consider the hidden internal states of the model. Since our probability of observations  $\{b_j(k)\}$  is conditioned on the hidden state, we can start by

calculating this probability conditioned on these states. Lets assume we know what the state sequence  $Q = \{q_1, q_2, \dots, q_t\}$  is. To  $\mathbb{P}(O|Q, \lambda)$  we can find the product of all the probabilities of an observation given the models state at all times  $t$ . In essence, this breaks the  $\mathbb{P}(O|Q, \lambda)$  into  $T$  parts.

$$\mathbb{P}(O|Q, \lambda) = \prod_{t=1}^T \mathbb{P}(O_t|q_t, \lambda) \quad (4.4)$$

An observation one may make is that these probabilities are simply taken from the matrix  $B$ .

$$\mathbb{P}(O_t|q_t, \lambda) = b_{q_t}(O_t), \quad \forall t \in [0, T] \quad (4.5)$$

Thus we can rewrite 4.4 as:

$$\mathbb{P}(O|Q, \lambda) = b_{q_1}(O_1)b_{q_2}(O_2)...b_{q_T}(O_T) \quad (4.6)$$

Our next objective is to remove  $Q$  from the conditioned part of the probability. To do this, we must first calculate  $\mathbb{P}(Q|\lambda)$ . This is simply the probability of transitioning from  $q_1$  to  $q_2$ ,  $q_2$  to  $q_3$  etc. More formally we can use matrix  $A$  to find the probability of each of these transitions, and since we are finding the total for the entire sequence, we multiply them all together. We start with  $\pi_{q_1}$  as we also need the probability of starting at  $q_1$ .

$$\mathbb{P}(Q|\lambda) = \pi_{q_1} a_{q_1 q_2} a_{q_2 q_3} \dots a_{q_{T-1} q_T} \quad (4.7)$$

We can now successfully remove  $Q$  from the condition using 4.6 and 4.7:

$$\mathbb{P}(O, Q|\lambda) = \mathbb{P}(O|Q, \lambda)\mathbb{P}(Q|\lambda) \quad (4.8)$$

$$= \pi_{q_1} b_{q_1}(O_1) a_{q_1 q_2} b_{q_2}(O_2) a_{q_2 q_3} \dots a_{q_{T-1} q_T} b_{q_T}(O_T) \quad (4.9)$$

This gives us the joint probability of observations and the internal states. In other words, it provides the probability that given observations  $O$  and internal state sequence  $Q$  was generated by model  $\lambda$ . To achieve our desired probability all we need to do is get rid of the  $Q$ . Since it is another input, all we must do is sum each value of 4.8. As we have accounted for every possible  $Q$ , we no longer need to worry about its particular value. This leaves us with:

$$\mathbb{P}(O|\lambda) = \sum_{all Q} \mathbb{P}(O|Q, \lambda) \mathbb{P}(Q|\lambda) \quad (4.10)$$

$$= \sum_{q_1, q_2, \dots, q_T} \pi_{q_1} b_{q_1}(O_1) a_{q_1 q_2} b_{q_2}(O_2) a_{q_2 q_3} \dots a_{q_{T-1} q_T} b_{q_T}(O_T) \quad (4.11)$$

Although this solution is correct, calculating this is infeasible because it requires too many computations. For  $T$  timesteps and  $N$  states, to find every possible  $Q$ , we must sum over  $N^T$  state sequences. For each timestep we require a multiplication to  $a_{q_{i-1} q_i}$  and  $b_{q_i}(O_i)$ , except the last where there are no transitions, which leads to  $2T - 1$  multiplications for each state sequence. Lastly, we require  $N^T$  addition operations to sum the result for each state sequence. Our final total number of operations of  $(2T - 1)N^T + (N^T - 1)$ .

**Example 4.3.** Lets refer back to 4.2. Suppose we would like to find the probability of a string of 10 observations using model H. Here  $N = 3$  and  $T = 10$ .

$$operations = (2T - 1)N^T + (N^T - 1) \quad (4.12)$$

$$= (20 - 1)3^{10} + (3^{10} - 1) \quad (4.13)$$

$$= 1180979 \quad (4.14)$$

We have a problem as even with a smaller model 4.3 we require a considerable number of calculations. For a moderate to large-sized model, we would require an infeasible amount of calculations. To overcome this problem, we look for a more efficient method, the Forward-Backward algorithm.

### 4.3.1 Forward-Backward Algorithm

The Forward-Backward algorithm (F-B) comprises two helper functions  $\alpha$  and  $\beta$ . We will start by discussing the former.

$$\alpha_t(i) = \mathbb{P}(O_1, O_2, O_3, \dots, O_t, q_t = S_i | \lambda) \quad (4.15)$$

$\alpha$  is an extremely powerful tool in reducing the number of calculations. As given by 4.3.1, it provides the probability that at time  $t$  we have seen a sequence of observations and are currently at state  $q_t = S_i$ . This is not quite  $\mathbb{P}(O|\lambda)$  but it represents a part of it. Instead of the probability of the whole sequence, it breaks it into a chunk of size  $t$ , commits to end at a particular state and then calculates the same probability for this.

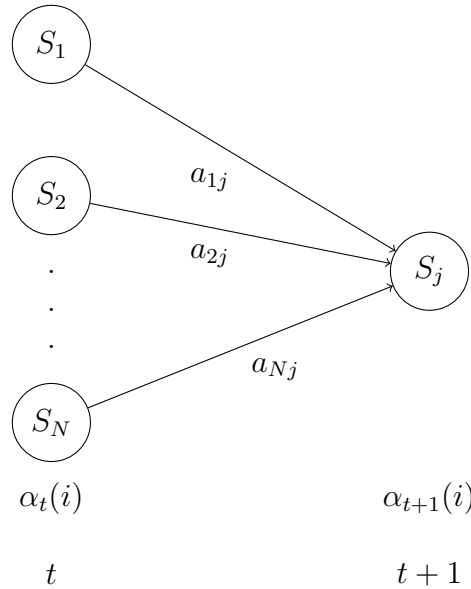
We can combine with induction to produce an iterative process that can calculate ??.

i Base case

For the base case we require the probability of the  $q_1$  being equal to  $S_1$  and giving us observation  $O_1$ . The former is addressed by  $\pi_i$  and the latter by  $b_i(O_1)$ . This gives us:

$$\alpha_1(i) = \pi_i b_i(O_1), \quad i \in [1, N] \quad (4.16)$$

ii Inductive step:



For each state  $j$ , for each inductive step, we follow the above method. Each  $\alpha_t(i)$  is multiplied by each  $a_{ij}$  and then summed up. This result multiplied by  $b_j(O_{t+1})$  gives us  $\alpha_{t+1}(j)$ .

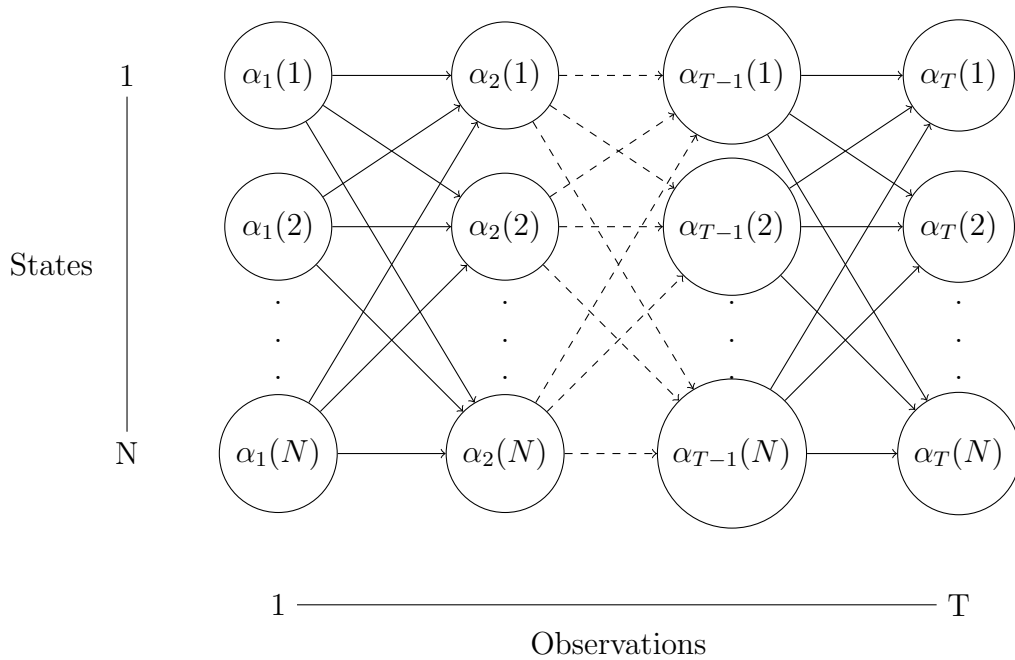
For the inductive step we must consider how to approach the next timestep. We will again be calculating for all  $j \in [1, N]$  and as such must take into consideration, for each  $j$ , every possible  $i$ . This is again the same set of  $[1, N]$ . Therefore, to account for all possible previous states and their transition to the current state, we must sum over 1 to  $N$  the product of  $\alpha_t(i)$  and  $a_{ij}$ . For the given observation, as before, we compute  $b_j(O_{t+1})$ . Additionally, we must stop before reaching the final step as there is no outward transition and thus this would not be applicable. This gives us:

$$\alpha_{t+1}(j) = \left[ \sum_{i=1}^N \alpha_t(i) a_{ij} \right] b_j(O_{t+1}), \quad j \in [1, N], t \in [1, T-1] \quad (4.17)$$

iii Termination step:

For the termination step, we sum over all alpha at the final time  $T$ . Each alpha represents the probability of being in that given state at that particular time, through all possible sequences that it may have followed.

$$\mathbb{P}(O|\lambda) = \sum_{i=1}^N \alpha_T(i) \quad (4.18)$$



For each  $\alpha$  at time  $t$ , we require all  $\alpha$  at time  $t-1$ . Thus we require  $N^2$  calculations for each timestep.

One can see that this method is far more efficient than 4.10. It requires  $N^2$  multiplications for  $\alpha_t(i)$  and  $a_{ij}$  for  $T$  time periods. At each  $T$  there are  $N$  addition operations for the summation and  $N$  multiplication for the  $b_j(O_{t+1})$ , which is true for all but the first and last timestep, where there are  $N$  multiplications and  $N$  additions respectively. This gives us  $(T-2)(N^2 + N + 1) + 2N$  calculations. Considering the order, we see that the previous method had order  $O(TN^T)$  whereas F-B gives us an order of  $O(TN^2)$ . It is now feasible to compute  $\mathbb{P}(O|\lambda)$ .

**Example 4.4.** We attempt to solve the problem 4.3 again but this time using the F-B algorithm. Suppose we would like to find the probability of a string of 10 observations using model H. Here  $N = 3$  and  $T = 10$ .

$$\text{operations} = (T - 2)(N^2 + N + 1) + 2N \quad (4.19)$$

$$= (10 - 2)(3^2 + 10 + 1) + 6 \quad (4.20)$$

$$= 118 \quad (4.21)$$

118 is many orders of magnitude smaller than 1180979, which was the requirement without F-B algorithm. Thus it is a significant improvement. The improvement against the base method will be even more drastic for larger models, as F-B has a fixed  $N^2$  instead of the  $N^T$  term.

