Hidden Markov Models for Rainfall Simulation

Diljeet Jagpal

Spring 2021

Preface

Contents

Chapter 1

Introduction

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.

Chapter 2

Hidden Markov Models

Before we try to apply Hidden Markov Models (HMMs) to rainfall, we must first understand how they work. This chapter will build an understanding of the basic principles behind HMMs, along with developing and answering a few key questions.

2.1 Definition

This section will define a HMM and introduce notation used throughout this paper.

A hidden Markov model is a doubly stochastic process, where the underlying process is Markovian and hidden; Rabiner and Juang, 1986. This definition comes from the fact that there are two stochastic processes, one determining the transition between states and another 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 HMM, we require five things. These are as follows: 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.

   ii The states are usually ergodic, i.e. from any given state, we can eventually reach all others, but this is not necessary.

   iii The states are from the state space $S = \{S_1, S_2, ..., S_N\}$.

2. M

   i M is the number of observable outputs.

   ii These make a discrete alphabet of observations called $V = \{v_1, v_2, ..., v_M\}$.

3. A

   i A is the state transition matrix.

   ii This is the same as the p matrix for simple Markov chains.

   iii A $= \{a_{ij}\}$

   iv $a_{ij} = \{p(i,j) = \mathbb{P}\{X_n = j | X_{n-1}\}$

4. B

   i B is the observation probability matrix.

   ii B = $\{b_j(k)\}$

   iii $b_j(k) = \mathbb{P}(V_k\ at\ t|q_t = S_j)_{1\leq j\leq N, 1\leq k\leq M}$

5. π

   i π is a vector containing all inital state probabilities.

   ii $\pi = \{\pi_i\}$

   iii $\pi_i = \mathbb{P}(q_1 = S_i)$ for $1 \leq i \leq N$

We can now combine the above to provide a formal definition of HMMs.

Definition 2.1. 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 more notation, which will be used for the rest of this paper.

i Q represents the Markov model's state sequence, i.e. the hidden process.

ii We will use $O = \{o_1, o_2, ..., o_T\},\ O_i \in V$ to represent the observation sequence.

iii we will occasionally represent the hidden markov model as $\{N, M, \lambda\}$ where $\lambda = \{A, B, \pi\}$

2.2 Using HMM

As with all models, we can use HMM to simulate processes. In this section, we will demonstrate how to do so for a HMM with known parameters and propose a few questions one may ask.

2.2.1 Simulation

Simulating a process is usually our reason for building a model. Thus, in this subsection, we will demonstrate how to simulate a HMM to generate an observation sequence.

Let H = $\{N, M, A, B, \pi\}$ be a HMM. To generate a sequence of T observations O we do the following steps:

1. Using π as a probability measure, set $t = 1$ and randomly sample a state as the first $q_1 = s_i$.

2. Using $b_i(k)$ as a probability measure, randomly sample the observation $O_t = v_k$.

3. Using row i of a_{ij} as a probability measures, set $t = t + 1$ and randomly sample a the next state $q_{t+1} = s_j$.

4. Repeat steps 2 and 3 until $t = T$

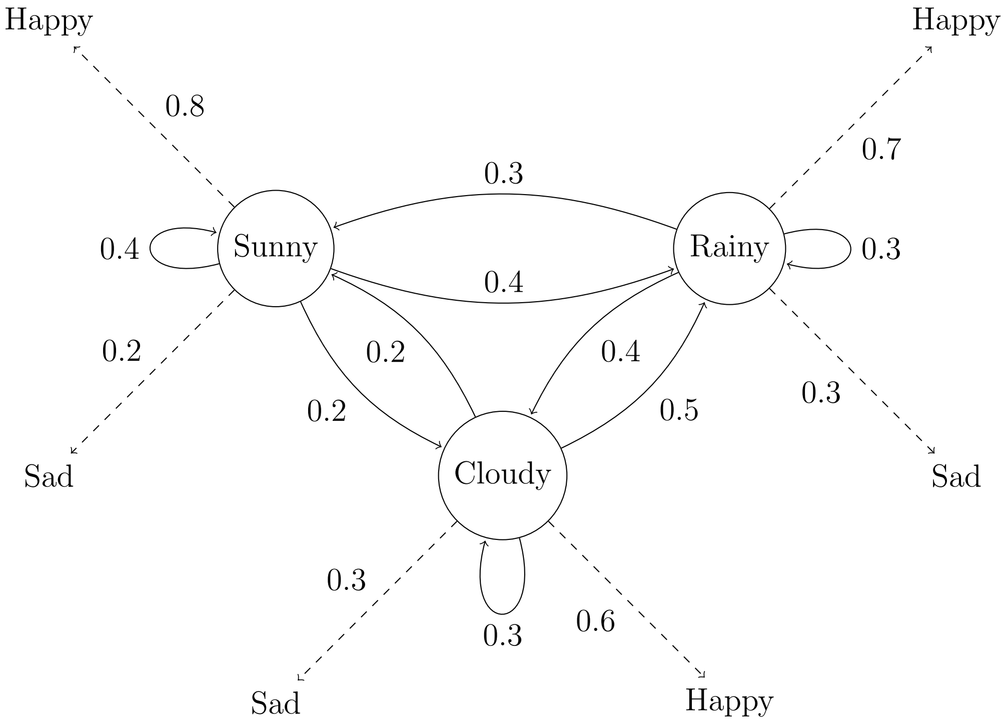


Figure 2.1: A chain representing the HMM with hidden states the weather and observable states mood.The solid lines represent transitions between hidden states, and the dashed represent observable.

Example 2.2. 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 π can she predict his mood for three consecutive days?

From the context, we can deduce the following:

N = 3, number of hidden states
M = 2, number of possible observations

$$A = \begin{array}{c} (Sunny) \\ (Rainy) \\ (Cloudy) \end{array} \begin{bmatrix} 0.4 & 0.4 & 0.2 \\ 0.3 & 0.3 & 0.4 \\ 0.2 & 0.5 & 0.3 \end{bmatrix} \tag{2.1}$$

$$B = \begin{array}{c} (Sunny) \\ (Rainy) \\ (Cloudy) \end{array} \begin{bmatrix} 0.8 & 0.2 \\ 0.7 & 0.3 \\ 0.6 & 0.4 \end{bmatrix} \tag{2.2}$$

$$\pi = \begin{array}{c} (Sunny) \\ (Rainy) \\ (Cloudy) \end{array} \begin{bmatrix} 0.5 \\ 0.3 \\ 0.2 \end{bmatrix} \tag{2.3}$$

We can now use 2.2.1 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 generate uniform random variables through python using the "rand.py" file. We will then combine these with inversion sampling to select the state.

i We generate a r.v. $= 0.0058$. Using Pi as the probaility 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 simulated observations O and see that it is equal to $\{v_1, v_1, v_1\}$, i.e. we observe three consecutive Happy days.

2.2.2 Key Problems

To ensure the validity of a model, one may ask questions regarding the probabilities of certain occurrences. We discuss these questions in this subsection.

We will focus on three famous questions highlighted in Rabiner and Juang, 1986.

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

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

3. Learning
   Given a set of observation $O = \{o_1, o_2, ..., o_T\}$, 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.

2.3 Problem 1: Evaluation

This section will tackle the Evaluation problem for HMM; 1. We will first attempt using a simple approach and then find a more efficient implementation; the Forward-Backward Algorithm.

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 applications. For example, we may have multiple potential models λ_i and cannot decide which one is most suitable. We can calculate this probability for each λ_i and then select the one with 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. Let us assume we have the state sequence $Q = \{q_1, q_2, ..., q_t\}$. For $\mathbb{P}(O|Q, \lambda)$ 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.

$$\mathbb{P}(O|Q, \lambda) = \prod_{t=1}^{T} \mathbb{P}(O_t|q_t, \lambda) \tag{2.4}$$

One may observe that these probabilities are simply taken from the matrix B.

$$\mathbb{P}(O_t|q_t, \lambda) = b_{q_t}(O_t), \quad \forall t \in [0, T] \tag{2.5}$$

Thus we can rewrite 2.4 as:

$$\mathbb{P}(O|Q, \lambda) = b_{q_1}(O_1) b_{q_2}(O_2)...b_{q_T}(O_T) \tag{2.6}$$

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 π_{q_1} as we also need the probability of starting at q_1.

$$\mathbb{P}(Q|\lambda) = \pi_{q_1} a_{q_1 q_2} a_{q_2 q_3} ... a_{q_{T-1} q_T} \tag{2.7}$$

We can now successfully remove Q from the condition using 2.6 and 2.7:

$$\begin{aligned} \mathbb{P}(O, Q|\lambda) &= \mathbb{P}(O|Q, \lambda)\mathbb{P}(Q|\lambda) && (2.8) \\ &= \pi_{q_1} b_{q_1}(O_1) a_{q_1 q_2} b_{q_2}(O_2) a_{q_2 q_3} ... a_{q_{T-1} q_T} b_{q_T}(O_T) && (2.9) \end{aligned}$$

This is the joint probability of observations and the internal states. In other words, it provides the probabilty that given observations O and internal state sequence Q were generated by model λ. To achieve our desired probablity 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 2.8. As we have accounted for every possible Q, we no longer need to worry about its particular value. This leaves us with:

$$\begin{aligned} \mathbb{P}(O|\lambda) &= \sum_{allQ} \mathbb{P}(O|Q, \lambda) P(Q|\lambda) && (2.10) \\ &= \sum_{q_1, q_2, ..., q_T} \pi_{q_1} b_{q_1}(O_1) a_{q_1 q_2} b_{q_2}(O_2) a_{q_2 q_3} ... a_{q_{T-1} q_T} b_{q_T}(O_T) && (2.11) \end{aligned}$$

Unfortunately, calculating this is infeasible as 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 2.3. Suppose we would like to find the probability of a string of 10 observations using model H, where $N = 3$ and $T = 10$.

$$\begin{aligned} \textit{operations} &= (2T-1)N^T + (N^T - 1) \quad (2.12)\\ &= (20-1)3^10 + (3^10 - 1) \quad (2.13)\\ &= 1180979 \quad (2.14) \end{aligned}$$

We have a problem as even with a small model; 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.

2.3.1 Forward-Backward Algorithm

The Forward-Backward (FB) algorithm offers an alternative, more efficient method to find $\mathbb{P}(O \mid H)$. In this subsection, we will demonstrate its use. Our explanations take motivation from Rabiner and Juang, 1986.

To use FB, we must first introduce two helper functions α and β. We will start by discussing the former.

$$\alpha_t(i) = \mathbb{P}(O_1, O_2, O_3, ..., O_t, q_t = S_i|\lambda) \quad (2.15)$$

α is an extremly powerful tool in reducing the number of calculations. As given by 2.15, it providse 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 proabbility for this.

We can combine 2.15 with induction to produce an iterative process that can solve our problem.

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 π_i and the later by $b_i(O_1)$. This gives us:

$$\alpha_1(i) = \pi_i b_i(O_1), \quad i \in [1, N] \quad (2.16)$$

ii Inductive step:

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:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1}), \quad j \in [1, N], t \in [1, T-1] \quad (2.17)$$

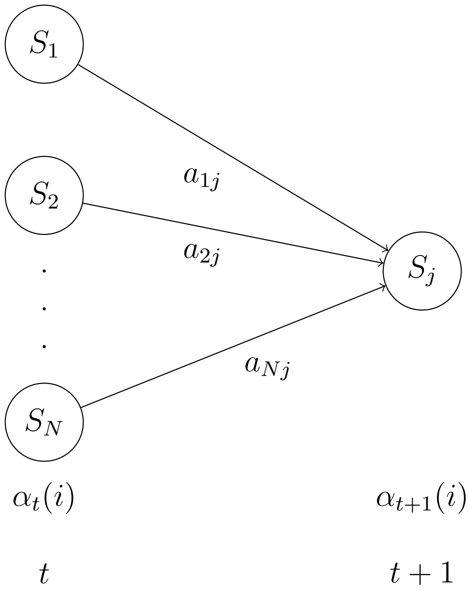


Figure 2.2: For each state j, for each inductive step, we follow the above method. Each$\alpha_t(i)$ is multiplied by each a_{ij} and then summed up. This resultmultipled by $b_j(O_{t+1}$ gives us $\alpha_{t+1}(j)$.

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.

$$\mathbb{P}(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{2.18}$$

One can see that this method is far more efficient than 2.10. 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 2.4. We attempt to solve the problem stated in 2.3 but this time using the FB algorithm. Suppose we would like to find the probability of a sequence of 10 observations using model H. Again, $N = 3$ and $T = 10$.

$$\begin{aligned} operations &= (T-2)(N^2+N+1)+2N \quad (2.19)\\ &= (10-2)(3^2+10+1)+6 \quad (2.20)\\ &= 118 \quad (2.21) \end{aligned}$$

118 is many orders of magnitude smaller than 1180979, which was the requirement without the FB algorithm. Thus it is a significant improvement. The improvement against

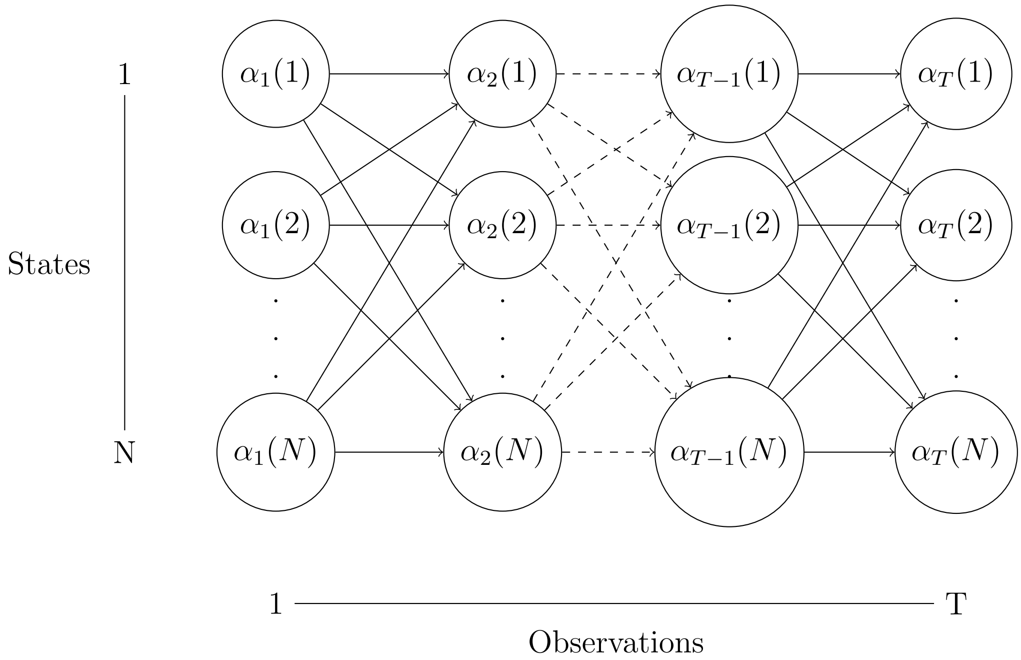


Figure 2.3: For each α at time t, we require all α at time $t-1$. Thus werequire N^2 calculations for each timestep.

the base method will be even more drastic for larger models, as FB has a fixed N^2 instead of the N^T term.

For question 1 this is sufficient. However, we will later require the β, 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.

$$\beta_t(i) = \mathbb{P}(O_{t+1}, O_{t+2}, , ..., O_T | q_t = S_i, \lambda) \tag{2.22}$$

This represents the probability of seeing the observations O_{t+1} up to O_T given that at time t the model λ is at state S_i. Once again, we calculate this using induction.

1. Base case
   For each state we must assume that it is the final state in order to iterate from it. This gives us:
   $$\beta_T(i) = 1, \quad i \in [1, N] \tag{2.23}$$

2. Inductive step:

   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, the transition 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 liklihood of being at that state based on future states $\beta_{t+1}(i)$ are multiplied. This gives us:
   $$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(i), \quad i \in [1, N], t \in [1, T-1] \tag{2.24}$$

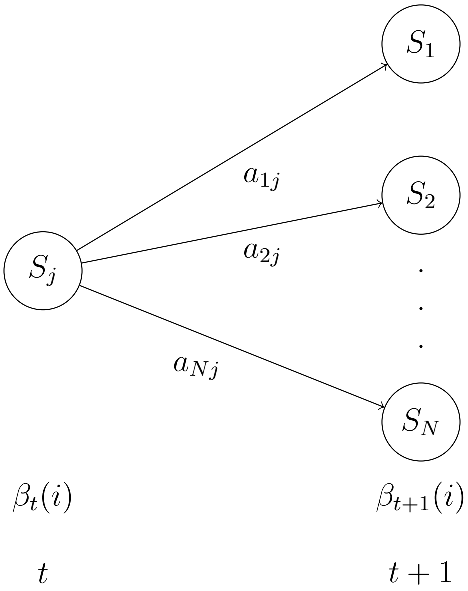


Figure 2.4: Similar to α, we use induction. However, this time we look into the future of the sequence and take steps backwards.

2.4 Problem 2: Decoding

This section will address the decoding problem 2. We will use the two helper functions developed in 2.3 together with the Viterbi Algorithm to do so. We will use ideas presented in Rabiner and Juang, 1986 and Forney, 1973.

The problem searches for the sequence of states that best explains the sequence of observations. However, the definition of 'best' is quite vague and open to interpretations. An obvious appraoch would be to look for each time t, find the most likely state, given all observations and the model. Lets define this probaility as γ.

$$\gamma_t(i) = \mathbb{P}(q_t = S_i | O, \lambda) \tag{2.25}$$

Using 2.15 and 2.22 we can solve for γ 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 suggests γ is equal to $\alpha_t(i)\beta_t(i)$. 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.