For question 1 this is sufficient. However, we will later require the  $\beta$ , the backward component and as such we will describe it now. Logically it is the same principle as the forward component except this time instead of moving forward step by step we are moving backwards.

$$\beta_t(i) = \mathbb{P}(O_{t+1}, O_{t+2}, \dots, O_T | q_t = S_i, \lambda) \quad (4.22)$$

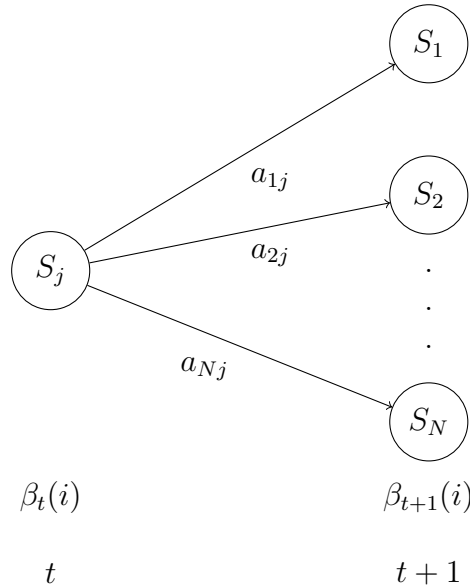
This represents the probability of seeing the observations  $O_{t+1}$  up to  $O_T$  given that at time  $t$  the model  $\lambda$  is at state  $S_i$ . We again calculate this inductively.

1. Base case

For each state we must assume that it is the final state in order to iterate from it. This gives us:

$$\beta_T(i) = 1, \quad i \in [1, N] \quad (4.23)$$

2. Inductive step:



Similar to  $\alpha$ , we use induction. However, this time we look into the future of the sequence and take steps backwards.

Each inductive step we move back by one timestep. As before, we will be calculating for all  $N$  states except this time  $i$  will be varying instead of  $j$ . This makes sense as we are stepping backward and we want to see which previous state is the most likely previous state. At each step, we the transaction probability of having coming from  $i$ ,  $a_{ij}$  to the probability of seeing the given observation at state  $S_j$ ,  $b_j(O_{t+1})$  and the likelihood of being at that state based on future states  $\beta_{t+1}(i)$ . This gives us:

$$\beta_t(i) = \sum_{j=1}^N a_{ij} b_j(O_{t+1}) \beta_{t+1}(j), \quad i \in [1, N], t \in [1, T-1] \quad (4.24)$$

## 4.4 Problem 2: Decoding

We now turn our attention to 2. This problem is somewhat more difficult, as the definition of 'best' is quite vague and open to interpretations. An obvious approach would be to look for the states that are individually most likely for each state for a set time  $t$  given all observations and the model. Lets define this probability as  $\gamma$ .

$$\gamma_t(i) = \mathbb{P}(q_t = S_i | O, \lambda) \quad (4.25)$$

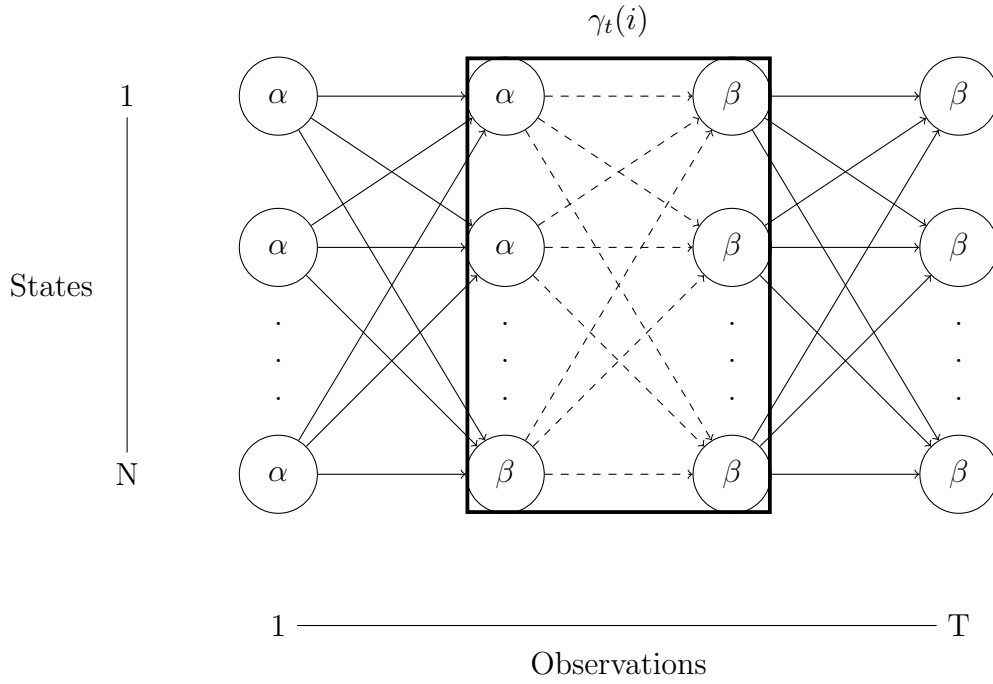
Using 4.3.1 and 4.22 we can solve for  $\gamma$  quite quickly. If we recall,  $\alpha_t(i)$  provides us with the probability of being in state  $i$  after seeing all the previous observations and  $\beta_t(i)$  gives us the probability of being in state  $i$  see all the remaining observations in the future. This gives us  $\alpha_t(i)\beta_t(i)$ . We now normalise this to get a probability, which is possible by dividing by the probability of getting this particular observation given all possible observation sequences  $\mathbb{P}(O|\lambda)$ . This is equivalent to dividing by the sum of  $\alpha_t(j)\beta_t(j)$  over all possible states  $j$ .

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\mathbb{P}(O|\lambda)} \quad (4.26)$$

$$= \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^N \alpha_t(j)\beta_t(j)} \quad (4.27)$$

This ensures that:

$$\sum_{i=1}^N \gamma_t(i) = 1 \quad (4.28)$$



Each dashed line represents a calculation for  $\gamma_t(i)$ . For each time step, we calculate these and then find the maximum. We select the state corresponding to the maximum and add it to the state sequence.

As such this ensures all  $\gamma_t(i)$  are probabilities. We can now go through each timestep  $t$  calculating  $\gamma_t(i)$  and selecting state  $i$  corresponding to the largest  $\gamma_t(i)$ .

This method will maximise the number of correct states and give the correct answer if the model is completely connected, i.e. each state can reach every other state. If this is not the case, we may get a state sequence that is not possible. e.g. At time  $t$  the model is in state  $i$  and at time  $t + 1$  it is in  $j$ , but  $a_{ij} = 0$ , thus is not possible.

In the scenario given, we must redefine 'best' to the path that maximises  $\mathbb{P}(Q|O, \lambda)$ . Since we are maximising this, we can equivalently maximise  $\mathbb{P}(Q, O|\lambda)$ . To solve this problem, we require the Viterbi Algorithm.

#### 4.4.1 Viterbi Algorithm

The Viterbi Algorithm Viterbi, 1967, explored in depth in Forney, 1973 follows a similar lattice structure of the F-B algorithm. Similar to how we broke 1 into smaller pieces, we do the same to  $\mathbb{P}(Q|O, \lambda)$ .

$$\delta_t(i) = \max_{q_1, q_2, \dots, q_{t-1}} \mathbb{P}(\{q_1, q_2, \dots, q_t = i\}, O_1, O_2, \dots, O_t | \lambda) \quad (4.29)$$

An important point to note is  $\delta_t(i)$  does not store the sequence. Thus we must introduce a new variable that will be responsible for doing so. We can call this  $\psi_t(i)$ , where it is equal to the state we have come from given we are at time  $t$  and state  $i$ . We will once again use induction. The process is as follows.

1. Base case

For each state we must calculate  $\pi_i b_i(O_1)$  to determine which initial state is most likely.



We also must set  $\psi_1(i) = 0$  as there have not been any states until now.

$$\delta_1(i) = \pi_i b_i(O_1) \quad (4.30)$$

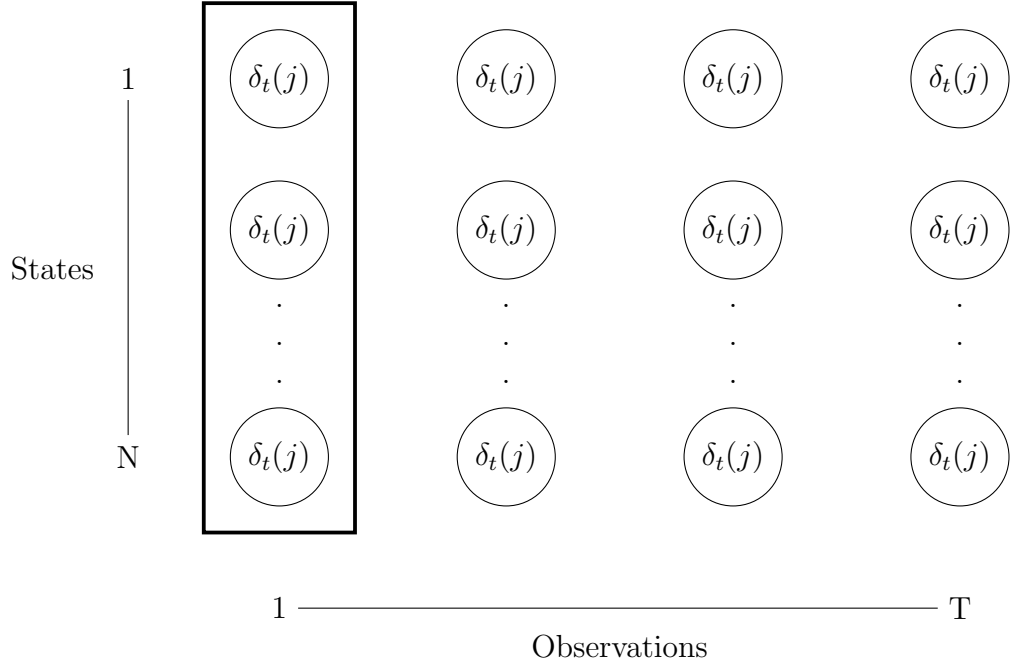
$$\psi_1(i) = 0 \quad (4.31)$$

2. Inductive step:

For each state we calculate the  $\delta_{t-1}(i)$  times  $a_{ij}$  to find the most likely next state based on previous state likelihood and the transition probability. We maximize this term and then multiply by  $b_j(O_t)$  to include a bias based on the observation probabilities. As before we store the argument of the maximum in  $\psi$ . This gives us:

$$\delta_t(j) = \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}] b_j(O_t), t \in [2, T] \quad (4.32)$$

$$\psi_t(j) = \operatorname{argmax}_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}], j \in [1, N] \quad (4.33)$$



Once  $\delta_t(j)$  is calculated for a particular  $t$ , we find the maximum, which is the state we add to our state sequence. Since  $\delta_t(j)$  accounts for transition probability  $a_{ij}$ , impossible transitions will always be equal to 0. Thus unless all transitions are equal to 0, they cannot be the maximum.