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\mathbb{P}(O|\lambda)} \tag{2.26}$$

$$= \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)} \tag{2.27}$$

This ensures that:

$$\sum_{i=1}^{N} \gamma_t(i) = 1 \tag{2.28}$$

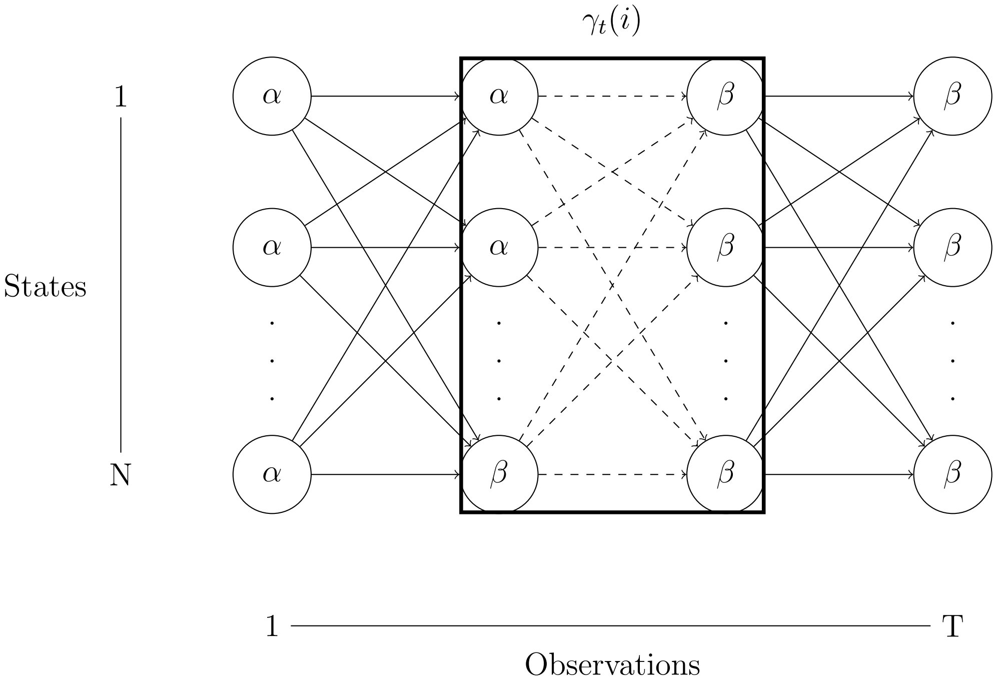


Figure 2.5: Each dashed line represents a calculation for $\gamma_t(i)$. For each time step, we calculate these and then find the maximum. We select the state corresponding to the maximum and add it to the state sequence.

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.

2.4.1 Viterbi Algorithm

In this subsection, we will demonstrate the application of the Viterbi Algorithm.

The Viterbi Algorithm was developed in Viterbi, 1967 and explored in further depth by in Forney, 1973. It follows a similar lattice structure of the helper functions in the FB algorithm 2.3.1. Similar to how we broke 1 into smaller pieces, we do the same to $\mathbb{P}(Q|O, \lambda)$.

$$\delta_t(i) = max_{q_1,q_2,...,q_t-1}\mathbb{P}(\{q_1, q_2, ...q_t = i\}, O_1, O_2, ..., O_t|\lambda) \tag{2.29}$$

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.

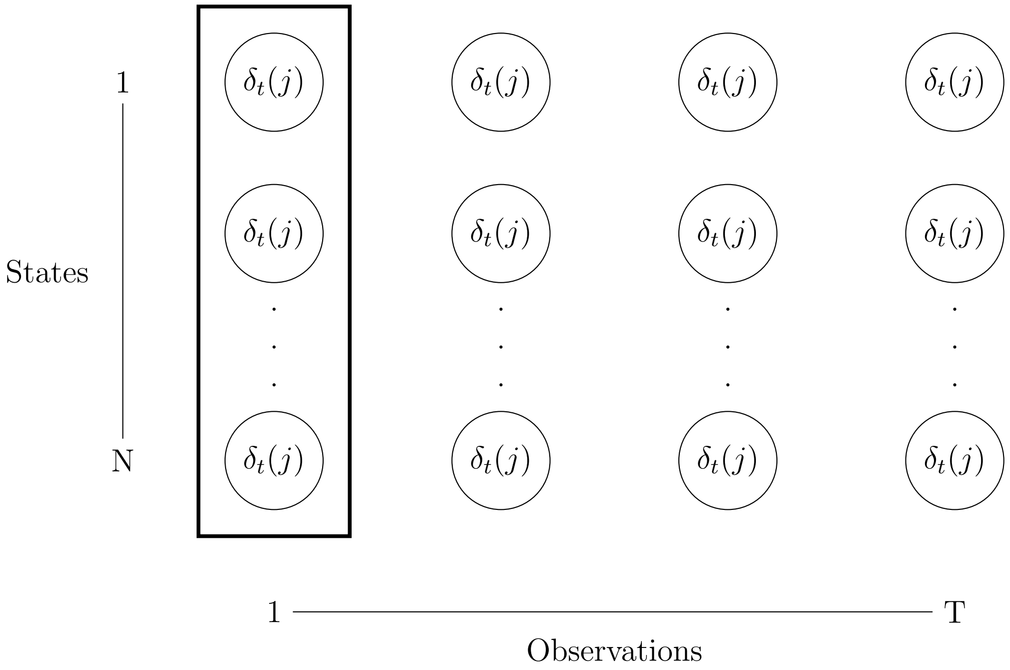


Figure 2.6: Once $\delta_t(j)$ is calculated for a particular t, we find the maximum, which is the state we add to our state sequence. Since $\delta_t(j)$ accounts for transition probability a_{ij}, impossible transitions will always be equal to 0. Thus unless all transitions are equal to 0, they cannot be the maximum.

1. Base case
   For each state we must calculate $\pi_i b_i(O_1)$ to determine which intial state is most likely. We also must set $\psi_1(i) = 0$ as there have not been any states until now.

$$\delta_1(i) = \pi_i b_i(O_1) \tag{2.30}$$
$$\psi_1(i) = 0 \tag{2.31}$$

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 liklihood and the transition probability. We maximize this term and the 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 ψ. This gives us:

$$\delta_t(j) = max_{1\leq i \leq N}[\delta_{t-1}(i)a_{ij}]b_j(O_t), t \in [2, T] \tag{2.32}$$
$$\psi_t(j) = argmax_{1\leq i \leq N}[\delta_{t-1}(i)a_{ij}], j \in [1, N] \tag{2.33}$$

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^*.

$$\mathbb{P}^* = max_{1\leq i \leq N}[\delta_T(i)] \tag{2.34}$$
$$q_T^* = argmax_{1\leq i \leq N}[\delta_T(i)] \tag{2.35}$$

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

2.5 Problem 3: Learning

HMMs have a large number of parameters. Obtaining these is a high-dimensional optimisation problem. In this section, we utilise the Baum-Welch Algorithm to solve all HMM parameters simultaneously.

The learning problem 3 addresses how can we learn a model from given sequence of observations? Given the sequence of observations $O = \{O_1, O_2, ..., O_T\}$ we want to find the most suitable $\lambda = (A, B, \pi)$.

2.5.1 Baum-Welch Algorithm

In this subsection, we will demonstrate the Baum-Welch Algorithm. Explanations have been inspired by Leonard E Baum et al., n.d.

For the Baum-Welch algorithm we will need all three parameters intrudced earlier; α 2.15, β 2.22 and γ 2.25. 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.

$$\xi_t(i,j) = \mathbb{P}(q_t = S_i, q_{t+1} = S_j | O, \lambda) \tag{2.36}$$

To solve this problem we can refer back to 2.15 for the left hand side (including q_t) and 2.22 for the right hand side (indlucing q_{t+1}). To get from state i to j we use $a_{ij}b_j(O_{t+1})$. Putting this all togther we get a liklihood. To normalize this into a probabiltiy we once more use $\mathbb{P}(O|\lambda)$. This gives us:

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\mathbb{P}(O|\lambda)} \tag{2.37}$$

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.

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)} \tag{2.38}$$

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:

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j) \tag{2.39}$$

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:

$$\sum_{t=1}^{T-1} \gamma_t(i) \tag{2.40}$$

Similarly for $\xi_t(i)$, 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:

$$\sum_{t=1}^{T-1} \xi_t(i,j) \tag{2.41}$$

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

$$\bar{\pi} = \gamma_1(i) \tag{2.42}$$

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.

$$\bar{a_{ij}} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{2.43}$$

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.

$$\bar{b_j(k)} = \frac{\sum_{t=1,s.t.O_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T-1} \gamma_t(j)} \tag{2.44}$$

We have used our λ and O to produce our parameters: $\alpha_t(i)$, $\beta_t(i)$, $\gamma_t(i)$ and $\xi_t(i,j)$. 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 equvialent. Proof of this can be found in Leonard E Baum et al., n.d. Once lambda stabalizes and is no longer changing, we can assume a local optimumum has been reached.

Example 2.5. Let us continue from Example 2.2. Alice is unsure of her model, thus records Bob's mood for five days. Her Observation sequence is {Happy, Sad, Happy, Sad, Sad}. She wants to use one iteration of Baum-Welch to improve her model. Code files are available in the "Example" Folder under "MyCode".

To apply the algorithm we need our $\alpha_t(i)$, $\beta_t(i)$, $\gamma_t(i)$ and $\xi_t(i,j)$.

We begin with $\alpha_t(i)$. We are expecting a lattice of size 3x5. We calculate using 2.15 and store the values in the following matrix:

$$\begin{bmatrix} 0.4 & 0.0494 & 0.048984 & 0.00912654 & 0.00250527 \\ 0.21 & 0.0849 & 0.059661 & 0.0173532 & 0.00540931 \\ 0.12 & 0.08 & 0.040704 & 0.018349 & 0.00570851 \end{bmatrix}$$

Similarly for $\beta_t(i)$ using 2.22,

$$\begin{bmatrix} 0.0178115 & 0.0648 & 0.0844 & 0.28 & 1 \\ 0.0197177 & 0.062835 & 0.0943 & 0.31 & 1 \\ 0.0196482 & 0.063591 & 0.0949 & 0.31 & 1 \end{bmatrix}$$

Using 2.25 to calculate $\gamma_t(i)$,

$$\begin{bmatrix} 0.522979 & 0.234978 & 0.303474 & 0.187581 & 0.183899 \\ 0.303949 & 0.391592 & 0.412978 & 0.39488 & 0.397069 \\ 0.173072 & 0.373431 & 0.283549 & 0.417539 & 0.419032 \end{bmatrix}$$

$\xi_t(i,j)$ produces a result that resembles a transition matrix for each timestep. We will have 5-1 = 4 of these.

- $$\begin{bmatrix} 0.152212 & 0.221395 & 0.149372 \\ 0.0599335 & 0.0871742 & 0.156841 \\ 0.0228318 & 0.083023 & 0.0672175 \end{bmatrix}$$
- $$\begin{bmatrix} 0.0979363 & 0.0957461 & 0.0412951 \\ 0.126237 & 0.123414 & 0.141942 \\ 0.0793006 & 0.193818 & 0.100312 \end{bmatrix}$$
- $$\begin{bmatrix} 0.0805428 & 0.133759 & 0.0891724 \\ 0.073574 & 0.122185 & 0.217218 \\ 0.0334641 & 0.138936 & 0.111149 \end{bmatrix}$$
- $$\begin{bmatrix} 0.0535945 & 0.0803918 & 0.0535945 \\ 0.0764283 & 0.114643 & 0.203809 \\ 0.0538761 & 0.202035 & 0.161628 \end{bmatrix}$$

Using these along with 2.5.1 will allow us to re-estimate our A, B and π parameters. π is the first column of the γ matrix. Thus our new inital distribution is

$$\bar{\pi} = \{0.522979, 0.303949, 0.173072\} \tag{2.45}$$

The number of transtions from state i to j can be taken as the sum of a particular position over all η matrices. These can be divded by the sum of the corresponding γ row, disregarding the last timestep.

$$\bar{a_{ij}} = \begin{bmatrix} 0.307672 & 0.425369 & 0.266959 \\ 0.223609 & 0.297603 & 0.478789 \\ 0.151871 & 0.495204 & 0.352925 \end{bmatrix} \tag{2.46}$$

For each row of γ, we sum the values where the observation is Happy and where it is Sad and then divide both by the sum of the row. This gives us a row of our new observation matrix.

$$\bar{b_j(k)} = \begin{bmatrix} 0.576765 & 0.423235 \\ 0.377237 & 0.622763 \\ 0.27398 & 0.72602 \end{bmatrix} \tag{2.47}$$

This method can be repeated until convergence to retrieve the optimal parameters (local optimum).

Chapter 3

Model Selection

Selecting the best model to represent a process is often not a trivial task. This chapter discusses methods of comparing models against data, approximating parameters, and testing whether a model fits.

3.1 Maximum Likelihood Estimators

We wish to find a model that can explain a given dataset. Assume we have two such models; how do we decide which is superior at this task? In this chapter, we will build on this idea.

3.1.1 Likelihood

In this subsection, we define likelihood and motivate the need for Maximum likelihood estimators.

To decide between models, we must answer which model explains the dataset better. We interpret this as the probability a given model produces a sequence of observations. This probability is known as the likelihood. We can now define this; Ross, 2004.

Definition 3.1. Likelihood Function
Given model θ and the observations $x_1, x_2, ..., x_n$, the likelihood function is given by

$$L(\theta|x_1, x_2, ..., x_n) = \mathbb{P}(x_1, x_2, ..., x_n|\theta) \tag{3.1}$$

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 data. A model with a higher Likelihood would have a superior fit to a model with a lower likelihood. As such, it is natural to maximise the likelihood in search of the ideal model.

3.1.2 Maximum Liklihood Estimators

Through maximising the likelihood function 3.1, we can find the model that best fits the given observations. This process is called Maximum Likelihood estimation. In this subsection, we will demonstrate its methodology.

We often use $\log(L(\theta|x_1, x_2, ..., x_n))$ instead of $L(\theta|x_1, x_2, ..., x_n)$ as this often leads to easier differenciation. This is possible as the log function is monotonically increassing thus has the same maximum at the same value of θ.

The method can be described as follows:

i Calculate the joint probability density of observeriving your data $X_1 = x_1, ..., X_n = x_n$ given your model θ, i.e.

$$\mathbb{P}(X_1 = x_1, ..., X_n = x_n|\theta) \tag{3.2}$$

ii Take the Natural logarithm of both sides.

$$\ln(\mathbb{P}(X_1 = x_1, ..., X_n = x_n|\theta)) \tag{3.3}$$

iii Take the partial derivative with respect to each parameter.

iv For each paramter, set the derivative equal to 0 and rearrange for its value. The given value will be the maximum likelihood estimation of that parameter.

Example 3.2. To demonstrate 3.1.2 we present the following example.

Let $\{1, 2, 4, 4, 4, 9\}$ be our Observed data. We are convived that this data follows an expoential(λ) distribution but we are unsure of what the λ parameter is. We begin by finding the joint probability density. According to our model all observations are independent thus the joint distribution is:

$$\mathbb{P}(X_1 = x_1, ..., X_6 = x_6|\theta) = \mathbb{P}(X_1 = x_1)...\mathbb{P}(X_6 = x_6) \tag{3.4}$$

We can now substitute in the PDF of an expoential distribtution at the different observations.

$$\mathbb{P}(X_1 = x_1, ..., X_6 = x_6|\theta) = \lambda e^{-\lambda}\lambda e^{-2\lambda}\lambda e^{-4\lambda}\lambda e^{-4\lambda}\lambda e^{-4\lambda}\lambda e^{-9\lambda} \tag{3.5}$$

We now simplify and take the natural log of both sides.

$$\begin{aligned} \ln(\mathbb{P}(X_1 = x_1, ..., X_6 = x_6|\theta)) &= \ln(\lambda^6 e^{-24\lambda}) && (3.6)\\ &= ln(\lambda^6) - 24\lambda && (3.7) \end{aligned}$$

We now differenciate with respect to our parameter, λ.

$$\frac{d}{d\lambda} = \frac{6}{\lambda} - 24 \tag{3.8}$$

Finally, to find the maximum we set the differential equal to 0 and solve for λ.

$$\begin{aligned} 0 &= \frac{6}{\lambda} - 24 && (3.9)\\ 24 &= \frac{6}{\lambda} && (3.10)\\ \lambda &= \frac{6}{24} && (3.11)\\ &= \frac{1}{4} && (3.12) \end{aligned}$$

Through MLE, our parameter estimate for λ is 0.25; thus our model is exponential(0.25). Logically this seems to make sense as the expected value of this model is 4, which seems to be the most popular and central value from the dataset.

As we can see, even for a simple model, the derivate is not trivial. Unfortunately, in real-world applications, this is often the case, particularly true for Hidden Markov based models. Thus we often require alternative methods to compare models.

3.2 Approximate Bayesian Computation

Maximum Likelihood Estimators are not always easy to calculate. Approximate Bayesian Computation (ABC) can provide an alternative method of parameter estimation in some such cases. In this section, we will discuss the principles of Bayesian Statistics, ABC and provide a demonstration.

ABC uses Bayesian inference. It involves using the prior distribution, our assumption of the model, combined with the likelihood to infer a posterior distribtution, the expected distribtution given our data and model. This can be more formally described using Bayes Theorem. Given x is the data and θ is the model,

$$\begin{aligned} \mathbb{P}(\theta|x) &= \frac{\mathbb{P}(x|\theta)\mathbb{P}(\theta)}{\mathbb{P}(x)} && (3.13) \\ \mathbb{P}(\theta|x) &\propto \mathbb{P}(x|\theta)\mathbb{P}(\theta) && (3.14) \\ posterior &\propto likelihood * prior && (3.15) \end{aligned}$$

ABC infers posterior distributions by replacing the likelihood calculation with a comparison between observed and simulated data. Toni et al., 2009. While there are many variations of ABC, we will focus on the simplest, ABC Rejection Sampling. The method can is as follows:

1. Sample θ' from the prior distribution $\mathbb{P}(\theta)$.
2. Simulate dataset x' using the sampled θ'.
3. Compare simulated dataset with observed dataset using a distance function. If the distance is smaller than a preset ϵ, then accept θ'; otherwise, reject. The distance function can use summary statistics.
4. Repeat previous steps.

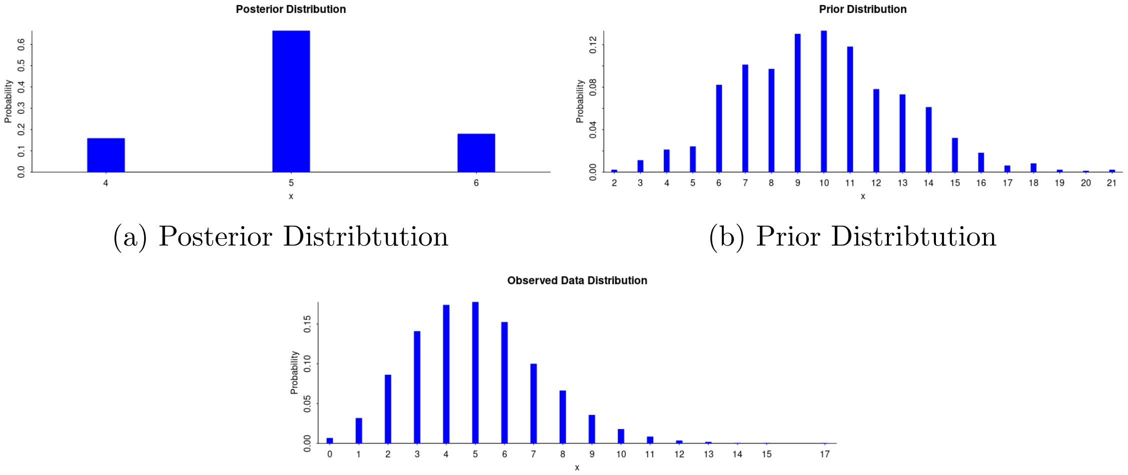
Given a small enough tolerence (ϵ) and an appropriate statistic we have:

$$\mathbb{P}_\epsilon(\theta|x) = \int \mathbb{P}_\epsilon(\theta, x'|x)dx' \approx \mathbb{P}(\theta|x) \tag{3.16}$$

Example 3.3. To demonstrate ABC, we will set the posterior distribtution and attempt to sample from it indirectly.