3. Termination step:

To terminate this recursion, we need to find the maximum  $\delta_T(i)$ . Here we maximise, as the values we already calculated the values in the induction. We will store this probability in  $\mathbb{P}^*$ . Similarly, we need to find the argument corresponding to this max for the final state, and we set this to the final state  $q_T^*$ .

$$\mathbb{P}^* = \max_{1 \leq i \leq N} [\delta_T(i)] \quad (4.34)$$

$$q_T^* = \operatorname{argmax}_{1 \leq i \leq N} [\delta_T(i)] \quad (4.35)$$

To find any particular states that we may be interested in  $q_t^*$ , we use  $q_t^* = \psi_{t+1}(q_{t+1}^*)$ .

## 4.5 Problem 3: Learning

The third problem 3 addresses learning. How can we learn a model from given data of observations? This problem is critical as we often do not have the model, and calculating it is not simple. In these cases, we must derive the model from the data we have, a set of observations. Given the set of observations  $O = \{O_1, O_2, \dots, O_T\}$  we want to find the most suitable  $\lambda = (A, B, \pi)$ . To do this, we use the Baum-Welch algorithm.

### 4.5.1 Baum-Welch Algorithm

For the Baum-Welch algorithm we will need all three parameters introduced earlier;  $\alpha$  4.3.1,  $\beta$  4.22 and  $\gamma$  4.25. We will also need a new parameter  $\xi_t(i, j)$ . This will capture the probability of being in some state  $S_i$  at time  $t$  and then  $S_j$  at time  $t + 1$  given the observations and the model.

$$\xi_t(i, j) = \mathbb{P}(q_t = S_i, q_{t+1} = S_j | O, \lambda) \quad (4.36)$$

To solve this problem we can refer back to 4.3.1 for the left hand side (including  $q_t$ ) and 4.22 for the right hand side (including  $q_{t+1}$ ). To get from state  $i$  to  $j$  we use  $a_{ij}b_j(O_{t+1})$ . Putting this all together we get a likelihood. To normalize this into a probability we once more use  $\mathbb{P}(O|\lambda)$ . This gives us:

$$\xi_t(i, j) = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{\mathbb{P}(O|\lambda)} \quad (4.37)$$

Since we are interested in the probability of going from one  $i$  to one  $j$ , for the denominator we use the sum of all  $i$  and all  $j$ .

$$\xi_t(i, j) = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^N \sum_{j=1}^N \alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)} \quad (4.38)$$

One may find it helpful to compare  $\xi_t(i, j)$  with  $\gamma_t(i)$ .  $\gamma_t(i)$  represents the probability of being at state  $i$  at timestep  $t$ . Summing over  $j$  for  $\xi_t(i, j)$  leaves us with  $\xi_t(i)$  which represents the probability of being at state  $i$  at timestep  $t$ . This is identical to  $\gamma_t(i)$ . We can now state:

$$\gamma_t(i) = \sum_{j=1}^N \xi_t(i, j) \quad (4.39)$$

For  $\gamma_t(i)$ , if we sum over  $t$  we can get a number that can be used as the expected number of times  $S_i$  is visited. Thus the expected number of transitions from  $S_i$  can be calculated by:

$$\sum_{t=1}^{T-1} \gamma_t(i) \quad (4.40)$$

Similarly for  $\xi_t(i, j)$ , if we sum over  $t$  we can get a number that can be used as the expected number of times  $S_i$  transitions to  $S_j$ . Thus the expected number of transitions from  $S_i$  to  $S_j$  can be calculated by:

$$\sum_{t=1}^{T-1} \xi_t(i, j) \quad (4.41)$$

Now that we have created a set of tools, we will need to tackle the learning problem itself. To build a improved  $\lambda$ ,  $\bar{\lambda}$ , we need to find  $\bar{\pi}$ ,  $\bar{A}$  and  $\bar{B}$ .

i  $\bar{\pi}$

Since  $\gamma_t(i)$  represents the expected frequency in  $S_i$  at time  $t$ , if we let  $t=1$  this now is equivalent to  $\bar{\pi}$

$$\bar{\pi} = \gamma_1(i) \quad (4.42)$$

ii  $\bar{a}_{ij}$

Here we can use the expected number of transitions from state  $i$  to  $j$  divided by expected number of transitions from  $i$  in total. This will provide the appropriate transition probability.

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \quad (4.43)$$

iii  $\bar{b}_j(k)$

For the B matrix, we will need to divide the expected number of times the model is in state  $j$  and observes  $v_k$  by the expected number of times it is  $j$ .

$$\bar{b}_j(k) = \frac{\sum_{t=1, s.t. O_t=v_k}^T \gamma_t(j)}{\sum_{t=1}^{T-1} \gamma_t(j)} \quad (4.44)$$

We have used our  $\lambda$  and  $O$  to produce our parameters:  $\alpha_t(i)$ ,  $\beta_t(i)$ ,  $\gamma_t(i)$ ,  $\xi_t(i, j)$ . This is called parameter re-estimation. We have then used these parameters to produce our  $\bar{\lambda} = \{\bar{\pi}, \bar{a}_{ij}, \bar{b}_j(k)\}$ . This is called expectation maximization. This method can be iterated and each iteration should provide an improvement for lambda or worst case senario be equivalent. Proof of this can be found in Leonard E Baum et al., n.d. Once lambda stabilizes and is no longer changing, we can assume a local optimum has been reached.

## 4.6 Modified HMM

### 4.6.1 GMM

# Chapter 5

## Model Selection / Parameter Estimation

Two models can use extremely different techniques to predict the same thing. Thus being able to compare these models is vital. Even if the models are the same, for parameter estimation, we must compare the Performance of the model with multiple sets of parameters. To do so we need a metric that we can compare across all models and parameter estimations. This metric is given by the likelihood function.

**Definition 5.1.** Likelihood Function

Given model  $\theta$  and the observation  $x_1, x_2, \dots, x_n$ , the likelihood function is given by

$$L(\theta|x_1, x_2, \dots, x_n) = \mathbb{P}(x_1, x_2, \dots, x_n|\theta) \quad (5.1)$$

In other words it is given by the probability that the given model produced the given observations.

This function allows us to directly compare which model has a higher probability of fitting the system, based on the data. To do so we use Maximum likelihood estimation.

### 5.1 Maximum Likelihood Estimators

Through maximizing the likelihood function 5.1, we can find the model that best fits the given observations. This process is called Maximum Likelihood estimation.

We often use  $\log(L(\theta|x_1, x_2, \dots, x_n))$  instead of  $L(\theta|x_1, x_2, \dots, x_n)$  as this often leads to simpler differentiation. This is possible since the log function is monotonically increasing thus they have their maximum at the same value of  $\theta$ .

The method can be described as follows:

- i Calculate the joint probability density of observing your data  $X_1 = x_1, \dots, X_n = x_n$  given your model  $\theta$ , i.e.

$$\mathbb{P}(X_1 = x_1, \dots, X_n = x_n|\theta) \quad (5.2)$$

- ii Take the Natural logarithm of both sides.

$$\log(\mathbb{P}(X_1 = x_1, \dots, X_n = x_n|\theta)) \quad (5.3)$$

- iii Take the partial derivative with respect to each parameter.
- iv For each parameter, set the derivative equal to 0 and rearrange for its value. The given value will be the maximum likelihood estimation of that parameter.

Unfortunately in real world applications the derivative of the log-Likelihood is analytically intractable. This is particularly true for Hidden Markov based models. Thus we often require alternative methods to compare models.

## 5.2 Approximate Bayesian Computation

Approximate Bayesian Computation (ABC) is an alternative method of parameter estimation and model Selection. Using Bayesian inference, we calculate the posterior distribution, one containing information from both the data and the prior distribution.

We use Bayes' original theorem applied to point probabilities and observations  $y$  to obtain the following:

$$\pi(\theta|y) \propto \pi(\theta)f(y|\theta) \quad (5.4)$$

where  $\pi(\theta|y)$  is the probability distribution of the parameters given the data (posterior distribution),  $\pi(\theta)$  is the probability distribution of the parameters before we take into account the evidence (prior distribution) and  $f(y|\theta)$  is the likelihood.

Instead of calculating the probability directly, using some preset parameters ABC creates a simulation and then compares this to the observation dataset. The method can be described as follows: \*\*\*\*\*REFERENCE\*\*\*\*\*

If  $N$  iterations are desired for  $i=1$  to  $N$ :

1. Generate simulation  $\theta'$  from the prior distribution  $\pi(\text{data})$ .
2. Generate  $Z$  (new value for parameter/s) from the likelihood  $f(\text{data}|\theta')$
3. If distance between statistic  $\eta$  of  $y$  and  $z$  is less than or equal to some  $\epsilon$  set  $\theta_i = \theta'$  else repeat until condition is fulfilled.

Given a small enough tolerance ( $\epsilon$ ) and appropriate statistic  $\eta$  we have:

$$\pi_\epsilon(\theta|y) = \int \pi_{\epsilonpsilon}(\theta, z|y) dz \approx \pi(\theta|y) \quad (5.5)$$

as the actual posterior distribution is the limit of 5.5 as  $\epsilon$  goes to 0.

## 5.3 Kolmogorov–Smirnov test

Kolmogorov–Smirnov test

# Chapter 6

## Replicating Existing Rainfall Models

### 6.1 Coppertwait

Coppertwait's rainfall models have been used for decades with a good deal of success. Although we will not dive deep into his methodology the following papers may be referenced if one wishes to **ref all coppertwait papers**.

Over time, Coppertwait has improved and adjusted his model for varying applications. For example, in one of his more recent papers **ref 2010 paper**, Coppertwait has turned the discrete variable describing the type of storm into a continuous one. These changes and developments, while interesting, are not the focus of our paper. Instead, we will focus on **ref 1994 paper** to understand the model at its simplicity and one may apply these ideas to a Hidden Markov Model.

All of Coppertwait's models use a Neyman-Scott point processes to describe the arrival of Storms. The model itself can be described as follows:

- The origin of each storm is a Poisson process in space-time
- Each storm is a circular region in two dimensional space with radii given by independent exponential random variables.
- Each storm generates a cluster of two dimensional circular rain cells with random duration, position and radius.
- The duration of each cell is an exponential random variable.
- The intensity of each cell is a random exponential distributed but dependent on the type of cell. In **ref 1994** we have two cell types, light and heavy.
- Overall intensity at any point is given by the sum of overlapping intensities at the point in space and time.

The GNSRP Model, discussed in **ref paper 1994**, is a generalisation of the Neyman-Scott-Rectangular-Pulse (NSRP) Model with only one cell type. The key feature of the NSRP is that, in addition to those above, the distance from Cell origins to its Storm origin are independently exponentially distributed. Unlike other models that used an independent uniform distribution for this, NSRP accounted for the realistic effect that rainfall tends to be more intense near the center of the storm **ref browning**.

## 6.2 Grando - HMM Based Model

### 6.2.1 Model

Grando extended on Coppertwait's ideas in **\*\*ref paper\*\***. She completely removes the double disc structure and attempts to enclose the correlations within the larger Storm discs from Coppertwait in Hidden Markov States.