Let our observation data be 10000 Poisson(5) random variables. Also, let our prior distribution be Poisson(10). In other words, the actual distribution (posterior distribution) is Poisson(5), but we believe it is Poisson(10). Let our summary statistics be mean and variance, both of which must be within $\epsilon = 1$ difference between simulated and actual. Code for this example is available in the "Example" folder under "MyCode" in the "ABC.R" file.

We begin with the first step of 3.2. We generate a Poisson(10) variable θ'. Now we simulate Poisson(θ'). If the mean and variance of this sample are within 1 of the mean and



(a) Posterior Distribtution

(b) Prior Distribtution

(c) Observation Data Distribtution

Figure 3.1: A graphical reprsentation of the posteior, prior and observed distributions for Example 3.3.

variance for the observation data we accept, else we reject. We generate simulations of size 10000 in order to generate 1000 random variables distributed by our posterior distribution.

As we can see from Figure 3.1, the ABC generated sample has sampled from the high probability density areas of the true distribution. To get a complete posterior, which accounts for extreme values with lower probability, one should investigate other summary statistics.

3.3 Kolmogorov-Smirnov Test

This section will introduce a method to compare the similarity between two sets of data; the Kolmogorov-Smirnov Test.

Once we have a model, we must test to see how effective it is as generating values that follow the observed distribution. One such test for continuously distributed data is the Kolmogorov-Smirnov test. This test is a form of hypothesis testing.

Suppose we have two sets of data, our observed and simulated. We begin by calculating the empirical cumulative distribution function for each. We will label these as F_o for observed and F_s for simulated data. We can now state the hypothesis.

- $H_0 := F_o = F_s$, the two datasets are distributed by the same distribtution.
- $H_1 := F_o \neq F_s$, the two datasets are not distributed by the same distributed.

The test statistic is given by the following:

$$D \equiv Max_x |F_s(x) - F_o(x)| \tag{3.17}$$

The P-value is calculated using a probability of this test statistic. We will not explore this in-depth, but more information can be found at Ross, 2004[506]. If the P-value is smaller than the selected significance level, then we reject the null hypothesis. Otherwise, we say we have insufficient evidence to reject.

If we reject the null hypothesis, then our model is incorrect. If we do not reject, then our model may produce simulations matching in distribution to the observed data.

Chapter 4

Replicating Existing Rainfall Models

In this chapter, we will explore existing rainfall models, with a particular focus on HMMs. We will replicate results found using HMMs and discuss potential drawbacks.

4.1 Cowpertwait

Cowpertwait's rainfall models have been used for decades with a good deal of success. This section will aim to explain the motivations behind Cowpertwaits models.

Although we will not dive deep into his methodology, one may reference the following paper for more information. All papers are available in the bibliography. The earliest paper we have referenced is P. S. Cowpertwait, 1994.

Over time, Cowperthwait has improved and adjusted his model for varying applications. For example, in one of his more recent papers P. S. Cowpertwait, 2010, 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 P. S. Cowpertwait, 1994 to understand the model at its simplicity, and one may apply these ideas to a Hidden Markov Model.

All of Cowpertwaits models use Neyman-Scott point processes to describe the arrival of Storms. The model itself is 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 cell type. In P. S. Cowpertwait, 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 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 storm's centre.

4.2 Grando

Grando extended on Cwppertwait's ideas in Grando, 2019. She completely removes the double-disc structure and attempts to enclose the correlations within the larger Storm discs from Cowpertwait in the Hidden Markov States. In this section, we will explore her model and how she fits her parameters.

4.2.1 Model

This subsection presents Grando's Rainfall Model in a summariesed way.

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

$$I_n(x) := \sum_{i=1}^{N^{(n)}} \sigma_i^{(n)} \mathbb{I}_{D(x_i^{(n)}, R_i^{(n)})}(x) \tag{4.1}$$

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 λ,ξ and τ are taken to be fixed for each type of storm. In other words, for each hidden state of the HMM, there is a unique λ,ξ and τ. Informally, the model has hidden states which determine the type of storm. This then dictates the values of the three above parameters, which influences the intensity of rain observed.

Thus we arrive at the learning problem. Due to the added complexity of additional parameters, we cannot directly use methods for HMMs discussed earlier in the paper. Direct likelihood calculation is complex for HMMs alone; after introducing 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 λ,ξ and τ. This problem is high-dimensional. To overcome this, Grando fixes certain parameters in order to gain a better understanding of the remaining. She uses two methods; Nelder-Mead and Approximate Bayesian Computation 3.2. We will focus on the latter as it has provided superior results in her testing.

4.2.2 Fitting

In this subsection, we will explore Grando's Model's fitting methodology.

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 have parameters for the HMM and parameters for the observations but have to extract both from only the rainfall intensity data.

Grando determines that even without the HMM parameters, the problem is high dimensional ($3m$ parameters). She decides to fix all HMM parameters to uniform(0,1) for each test whilst ensuring the transition matrix is stochastic through the normalisation of rows. She believes that this will allow her to retrieve close approximations of all λ,ξ and τ from the data, which she can then use to calculate the HMM parameters.

For each simulation, she will generate a new value for all λ,ξ and τ. 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 actual value of each is. Here, she sets her ABC tolerance to ∞ and then manually narrows 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 only aimed to explore the problem, as decisions made are completely arbitrary.". Theoretically, we could pick any values, but we will match the values she has used for consistency.

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.

4.3 Replicating Grando

This section will be replicating Grando's results to analyse her finding in further depth and explore the following stages for research. We replicate both nine-parameter and three-parameter estimation, in addition to our improved algorithm used to test the validity of the testing methodology itself.

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 January 1931 to 31st December 2000. Co-ordinate data for each site is also given in an additional file, labelled 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.

4.3.1 Code

We present a brief summary of the coding process in this subsectino.

To ensure the validity of the 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 configuration is described thoroughly in the notes given with the code.

C++ was used for the actual simulation as this was the slowest part of Grando's algorithm. Many parameters have been hardcoded. This is to allow for efficiency and can later be altered if needed. The code has been Multi-threaded for further reduction in run-time. Admittedly the code can be further parallelised, 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.

4.3.2 Results

In this subsection, we present our results when replicating Grando's work in Grando, 2019. We finish with our adjusted test to show why Grando's estimation methodology was flawed.

Nine Parameter Estimation

We begin with using ABC to estimate all parameters, found on pages Grando, 2019[75-76]. The parameters are picked as follows:

- $\lambda_1 \sim$ Unif(5,15), $\lambda_2 \sim$ Unif(16,25) and $\lambda_3 \sim$ Unif(26,35)
- $\xi_1 \sim$ Unif(0.03,0.07), $\xi_2 \sim$ Unif(0.008,0.02) and $\xi_3 \sim$ Unif(0.001,0.007)
- $\tau_1 \sim$ Unif(0.8,1.4), $\tau_2 \sim$ Unif(0.3,0.7) and $\tau_3 \sim$ Unif(0.07,0.2)

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

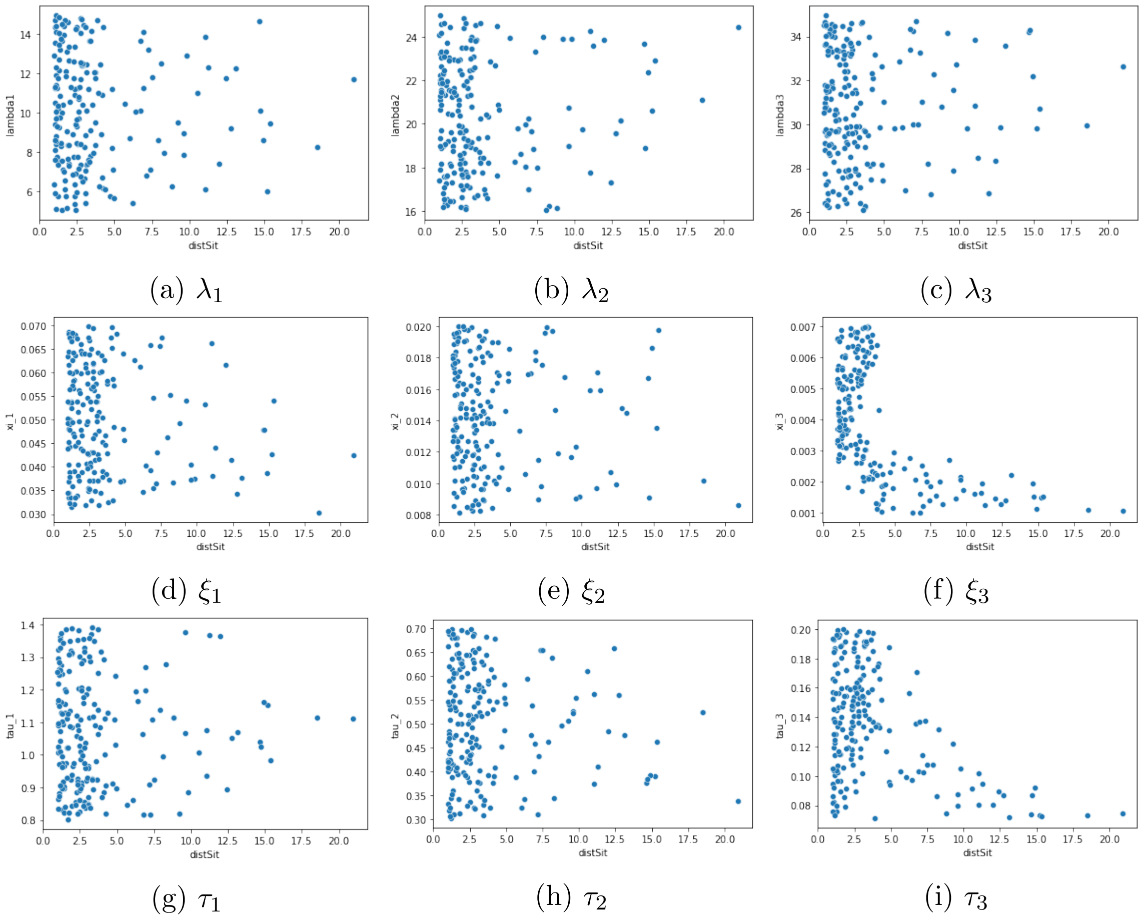
The results show ABC on nine parameters was not successful. Figure 4.1 shows an extremely uniform scatter for all parameters suggesting the posterior distribution is still uniform in a neighbourhood of the actual parameters. These results are an exact match to Grando's.

Three Parameter Estimation

Grando then decides to perform ABC on a subset of our nine parameters. She claims choosing only λ_i, only ξ_i or only τ_i does not yield better outcomes. She then tests for only λ_3,ξ_3 and τ_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 $\lambda_3 = 30$
- $\xi_1 = 0.05$, $\xi_2 = 0.01$ 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 λ_3 we change its value from 30 to $\lambda_3 \sim$ Unif(26,35). Similarly, when computing for ξ_3, we change its value to $\xi_3 \sim$ Unif(0.001,0.007) and when computing for τ_3 we change its value to $\tau_3 \sim$ Unif(0.07,0.2).



(a) λ_1

(b) λ_2

(c) λ_3

(d) ξ_1

(e) ξ_2

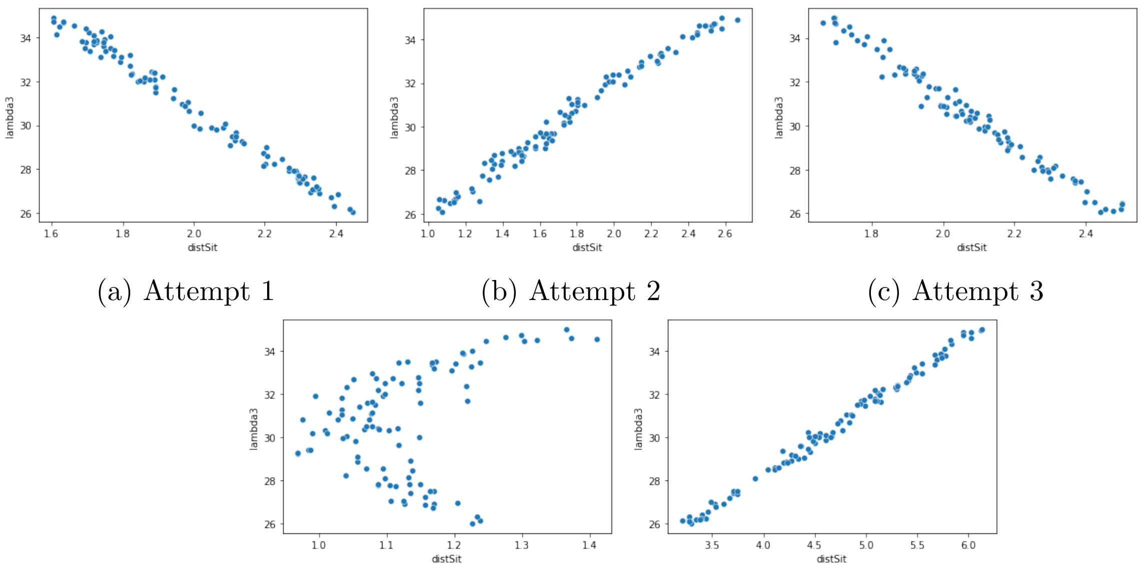
(f) ξ_3

(g) τ_1

(h) τ_2

(i) τ_3

Figure 4.1: Parameter values plotted against corresponding normalised Euclidean Distance between simulation and data means.



(a) Attempt 1

(b) Attempt 2

(c) Attempt 3

(d) Attempt 4

(e) Attempt 5

Figure 4.2: λ_3 values plotted against corresponding normalised Euclidean Distance between simulation and data means for 5 attempts.

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 4.2, 4.3 and 4.4 correspond to the ones found in Grando's *** REF** paper pages 78-80. We have come across unexpected results. Although the graphs show a similar pattern, there is an inconsistency in values.

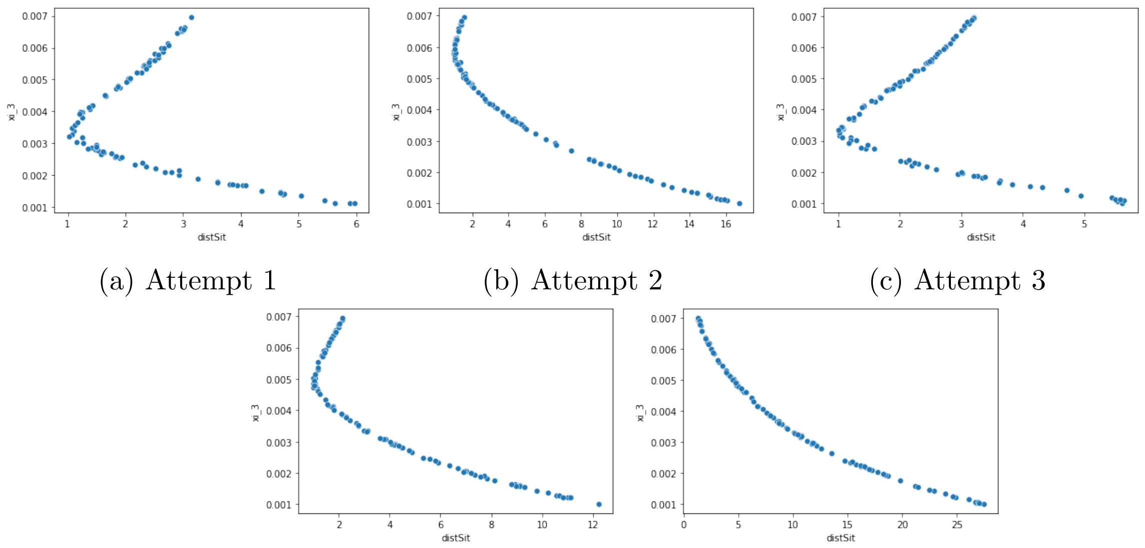
For λ_3 Grando's estimate was 80.436. This result was after running the algorithm in a larger interval. Although we have not done this, our data is sufficient to conclude. We expect a minimum distance; in other words, we expect the graph to be a parabola in y, the parameter values.

- The negative slope of 4.2a and 4.2c suggests the true value is larger than our upper limit of 35.
- The positive slope of 4.2b and 4.2e suggests the true value is lower than our lower bound of 26.
- 4.2d suggests the true value is somewhere within our interval of [26,35].

Not all three can be true as each contradicts the other. This suggests the value of λ_3 is changing through new attempts.

For ξ_3 Grando's estimate was 0.0027. From our results we can deduce the following:

- Both 4.3a and 4.3c suggest the true value is around 0.003.
- 4.3b and 4.3d suggest the true value is between 0.005 and 0.006.
- The negative gradient of 4.3e suggests the true value larger than our upper limit of 0.007.



(a) Attempt 1 (b) Attempt 2 (c) Attempt 3

(d) Attempt 4 (e) Attempt 5

Figure 4.3: ξ_3 values plotted against corresponding normalised Euclidean Distance between simulation and data means for 5 attempts.

Although it seems the value remains within the interval most of the time, it does not remain consistent. Again, as with λ_3, these changes in estimated value suggest the actual value may be changing through new attempts.

For τ_3 Grando's estimate was 0.0386. From our results we can deduce the following:

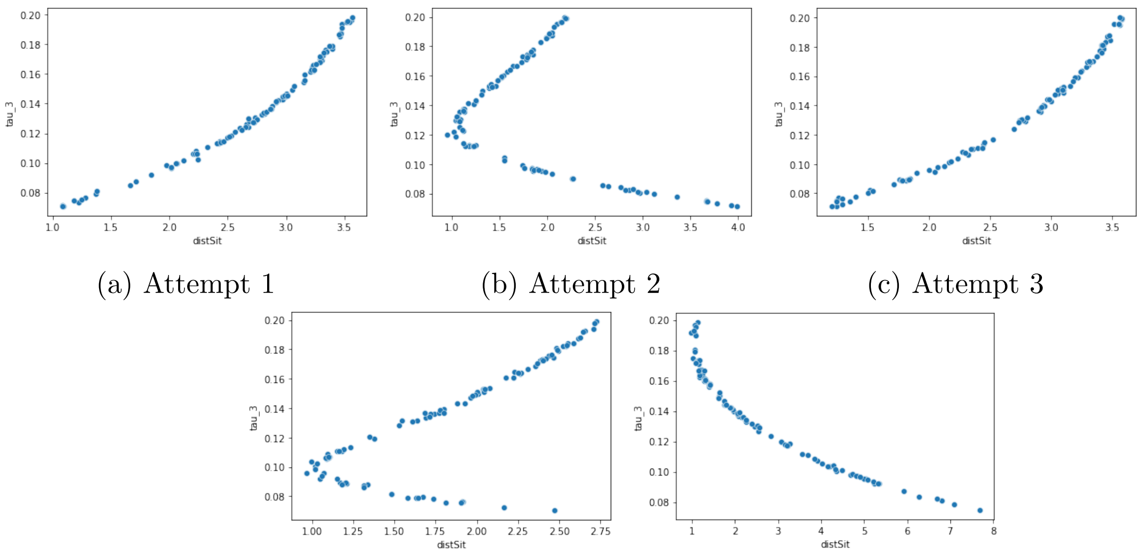
- The positive slope of 4.4a and 4.4c suggest the true value is below our lower bound of 0.07.
- 4.4b suggests the true value is around 0.13.
- 4.4d suggests the true value is around 0.10.
- The negative slope of 4.4e suggests the true value is above our upper limit of 0.2.