She defines her discrete time intensity process as follows: For  $n \geq 1$  and  $x \in A$ , where  $A \subset \mathbb{R}^2$

$$I_n(x) := \sum_{i=1}^{N^{(n)}} \sigma_i^{(n)} \mathbb{I}_{D(x_i^{(n)}, R_i^{(n)})}(x) \quad (6.1)$$

where:

- $I_n(x)$  is the rain intensity
- $N^{(n)} \sim \text{Pois}(\lambda)$  represents the number of rain discs generated
- $\sigma_i^{(n)} \sim \text{Exp}(\tau)$  is a random variable providing the rain intensity
- $D(x_i^{(n)}, R_i^{(n)})$  is a rain disc with centre  $x_i^{(n)} \sim \text{Unif}(A)$  and radius  $R_i^{(n)} \sim \text{Exp}(\xi)$ .
- $\mathbb{I}_{D(x_i^{(n)}, R_i^{(n)})}(x)$  is an indicator function, equal to 1 if the current rain disc overlaps position  $x$ .

Parameters  $\lambda, \xi$  and  $\tau$  are taken to be fixed for each type of storm. In other words, for each hidden state of the HMM there is a unique  $\lambda, \xi$  and  $\tau$ . Informally, the model has hidden states which determine the type of storm. This then dictates the values of the three above parameters, which in turn influence the intensity of rain observed.

Thus we arrive at the learning problem. Due to the added complexity of additional parameters we are unable to directly use methods for HMMs discussed earlier in the paper. Direct likelihood calculation is complex for HMMs alone, after we introduce these additional parameters it is not feasible.

Given the model has  $m$  states, we will have  $m^2 + 4m$  unknown parameters,  $m^2$  for the transition probabilities,  $m$  for the initial distribution and  $m$  for each of  $\lambda, \xi$  and  $\tau$ . This is clearly a high-dimensional problem. To overcome this Grando fixes certain parameters in order to gain a better understanding of the remaining. She uses two methods; Nealder-Mead and Approximate Bayesian Computation. We will focus on the latter as it has provided superior results in her testing.

### 6.2.2 Model Fitting

We begin by trying to replicate Grando's results in order to establish a starting point. For ABC we require the prior distribution along with a summary statistic. Deriving the prior distribution is not a trivial problem. We not only have parameters for the HMM as well as parameters for the observations but have to extract both from only the intensity values.

Grando determines that even without the HMM parameters, the problem is high dimensional ( $3m$  parameters). Due to this she decides to fix all HMM parameters to uniform(0,1) for each test, whilst insuring the transition matrix is stochastic through normalization of rows.

She believes that this will allow her to retrieve close approximations of all  $\lambda, \xi$  and  $\tau$  from the data, which she can then use to calculate the HMM parameters.

For each simulation she will generate a new value for all  $\lambda, \xi$  and  $\tau$ . Then plot the summary statistic against the values for each and then find areas of high concentration to decide what a suitable approximation for the true value of each is. Here she is setting her ABC tolerance to  $\infty$  and then manually narrowing her range through the graph.

Grando picks values for each parameter using the uniform distribution between preset values. She says "This technique is purely heuristic and it is only aimed to explore the problem, as decisions made are completely arbitrary.". Theoretically we could pick any values for but for consistency we will match the values she has used.

She decides to use the mean intensity as her summary statistic for ABC. She calculates this for both the simulated and given data. She then calculates the normalised Euclidean distance between the two as her metric for model performance.

## 6.3 Numerical Results

We will be using the same algorithm, same data and same parameters as Grando to ensure consistency. Our data measures rainfall across 30 sites in Germany over the period from 1st of January 1931 to 31st December 2000. Co-ordinate data for each site is also given in an additional file. These are labeled as Easting and Northing.

Grando fixes  $m=3$ , allowing for a low, medium and high state for rainfall. This gives us 21 unknown parameters of which we are attempting to estimate 9.

All programs have been run on the personal desktop (AMD Ryzen™ 9 3900X with 12 Cores and 24 Threads running from 3.8GHz up to 4.6GHz) using C++20.

### 6.3.1 Code

To ensure validity of results, we wanted to run multiple tests. Grando's code was written in R and required multiple hours to produce a result. To improve on this we have rewritten the program in C++ and used Python on Jupyter Notebooks for analysis.

Data preparation took place in excel. For simplicity in importing into C++, the data needed to be configured in a particular way. This is described thoroughly in the notes given with the code.

C++ was used for the actual simulation as this was the slowest part of Grandos algorithm. Many parameters have been hardcoded. This is to allow for efficiency and can later be altered if needed. Notes on manipulating the code can be found *\*\*reference to later\*\**. Multi-threading has been used for further reduction in run-time. Admittedly the code can be further paralised but in its current state it is acceptable; run times of under 5 minutes.

Analysis was done using Python in Jupyter Notebooks. The C++ program created multiple CSV files as output and these were taken as input for analysis.

### 6.3.2 Results - Nine Parameter Estimation

We begin with using ABC to estimate all parameters, found on pages 75-76 *\*\*Grando ref\*\**. The parameters are picked as follows:

- $\lambda_1 \sim \text{Unif}(5,15)$ ,  $\lambda_2 \sim \text{Unif}(16,25)$  and  $\lambda_3 \sim \text{Unif}(26,35)$



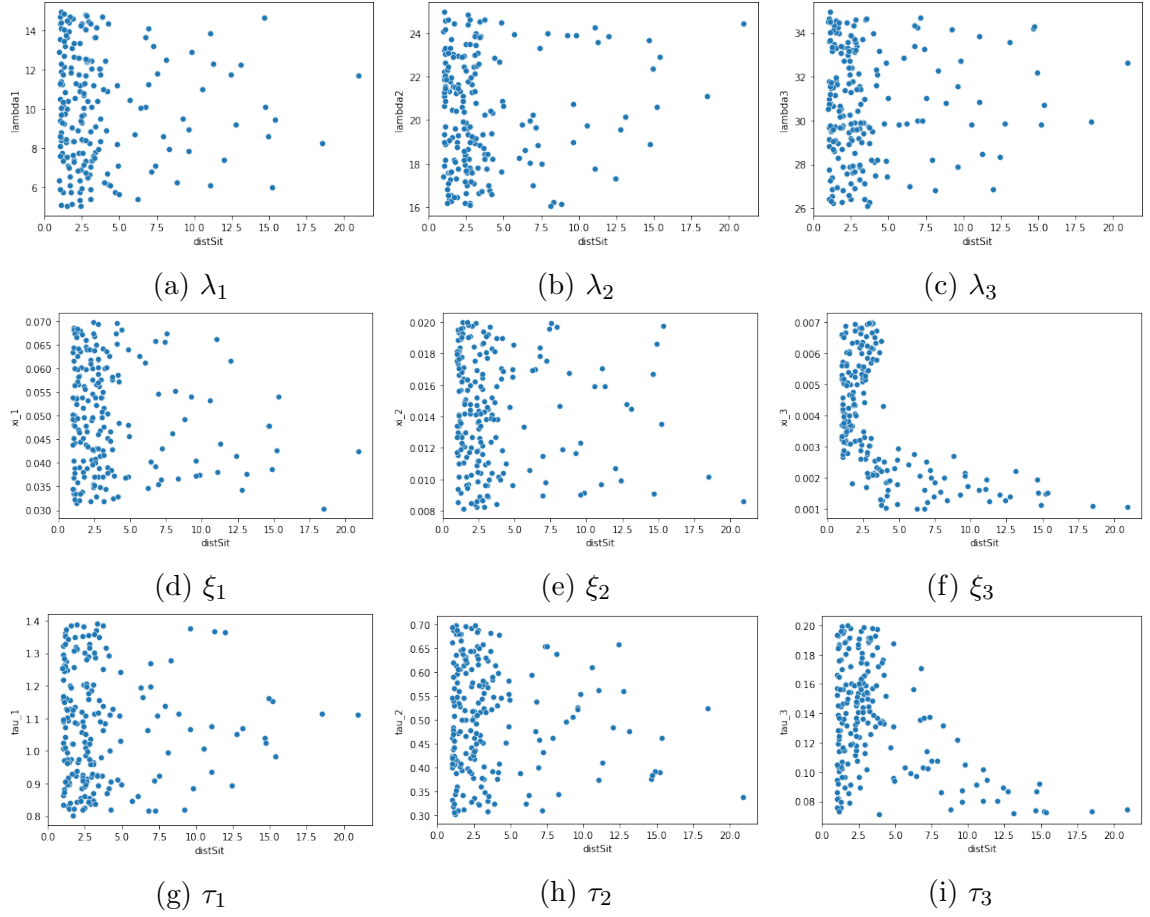


Figure 6.1: Parameter values plotted against corresponding normalised Euclidean Distance between simulation and data means.

- $\xi_1 \sim \text{Unif}(0.03, 0.07)$ ,  $\xi_2 \sim \text{Unif}(0.008, 0.02)$  and  $\xi_3 \sim \text{Unif}(0.001, 0.007)$
- $\tau_1 \sim \text{Unif}(0.8, 1.4)$ ,  $\tau_2 \sim \text{Unif}(0.3, 0.7)$  and  $\tau_3 \sim \text{Unif}(0.07, 0.2)$

For each simulation we generate data for 365x5 days and calculate the normalised Euclidean distance between the mean of these and the mean of the whole given dataset. Due to random sampling we attempt to find estimates for all 9 parameters simultaneously. We completed 200 iterations on the personal desktop in 42.2841 seconds.

The results show ABC on nine parameters was not successful. Figure 6.1 shows for all parameters an extremely uniform scatter suggesting the posterior distribution is still uniform in a neighbourhood of the true parameters. This is an exact match to Grando's results.

### 6.3.3 Results - Three Parameter Estimation

Grando then decides to perform ABC on a subset of our nine parameters. She claims choosing only  $\lambda_i$ , only  $\xi_i$  or only  $\tau_i$  does not yield better outcomes. She then goes on to test for only  $\lambda_3, \xi_3$  and  $\tau_3$ , fixing all other parameters to arbitrary values. Once again, for consistency we choose values to match those of Grando.

- $\lambda_1 = 10$ ,  $\lambda_2 = 20$  and  $\lambda_3 = 30$

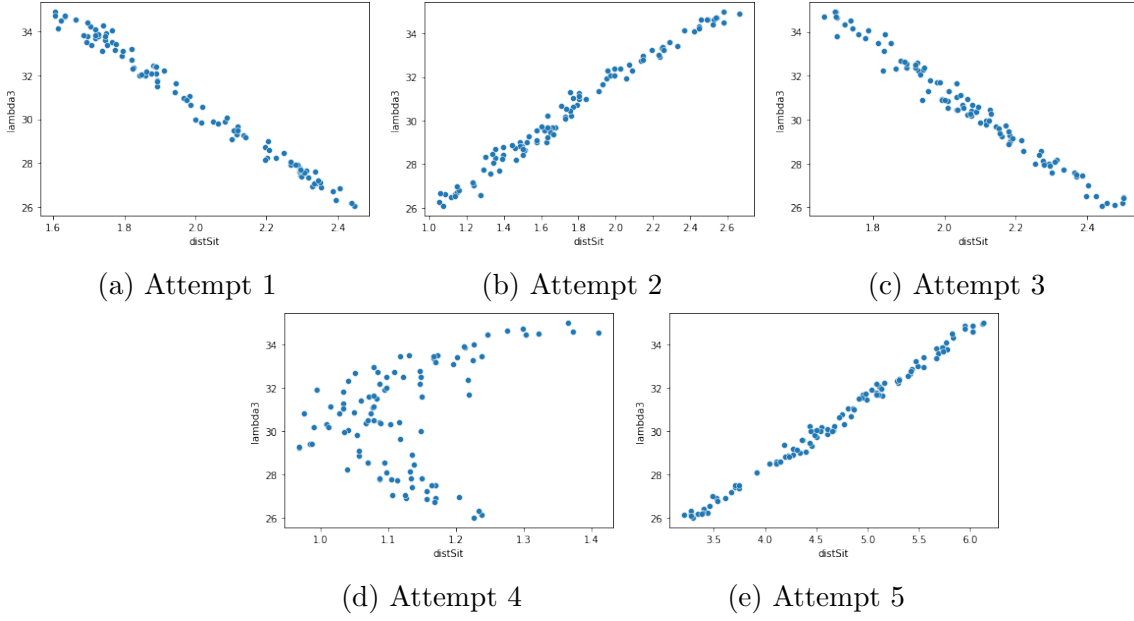


Figure 6.2:  $\lambda_3$  values plotted against corresponding normalised Euclidean Distance between simulation and data means for 5 attempts.

- $\xi_1 = 0.05$ ,  $\xi_2 = 0.01$  and  $\lambda_3 = 0.005$
- $\tau_1 = 1.1$ ,  $\tau_2 = 0.5$  and  $\tau_3 = 0.1$

For each test we require three simulations. When computing for  $\lambda_3$  we change its value from 30 to  $\lambda_3 \sim \text{Unif}(26,35)$ . Similarly, when computing for  $\xi_3$ , we change its value to  $\xi_3 \sim \text{Unif}(0.001,0.007)$  and when computing for  $\tau_3$  we change its value to  $\tau_3 \sim \text{Unif}(0.07,0.2)$ . At any given time there is one random variable between our nine that we are testing. For efficiency we process this over multiple threads. We completed 100 iterations and ran 5 attempts on the personal desktop. Attempt time ranged between 152.725 in to 199.386 seconds.

The graphs found in Figures 6.2, 6.3 and 6.4 correspond to the ones found in Grando's \*\*\* REF\*\* paper pages 78-80. We have come accross unexpected results. Although the graphs show a similar pattern, there is an inconnsistency in values.

For  $\lambda_3$  Grando's estimate was 80.436. This was after running the algorithm in a larger interval. Although we have no done this, our data is sufficient to draw conclusions. We are expecting a minimum distance, in other words we expect the graph be a parabola in y; the parameter values.

- The negative slope of 6.2a and 6.2c suggests the true value is larger than our upper limit of 35.
- The positive slope of 6.2b and 6.2e suggests the true value is lower than our lower bound of 26.
- 6.2d suggests the true value is somewhere within our interval of  $[26,35]$ .

It is clear that not all three can be true as each contridicts the other. This suggests the value of  $\lambda_3$  is changing through new attempts.

For  $\xi_3$  Grando's estimate was 0.0027. From our results we can deduce the following:

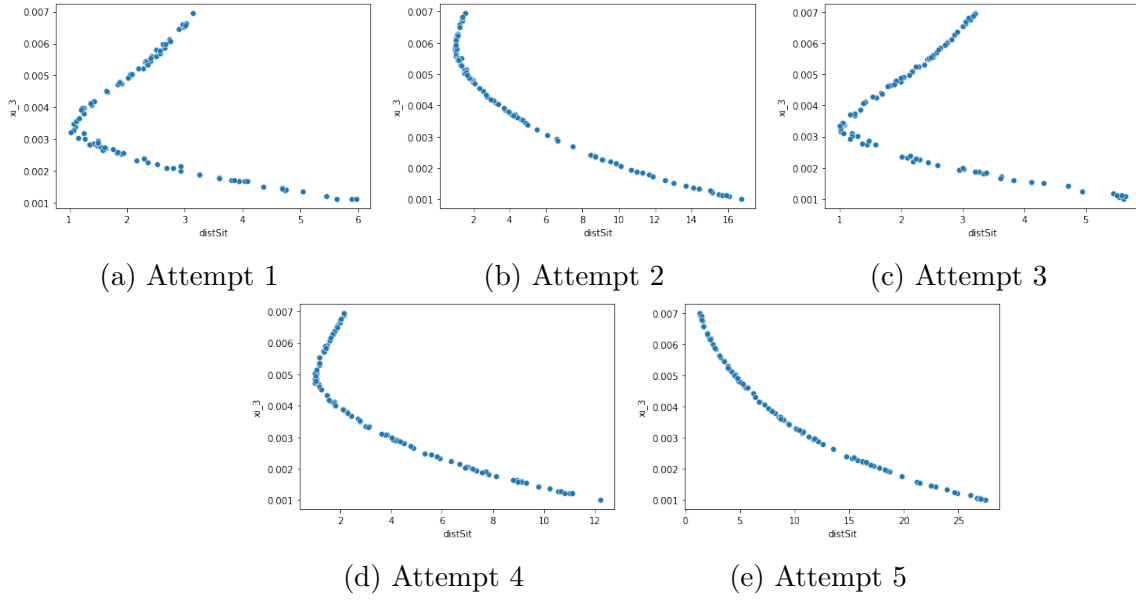


Figure 6.3:  $\xi_3$  values plotted against corresponding normalised Euclidean Distance between simulation and data means for 5 attempts.

- Both 6.3a and 6.3c suggest the true value is around 0.003.
- 6.3b and 6.2d suggest the true value is between 0.005 and 0.006.
- The negative gradient of 6.3e suggests the true value larger than our upper limit of 0.007.

Although it seems the value remains within the interval most of the time, it does not remain consistent. Again, as with  $\lambda_3$ , these changes in estimated value suggest the true value may be changing through new attempts.

For  $\tau_3$  Grando's estimate was 0.0386. From our results we can deduce the following:

- The positive slope of 6.3a and 6.3c suggest the true value is below our lower bound of 0.07.
- 6.4b suggests the true value is around 0.13.
- 6.4d suggests the true value is around 0.10.
- The negative slope of 6.4e suggests the true value is above our upper limit of 0.2.

Once again, multiple attempts provide contradicting results. This suggests the true value is changing with each attempt.

### 6.3.4 Adjusted ABC Algorithm

From all 6.2, 6.4 and 6.3 we have the same indication; the parameter values are changing for new attempts. To justify this we must isolate the cause of this change. Our model requires our nine parameters to be constants. Furthermore, we have controlled the values of each in our testing, thus any change in parameter value must be caused by external factors. The only other factors within the model are the hidden markov parameters.

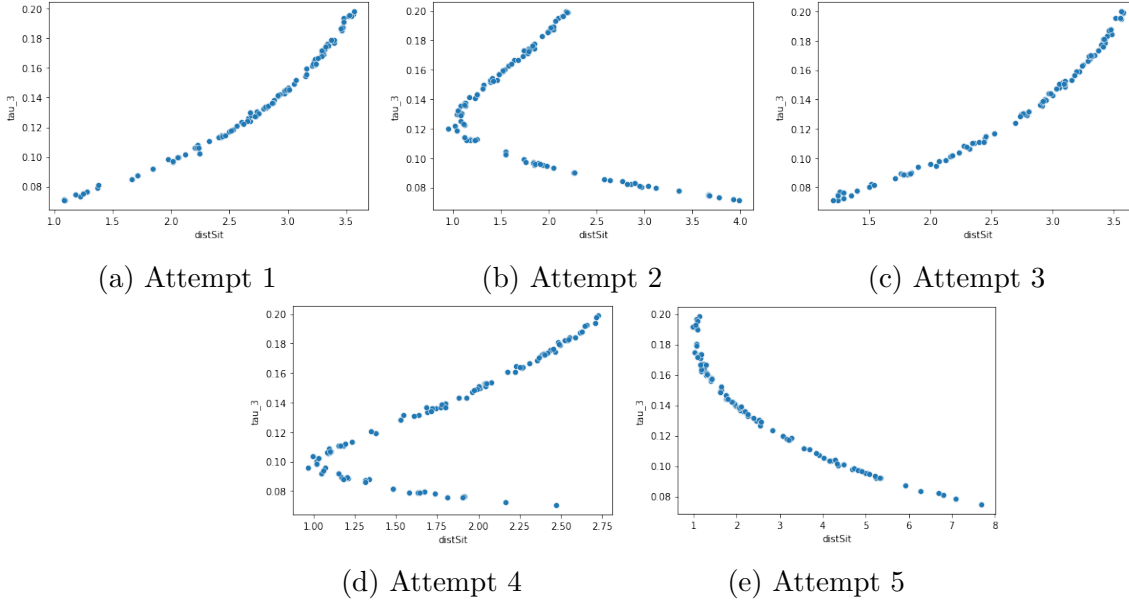


Figure 6.4:  $\tau_3$  values plotted against corresponding normalised Euclidean Distance between simulation and data means for 5 attempts.

Looking closely at our simulation algorithm, one may notice for each attempt the HMM parameters are generated by a uniform random variable and then kept the same for each simulation. In other words, they only change on new attempts. When we estimate the parameters using ABC, we are conditioning on the parameters values we have preset; in particular the HMM parameters. Since these vary for each attempt, the condition our algorithm is based on varies for each attempt. Thus, we can confirm the model is different for each attempt and the parameters estimates are in fact also different for each attempt. We can conclude ABC with the given algorithm has failed to provide parameter estimates. The increase in efficiency of the ABC algorithm has allowed for multiple attempts which has highlighted a flaw in the algorithm itself.

We can further highlight this issue through an adjustment to the ABC algorithm. We move the generation of HMM parameters to within the iterative process. This forces a new set of HMM parameters for each simulation. Whilst this would not be accurate to the model, this randomisation prevents the method providing parameter estimates that fit to a particular set of HMM parameters. We use this for three parameter estimation. We increase the number of iterations to 1000, to ensure we can find a pattern if there is one as there is an increase in randomisation. Using the same parameter values as 6.3.3, computation on the personal desktop took 652.842 seconds.

As we can see from Figures 6.5, we have a somewhat uniform scatter. This suggests we are unable to find a parameter estimate for our non-HMM parameters without fixing the HMM parameters. Thus we must search for an alternative method of parameter estimation. Our method must support high-dimensional parameter estimation in our current model we have 21 unknowns.