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

Adjusted Algorithm

From all 4.2, 4.4 and 4.3 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.

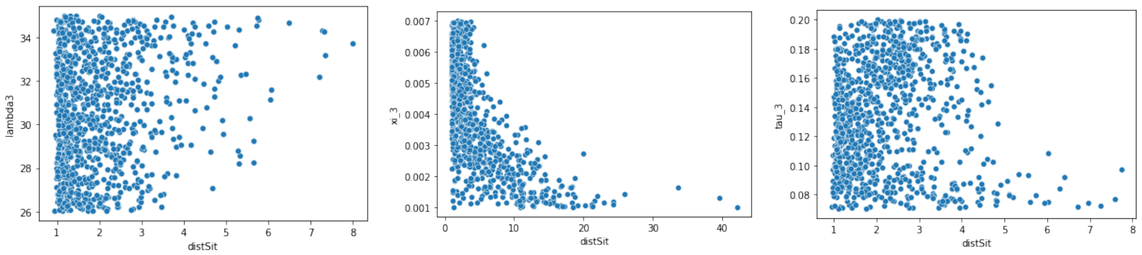
Looking closely at our simulation algorithm, one may notice for each attempt, the HMM parameters are generated by a uniform random variable and then kept the same for each simulation. In other words, they only change on new attempts. When we estimate the parameters using ABC, we are conditioning the values of the parameters we have preset, particularly the HMM parameters. Since these vary for each attempt, the condition for our



(a) Attempt 1 (b) Attempt 2 (c) Attempt 3

(d) Attempt 4 (e) Attempt 5

Figure 4.4: τ_3 values plotted against corresponding normalised Euclidean Distance between simulation and data means for 5 attempts.



(a) λ_3 (b) ξ_3 (c) τ_3

Figure 4.5: Parameter values plotted against corresponding normalised Euclidean Distance between simulation and data means for simulation calculated using adjusted ABC algorithm

algorithm 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, highlighting 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 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 by providing parameter estimates that fit 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 4.3.2, computation on the personal desktop took 652.842 seconds.

As we can see from Figures 4.5, we have a somewhat uniform scatter. This result suggests we cannot 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.

Chapter 5

Extending HMM based Rainfall Models

While we have demonstrated Grando's Methodology was flawed, her ideas have are yet to be explored. In this chapter, we will extend Grando's ideas and adapt her model.

5.1 Disc 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 not to cover all sites, suggesting this method will generate many potential discs and then reject most of them. This method 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 account for this change.
- Simulate the number and size of discs overall sites independently. Here we ignore the spatial data of the sites. This prevents us from understanding which sites the same disc's cover, but this is not important to us and can thus 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 each site independently, we decide to calculate parameters for each separately rather than combining these results.

5.2 Seasonal Effects

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 Cowpertwait's method, splitting the data into months. This method allows for simulating for particular months as well as a better fit for the overall data. Since our data is a time series of rainfall amounts with not much other information, 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. We set this value as the mean of the months' rainfall, ignoring the missing data to solve this.

5.3 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 simplify the data using dimensionality reduction methods, but this may limit our ability to fit our model well.

To address this problem, we revert to 2.5.1. 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 significant decrease in dimensions. However, our model does not have an observation matrix, which the Baum-Welch algorithm requires. We address this in the following chapter.

Chapter 6

Simple Rainfall HMM

We begin testing HMMs for rainfall by developing a simple HMM that we can fit using only Baum Welch. We then simulate using our model and analyse its similarities to the observation data.

6.1 Baum-Welch

The only thing separating Grando's rainfall model from a simple HMM is the lack of an observation matrix 2.1. In this section, we build this simpler model and then discuss finding the global optimum.

6.1.1 Building the Observation Matrix

To use Baum-Welch, we require an observation probability matrix; thus, to use it, we must create one. To create a finite matrix of probabilities, we require our data to split into finite groups. Fortunately, the algorithm increases in computation cost linearly with the dimensions of the observation matrix rather than square compared to 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 2.1, 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 the "Baum Welch" folder under "MyCode".

Note 6.1. The values of the alpha and beta helper functions become extremely small for a large sequence of observations. After about 1500 observations, these values become too precise to store in "long double" in our testing. This mathematically holds as the sum of the final alphas and betas represents the probability of observing the sequence 2.15, with a larger sequence, the probability naturally decreases. We avoid this problem by using a sequence of 1000 observations.

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

6.1.2 Local vs Global Optimum

Once again, from 2.5.1, we know that Baum-Welch converges to local optima. This suggests there may be a superior optimum, 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 initial distribution, transition matrix and observation matrix. While this does not guarantee we will find the global optima, it reassures us that our given optimum is reasonably suitable.

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

6.2 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 computation 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. In this section, we will graphically analyse our results.

6.2.1 Convergence

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

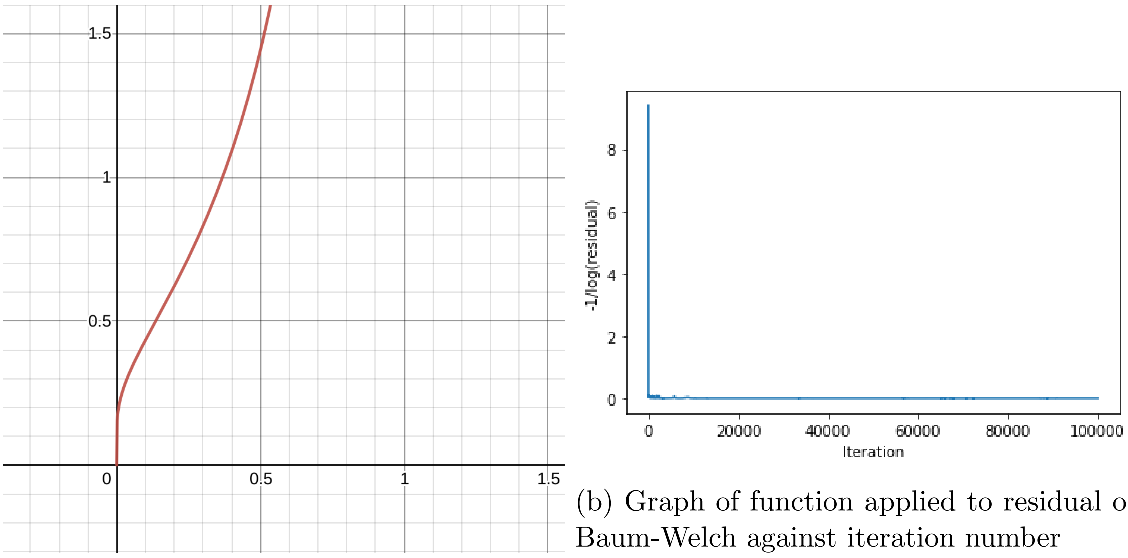
The values decreases quite rapidly, thus to create a visual representation, we used the function $y = \frac{-1}{ln(x)}$, where x is the residual, to scale the values. We can see in Figure 6.1a that our function is strictly increasing thus can be used for scaling.

Along with the CSV files containing the log, a graphical representation can be found in the Analysis Jupyter notebooks within each months folder. Figure 6.1b shows one such graphical representation for the convergence log of attempt 1 for month 0. We can see that the residual converges to 0 relatively quickly, suggesting we could reduce the number of iterations. The convergence itself confirms that our two matrices and our initial distribution contain values close to the local optima.

6.2.2 Selecting an Optimum

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

For each attempt, we count and record the frequency of occurrence of each integer 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