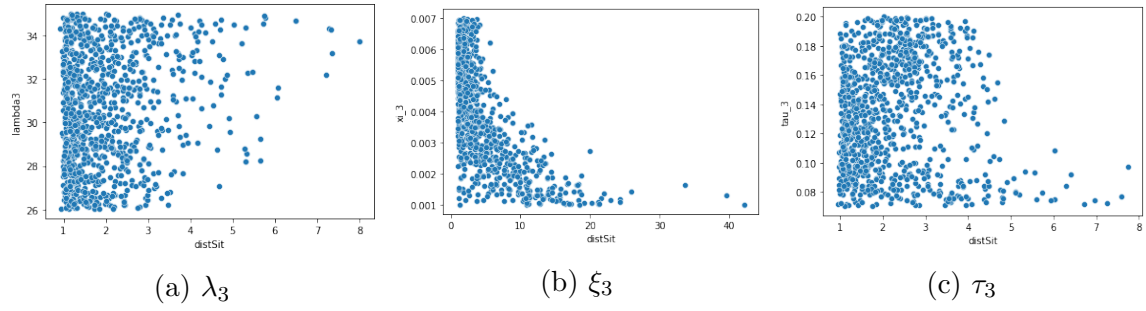


Figure 6.5: Parameter values plotted against corresponding normalised Euclidean Distance between simulation and data means for simulation calculated using adjusted ABC algorithm

# Chapter 7

## Extending HMM based Rainfall Models

In this chapter we will extend on Grando's ideas and adapt her model.

### 7.1 Disk Structure

Cowpertwait's disk structure for rain models is derived from nature. This makes it extremely appealing as a model. Giving Grando good reason to translate these ideas to her HMM based model. However, for our goal of rain simulation, her model has some unnecessary complexity.

Grando calculates the number of storm discs over the entire space (all sites) but then only selects discs above the current site. She repeats this for all sites. We can expect most storm discs will not cover all sites, suggesting this method will generate a large number of potential discs and then reject most of them. This is extremely inefficient and computationally expensive. To solve this issue one may propose the following:

- Simulate the number and size of discs over the entire map of sites and determine which storms are above which sites. This method stops the unnecessary iteration through all sites and provides the same results as the parameter estimates will be account for this change.
- Simulate the number and size of discs over all sites independently. Here we ignore the spatial data of the sites. This prevents us from understanding which sites are covered by the same discs, but this is not of importance to us and thus can be left as is. Furthermore, we can remove the assumption of the shape of the disc as it is no longer relevant, allowing our model to be further generalised. This allows us to remove the radius parameters for each state, reducing total parameters by the number of states.

While both methods mentioned above allow for improvement, ultimately it does not matter which we pick. The model will fit the given parameters and the orientation of the parameters does not make a difference as they will adjust to compensate, resulting in a model that produces a similar output. As such, for our model we decide to go with the second method. Since we will be calculating over each site independently, we decide to calculate parameters for each separately rather than combining these results.

## 7.2 Split monthly

Rainfall is not consistent throughout the year. A simulation of rainfall in summer would likely have significantly different statistical properties to one from the winter. Grando's Model does not account for this, as such a simulation using this model could produce an output more likely in winter or summer, one cannot tell.

To address these concerns, we utilise Cowpertwaits [\\*\\*ref\\*\\*](#) methodology, splitting the data into months. This allows for simulating for particular months as well as a better fit for the overall data. Since our data is a timeseries of rainfall amounts with not much other information given we divide our data into the average number of days in a month; 30. We then label the first twelve groups 0 to 11 and fill each group cycling through 0-11. The exact methodology and jupyter notebook can be found in the "DataPrep" folder under "MyCode" within the code files.

Another necessary modification was changing the missing values. The given data contained "-9999" for any days with missing data. Removing these values could create unequal sized datasets for each month. To avoid this we decided to set these values to a particular number. Unfortunately any value we select will cause the data mean to shift towards the value. To solve this we set this value as the mean of the months rainfall ignoring the missing data.

## 7.3 High dimensionality

HMMs with even a few states contain a large number of parameters. Grando's Model contains 21. Through our adjustments made to the disc structure, we have reduced this to 18. High dimensionality is a common problem but Unfortunately does not have straightforward solutions. We could introduce new assumptions and try to simplify the data using dimensionality reduction methods but this may limit our ability to fit our model well.

To address this problem, we revert back to chapter 4.4. Instead of approaching this problem by starting the parameter estimation from the non-hmm parameters, we approach from HMM parameters. Momentarily ignoring the non-HMM parameters, we know that Baum-Welch can be used to estimate 12 of these 18 parameters. While this is not all, this is a great decrease in dimensions. However, our model does not have an observation matrix, which is required by the Baum-Welch [\\*\\*ref\\*\\*](#) algorithm. We address this in the following chapter.

# Chapter 8

## Simple HMM Rainfall Model

### 8.1 Applying Baum Welch to Rainfall data

#### 8.1.1 Building the Observation Matrix

To be able to use Baum-Welch we require an observation probability matrix `**ref**`, thus in order to use it we must create one. To create a finite matrix of probabilities we require our data to be split into finite groups. Fortunately the algorithm increases in computation cost linearly with the dimensions of the observation matrix rather than squared as it does with the number of states. Through efficient implementation working with large observation matrices is feasible.

Through looking at the datasets, we can see the maximum rainfall is usually up to 1000mm. In our case this value was 743. Furthermore, the maximum precision of our data is integers. Thus, we can discretise our observations into integers up to the maximum rainfall for the given dataset. As we know from `**ref**`, the rows of the observation matrix represent the states and the columns represent the observations, giving us a matrix 3 by M matrix, where M is the number of observations.

We can now apply Baum-Welch to produce a HMM model that simulates integer values of daily rainfall in mm up to a given maximum amount. Code files can be found in "Baum Welch" folder under "MyCode".

**Note 8.1.** The values of the alpha and beta helper functions become extremely small for large sequence of observations. In our testing we found after about 1500 observations these values become too precise to store in "long double". This mathematically holds as the sum of the final alphas and betas represents the probability of observing the sequence `**ref**`, with a larger sequence the probability naturally decreases. We avoid this problem by using a sequence of 1000 observations.

When using this method, one must ensure the alpha and beta values are not all being stored as zeros.

#### 8.1.2 Local optimum vs global Optimum

Once again from `**ref**`, we know that Baum-Welch converges to local optima. This suggests there may be a superior optima, a superior fit for the model, that we have not found. To ensure a satisfactory fit is found, we repeat the algorithm for multiple random starts; random



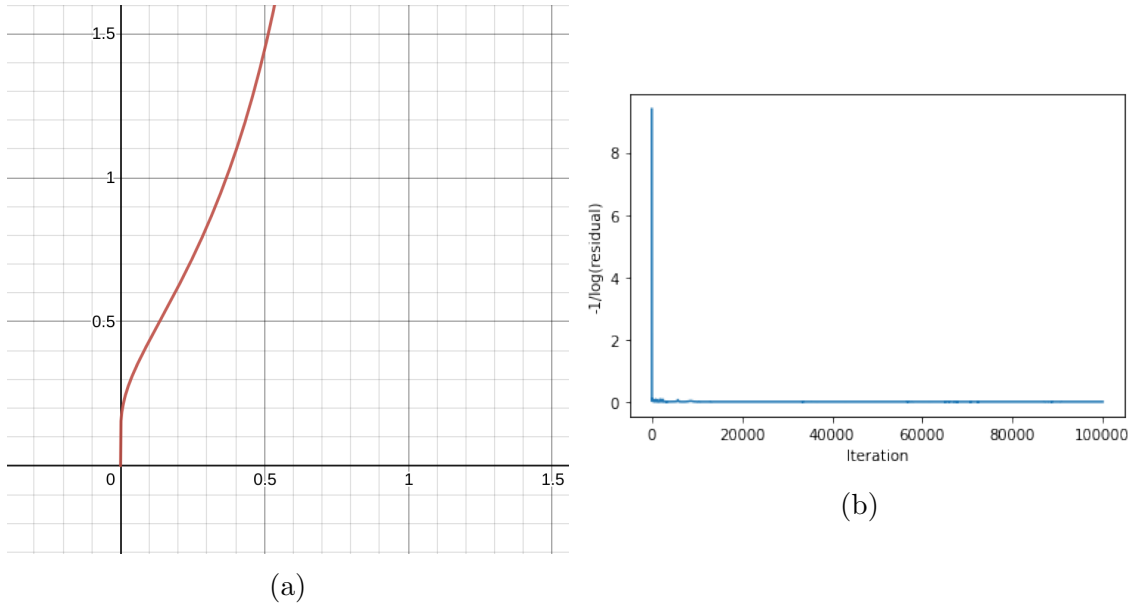


Figure 8.1

initial distribution, transition matrix and observation matrix. While this does not guarantee we will find the global optima it reassures us that our given optima is fairly suitable.

We choose to do 3 attempts. This was an arbitrary choice. We recommend using a much larger number as we have selected this for simplicity of demonstration.

## 8.2 Analysis

The program "BaumWelch" requires the data split into months in csv files. For each month it outputs the fitted transition matrix, observation matrix, initial vector, a Convergence log and a simulation using the model. This is done over 12 threads, allocating one for each month. For our data we computed 100000 iterations of Baum-Welch. The three attempts took 1074.76, 1081.38 and 1076.16 seconds respectively.

### 8.2.1 Convergence Log

We introduced a Convergence log to ensure the algorithm did in fact Converge. At each iteration, the difference between the current and previous transition matrix, observation matrix and initial distribution parameters was summed. This sum represents the residual, which we expect to tend to 0 as the number of iterations tends to  $\infty$ .

The values decrease quite rapidly, thus to create a visual representation we used the function  $y = \frac{-1}{\ln(x)}$ , where  $x$  is the residual, to scale the values. We can see in Figure 8.1a that our function is strictly increasing thus can be used for scaling.

Along with the csv files containing the log, a graphical representation can be found in the Analysis jupyter notebooks within each month's folder. Figure 8.1b shows one such graphical representation for the convergence log of attempt 1 for month 0. We can see that the residual converges to 0 quite quickly, suggesting we could potentially reduce the number of iterations. The convergence itself confirms that our two matrices and our initial distribution contain

values close to the local optima.

### 8.2.2 Selecting the optimum optima

As mentioned earlier, we perform multiple random starts to try and obtain the global optimum. However, from the model alone we do not know if it is the "most optimum". To do this we count frequencies.

For each attempt we count and record the frequency of occurrence of each interger in our observation set from our training rainfall data of 1000 observations as well as from our simulation generated by the HMM, also containing 1000 observations. We now find the difference the two sets of frequencies, square the given value and then take a sum. The attempt the lowest sum of squares is then accepted as the most optimum optima.

For our example of month 0, we had the following results:

Attempt	Sum of Square difference
1	656
2	672
3	1692

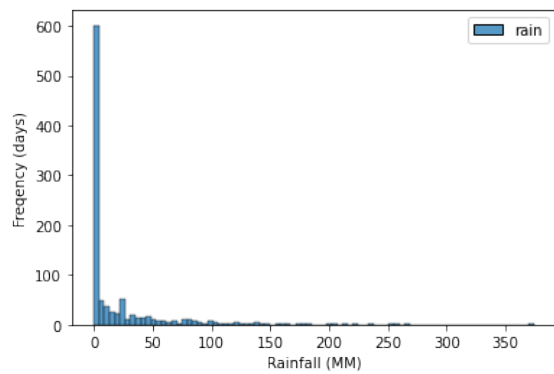
As such we selected attempt 1 for our testing. The attempt selected for each month is given in below table.

Month	0	1	2	3	4	5	6	7	8	9	10	11
Attempt	1	3	2	1	1	1	1	2	2	3	3	2

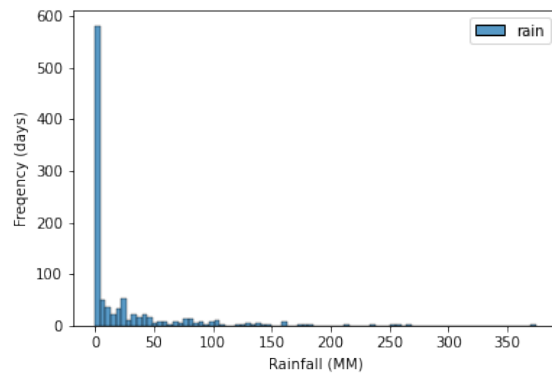
### 8.2.3 Results

Since the results for each month are quite similar , we will only present results for month 0. The Results for other months can be found in the analysis files accopymanyng each months folder.

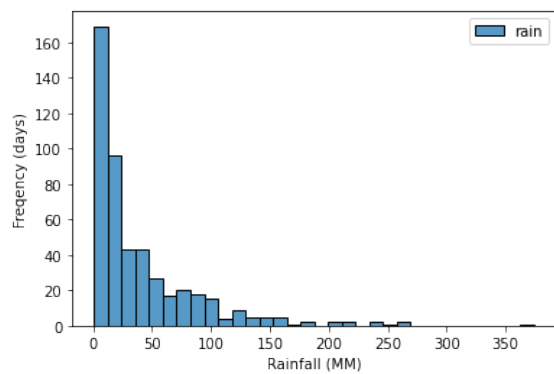
From Figure 8.2 we can see that the frequencies seem to be simliar for both simulation and recorded data. Furthermore, smaller details such as the peak at around 25mm also remains consitent. While this is promosing, the large frequency for 0mm distorts the graph. To compare further we plot the histograms once more, removing the first bar for 0mm. Here we can see it is not identical but seems to have similar properties, such as max, medium and high density areas. We will investigate this result numerically in the results chapter of this paper.



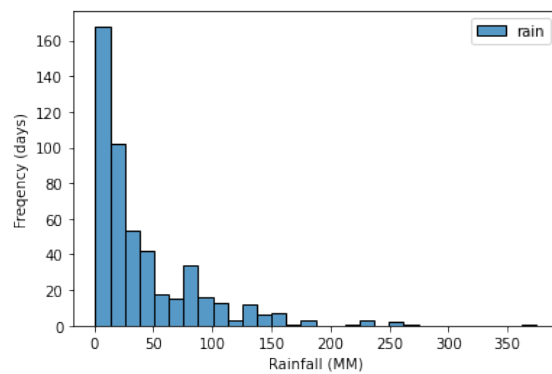
(a)



(b)



(c)



(d)

Figure 8.2

# Chapter 9

## Generalising the Observation Matrix

### 9.1 Generalising the model

We have seen that HMM can be used to simulate rainfall similar to that given in the training data. While this is useful, it also has some limitations. For example, due to our discretised system with interger value groups, some groups have 0 probability. This is justified by the data but cannot be explained by nature. For example, for our selected attempt of 1 in month 0, given the system is in state 1, the probability of 1mm of rainfall is 0.0284927 however the probability of 2mm is 0. This may be true for the data but is not a reasonable assumption in nature. As such we must find a method to generalise the model. Generalising the observation matrix, such that the simulation not only represents the training data but is infact useful in simulating future rainfall patterns.

### 9.2 Finding a function for Observation Matrix

To generalise our model we require a function for each row of the observation matrix. We could test for polynomial fits but we will attempt to use our model as it has real life justifications. We propose our model, which was developed by adjusting Grando's Model.

**Proposition 9.1.** *Given the system is in state  $i$ , the amount of rainfall in mm is given by*

$$\sum_{i=0}^N \sigma_i \tag{9.1}$$

*where  $N \sim \text{pois}(\lambda)$  is the number of storm discs and  $\sigma \sim \text{exp}(\tau)$  is the intensity of rain in each.*

### 9.3 Parameter Estimation

For each row of each observation matrix, according to 9.1, we have two unknown parameters. In other words, for our example, we have six unknown paramters for each month. Fortuantely we do not have to attempt to estimate all of these from the same data. We can use the distribution, given by each row, to generate a simulation and then attempt to fit our two parameters to this.

- For given min and max lambda and tau, simulate rainfall, calculate frequencies and find absolute difference between this and freq in row given by matrix B.
- Analysis done in R
- Plot 3D graph of lambda, tau and residual, look for minimum
- Repeat, optimising min and max lam and tau each time
- Look for hyperbola in tau-residual and lam-residual graphs and find minimum.
- Record minimum values as parameter estimates

Model Generation - Simply go through HMM and other model features using parameter estimates found in prev.

### 9.3.1 Methodology

The programs used for this are "pe" and "Analysis.R" both in the parameter estimation folder. The program "pe" follows the following steps:

1. Input of the minimum and maximum value desired of  $\lambda$  and  $\tau$  as well as which row is being fit.
2. Uniformly randomly selects values within the given ranges and generate a simulation of size 100000 for the given row.
3. Create array containing frequency counts of each integer from simulation.
4. Using observation matrix row as a distribution, find expected frequency for 100000 simulations by multiplying probability by number of iterations. Store results in array.
5. Find difference between each element of frequency of simulation and frequency of expectation array. Sum absolute value of each and take this value as residual for given  $\lambda$  and  $\tau$ .
6. Repeat steps 1000 times for given  $\lambda$  and  $\tau$ . Record  $\lambda$  and  $\tau$  and residual in csv file.

### 9.3.2 Results

"pe" has been optimised to run over up to 24 threads. Due to this, when given the 24 threads, the program finishes the above computation within 10-30 seconds.

"Analysis.R" is used to analyse the csv file. It plots three graphs; a 3D graph of  $\lambda$  against  $\tau$  against residual,  $\lambda$  against residual and  $\tau$  against residual. We use these three graphs to find an approximate area of  $\lambda$  or  $\tau$  where residual is lowest and then repeat "pe" for this new area.

We will demonstrate the results for the first row of the observation matrix for Month 0. The process followed for calculating all other parameter estimates has been the same.

We begin by running "pe" between 0-10 for both tau and lambda. Through initial testing with random ranges we found that both values tend to have their minimum residual points within these ranges. It is clear from Figure 9.1 that  $\lambda \in [0, 2]$  and  $\tau \in [0, 0.5]$ . Thus we repeat the program for this new range, the results can be seen in figure 9.2. This time we can see that  $\lambda \in [0.5, 0.75]$  and  $\tau \in [0, 0.1]$ . We repeat the program one more time with these new ranges and retrieve Figure 9.3. We can now approximate that  $\lambda \approx 0.625$  and  $\tau \approx 0.05$ .

We repeat this process for every row of the observation matrix for every month. The results are as given:

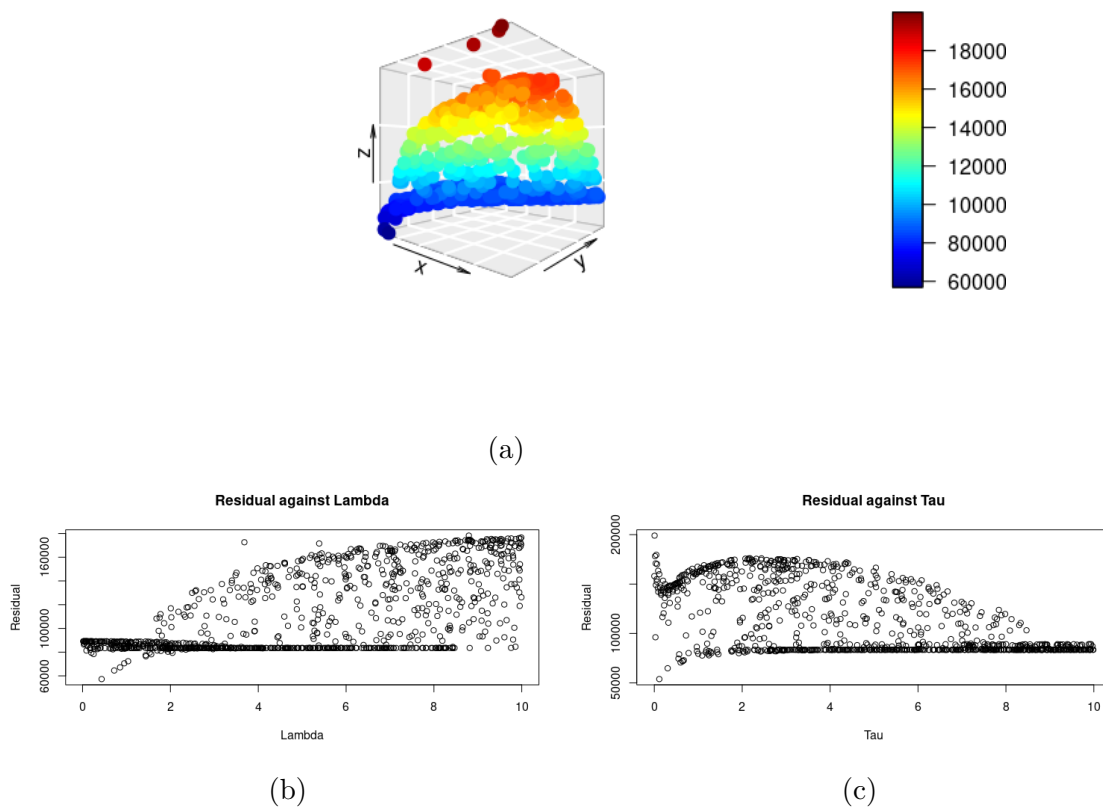


Figure 9.1

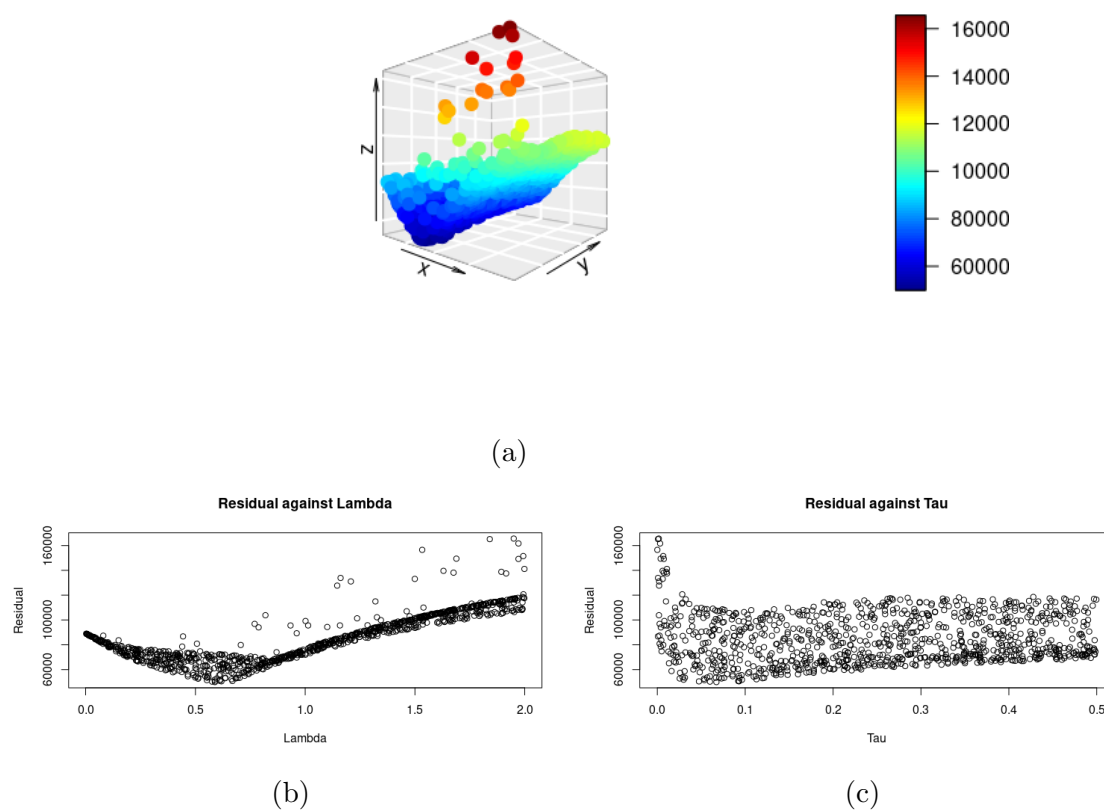
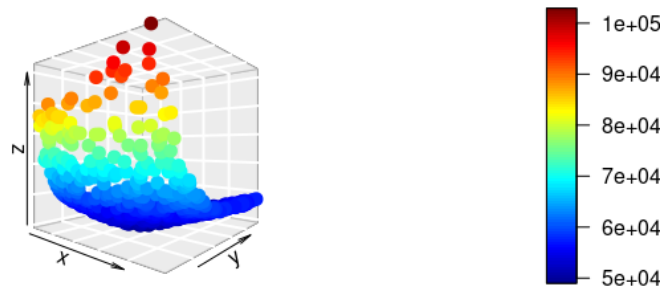
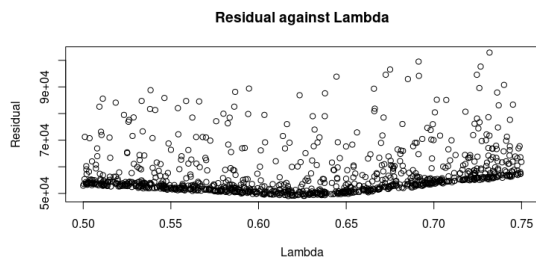


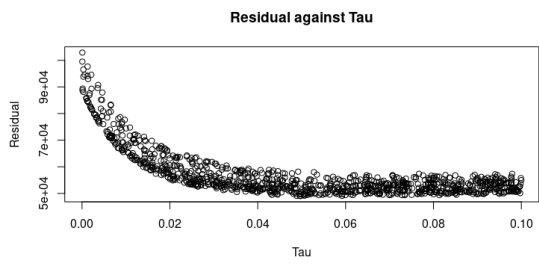
Figure 9.2



(a)



(b)



(c)

Figure 9.3



Month	Lambda 1	Lambda 2	Lambda 3	Tau 1	Tau 2	Tau 3
0	0.625	0.9	0.775	0.05	0.2	0.06
1	0.265	3.25	1.95	0.06	0.15	0.055
2	0.42	3.3	4	0.09	0.11	0.1
3	2.5	2.5	0.35	0.12	0.175	0.06
4	2.3	3.25	0.525	0.09	0.16	0.075
5	4	1.55	0.3	0.13	0.1	0.06
6	2.25	0.6	0.55	0.075	0.075	0.075
7	3.4	0.24	2.15	0.2	0.075	0.1
8	0.57	0.52	3.2	0.04	0.06	0.12
9	5.5	0.3	2.6	0.17	0.05	0.25
10	0.27	9	2.75	0.05	0.64	0.075
11	4.15	0.24	2	0.1	0.075	0.11

We can see from table 9.3.2 that while the values of  $\tau$  are often similar, lambda values vary drastically. Regardless, we are concerned with the performance of this model. In the next chapter we test this model as well as apply more rigorous tests on the hmm only model.

# Chapter 10

## Results

### 10.1 Our Datasets

Before we begin testing we summarise what data we have so far and if we require any more.

Each month has 2138 days of rainfall data. Due to unforeseen issues with the Baum-Welch algorithm we used only 1000 of this for training. The remaining 1138 has been left aside for out of sample testing. If our simulations fit this dataset, it is a strong indication of success, as fitting the training set could just mean the model has fit the given data and not the generalised case. Thus we have two sets of recorded data; the training, of size 1000, and the test of size 1138.

When we first created our HMM in chapter 8 **ref** we also generated a simulation of size 1000 days. We will use this simulation for our testing now. From our parameter estimates in chapter 9 **ref** we generated simulations of size 1000 using "generator", which can be found in the "Model Generator" folder.

Therefore we now have four sets of data two recorded, training and test, and two simulated, HMM only and our Generalised HMM model.

### 10.2 Results

We carry out our testing in "*KS<sub>tests</sub>.R*".

#### 10.2.1 CDFs

We begin by comparing the empirical cumulative distribution functions of each. We plot training and test data together. From Figure 10.1 we can instantly see that neither the HMM only or our generalised model simulate any extreme values observed in the recorded data. However, the shape of the graphs seem similar as well as the rate of increase.

#### 10.2.2 Kologmorov-smirnof Tests

To further test we apply the Kologmorov-Smirnof (KS) test for two samples **ref**. We apply this to both of our models against both training and test data. Our hypothesis are given as follows:

- $H_0$ : The two samples are from the same distribution

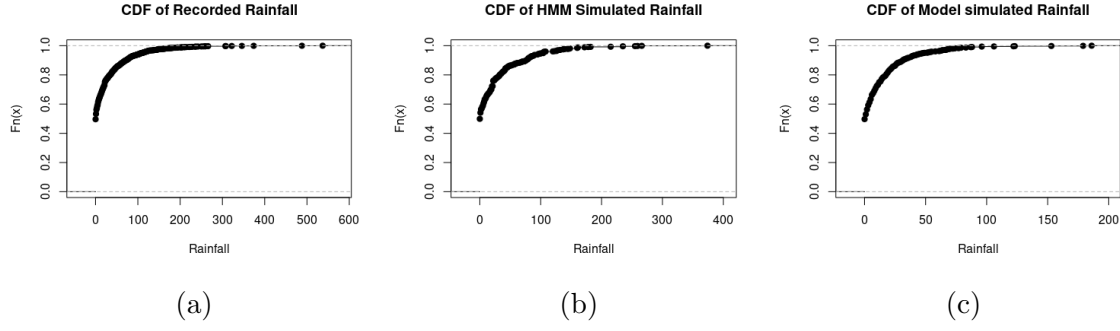


Figure 10.1

- $H_1$ : The two samples are not from the same distribution.

We will use a 5% significance level for testing. Thus if the p-value is smaller than 0.05 we will reject the null hypothesis. We record all four p-values for each month in the table given below.

Month	Train	Train	Test	Test
	HMM	Gen Model	HMM	Gen Model
0	0.8593	$4.939E^{-05}$	0.7899	$4.60E^{-08}$
1	0.9969	0.002038	0.105	$5.46E^{-06}$
2	1	$2.13E^{-05}$	0.1118	$7.65E^{-07}$
3	1	$1.79E^{-06}$	0.02163	$1.11E^{-07}$
4	0.4324	$1.38E^{-05}$	0.0009582	$7.98E^{-08}$
5	1	0.006202	0.2979	0.0004948
6	1	$1.11E^{-05}$	0.1809	$8.53E^{-08}$
7	1	$2.13E^{-05}$	0.03555	$4.53E^{-05}$
8	0.9999	0.0001108	0.07192	0.001033
9	0.9969	$6.06E^{-05}$	$5.22E^{-07}$	$1.22E^{-07}$
10	0.8879	0.0008667	0.1914	$9.07E^{-05}$
11	0.9999	$6.06E^{-05}$	0.265	$2.15E^{-06}$

It is clear to see that for all months, the KS test produces non-rejections. Thus one can assume the HMM describes the training data. For the test data we can see that for 4 out of 12 months we rejected the null hypothesis. To find the justifications for why this is the case further testing is required. Initial thoughts include: maybe the training data was more similar to the test for the 8 that did not reject or maybe the optima found through baum-welch was not infact a suitable optima.

For our generaliseed model we rejected the null hypothesis every time when tested against both training and test sets. This suggests the model is not describing the rainfall data well.

# Chapter 11

## Future Research

Our testing in chapter 10 *ref* suggests the HMM on its own does have potential. For this we used only 3 random starts of Baum-Welch. It is highly likely that increasing this number would increase the likelihood of achieving a better fit by finding a superior optima.

Our generalised model was not succesful. However if we can find a function of random variables that fits the rows of the observation matrix perfectly, then hypothetically we can expect to have matching performance to that of the HMM only model.

We tested a three state model, however this was an arbitraray choice. For future research, one may consider increasing this number as it may allow for a superior fit to the data. This, of course, must be done after careful considerations of computational cost as the algorithms used in testing are extremely sensitive to increase in value of this paramter.

To conclude, can hidden markov models be used to simulate rainfall? From our testing we have insufficient evidence to suggest not.

# Bibliography

- Baum, Leonard E et al. (n.d.). “An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes”. In: ().
- Baum, Leonard E, John Alonzo Eagon, et al. (1967). “An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology”. In: *Bulletin of the American Mathematical Society* 73.3, pp. 360–363.
- Baum, Leonard E. and Ted Petrie (Dec. 1966). “Statistical Inference for Probabilistic Functions of Finite State Markov Chains”. In: *Ann. Math. Statist.* 37.6, pp. 1554–1563. DOI: 10.1214/aoms/1177699147. URL: <https://doi.org/10.1214/aoms/1177699147>.
- Dempster, Arthur P, Nan M Laird, and Donald B Rubin (1977). “Maximum likelihood from incomplete data via the EM algorithm”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 39.1, pp. 1–22.
- Forney, G David (1973). “The viterbi algorithm”. In: *Proceedings of the IEEE* 61.3, pp. 268–278.
- GRIMMETT, GEOFFREY STIRZAKER et al. (2020). *Probability and random processes*. Oxford university press.
- Markov, Andrei (n.d.). “An example of statistical investigation of the text Eugene Onegin concerning the connection of samples in chains”. In: ().
- Rabiner, L. and B. Juang (1986). “An introduction to hidden Markov models”. In: *IEEE ASSP Magazine* 3.1, pp. 4–16. DOI: 10.1109/MASSP.1986.1165342.
- Viterbi, Andrew (1967). “Error bounds for convolutional codes and an asymptotically optimum decoding algorithm”. In: *IEEE transactions on Information Theory* 13.2, pp. 260–269.