(a) Graph of the function $y = \dfrac{-1}{ln(x)}$

(b) Graph of function applied to residual of Baum-Welch against iteration number

Figure 6.1

difference between 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

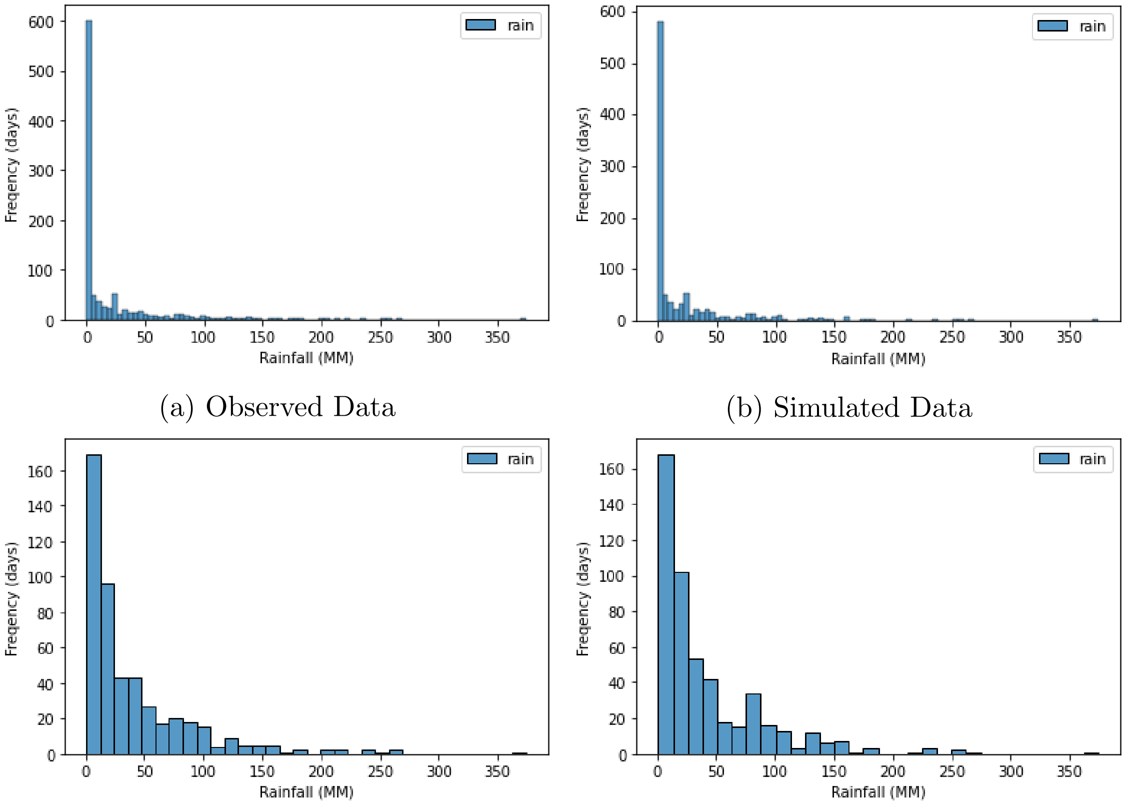
As such, we selected attempt 1 for our testing. The attempt selected for each month is in the below table.

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

6.2.3 Results

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

Figure 6.2 shows that the frequencies seem to be similar for both simulations and recorded data. Furthermore, more minor details such as the peak at around 25mm also remains consistent. While this is promising, 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) Observed Data

(b) Simulated Data

(c) Observed Data removing all 0mm days

(d) Simulated Data removing all 0mm days

Figure 6.2: Comparison of Observed data and Data simulated by HMM

Chapter 7

Generalising the Observation Matrix

Our HMM has an extremely large observation matrix. In this chapter, we aim to simplify this to a function of random variables that defines the whole matrix with only a handful of parameters and then solve for these parameters. We take inspiration from ideas presented by Cowperthwait and Grando.

7.1 Generalised Model

We have seen that HMM can 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 integer value groups, some groups have 0 probability. The data justifies this result, but it 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, 2mm is 0. This result may be accurate 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 represents the training data and helps simulate future rainfall patterns.

7.2 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 to have real-life justifications. We propose our model, which was developed by adjusting Grando's Model.

Proposition 7.1. *Given the system is in state i, the amount of rainfall in mm is given by*

$$\sum_{i=0}^{N} \sigma_i \tag{7.1}$$

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

7.3 Parameter Estimation

Now that we have a function, in this section, we estimate our parameters.

According to 7.1, we have two unknown parameters for each row of each observation matrix. In other words, for our example, we have six unknown parameters for each month. Fortunately, 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.

7.3.1 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 λ and τ 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 the expected frequency for 100000 simulations by multiplying the probability by the number of iterations. Store results in an array.

5. Find the difference between each frequency of each element in the simulation and the expectation array. Sum absolute value of each and take this value as residual forgiven λ and τ.

6. Repeat steps 1000 times forgiven λ and τ. Record λ and τ and residual in CSV file.

7.3.2 Estimation

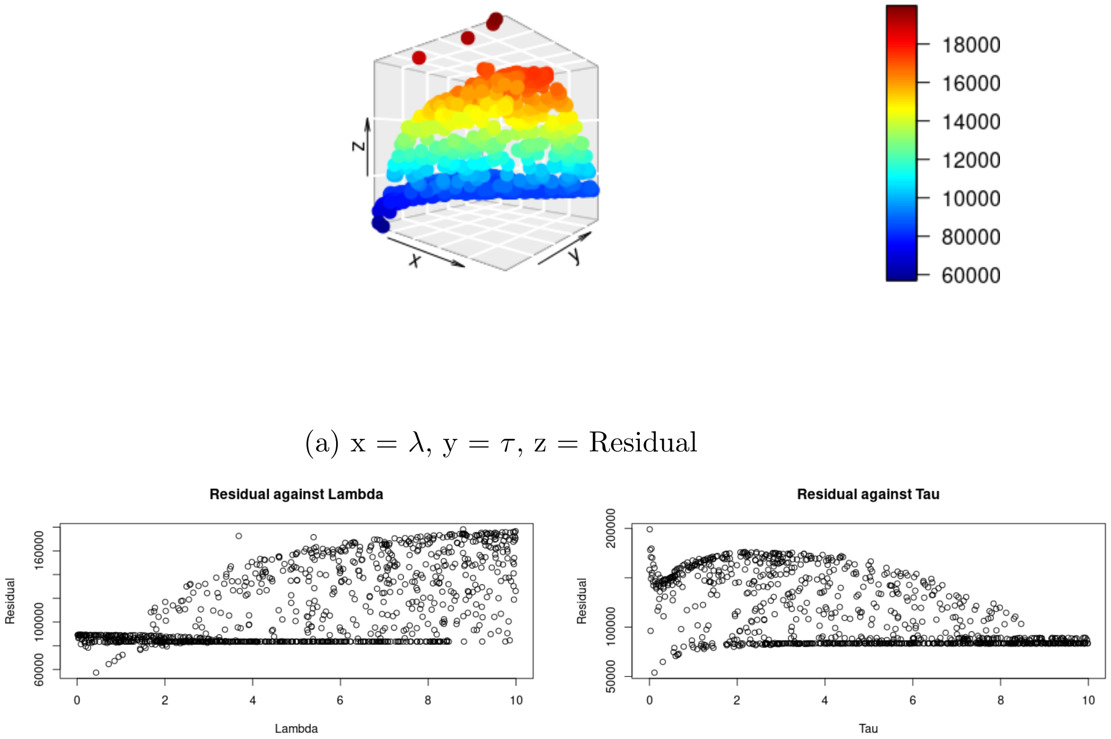
"pe" has been optimised to run over up to 24 threads. When given the 24 threads, the program finishes the above computation within 10-30 seconds.

"Analysis.R" is used to analyse the CSV file. It plots three graphs; a 3D graph of λ against τ against residual, λ against residual and τ against residual. We use these three graphs to find an approximate area of λ or τ 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. We found that both values tend to have their minimum residual points within these ranges through initial testing with random ranges. It is clear from Figure 7.1 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 7.2. 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 7.3. 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:

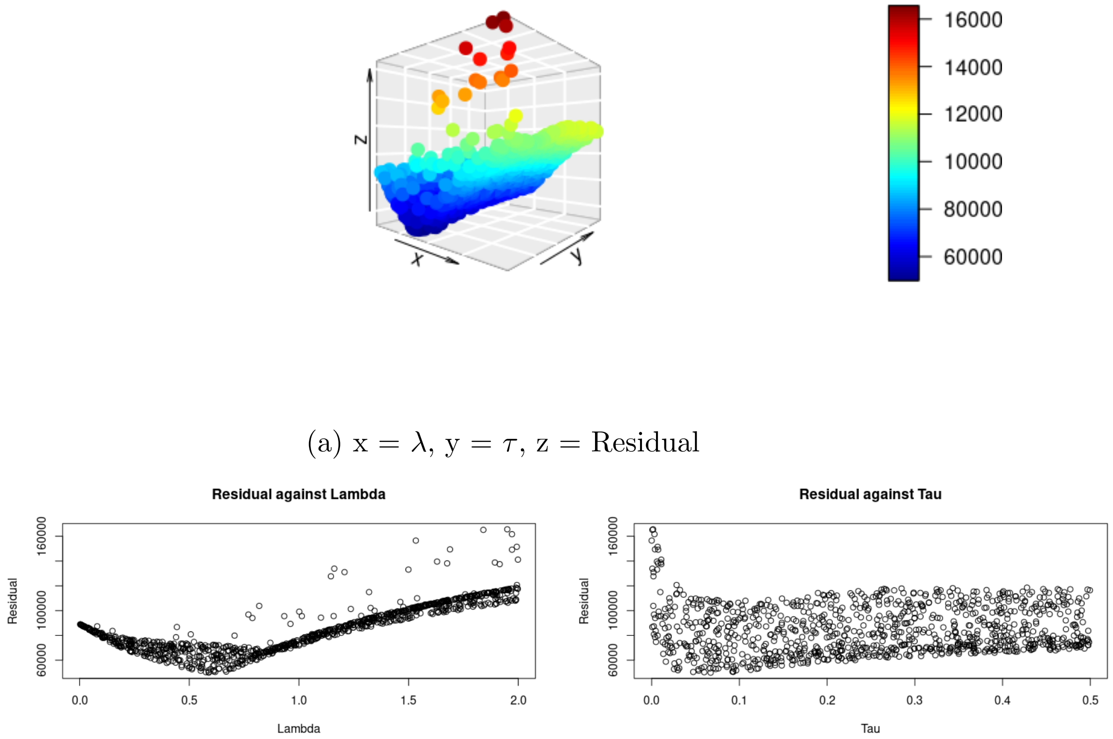


(a) x = λ, y = τ, z = Residual

(b)

(c)

Figure 7.1: First iteration of parmeter estimation. Parameters set to: λ between 0 and 10, τ between 0 and 10. Graphs show clear drop in residual where $\lambda \in [0, 2]$ and $\tau \in [0, 0.5]$.

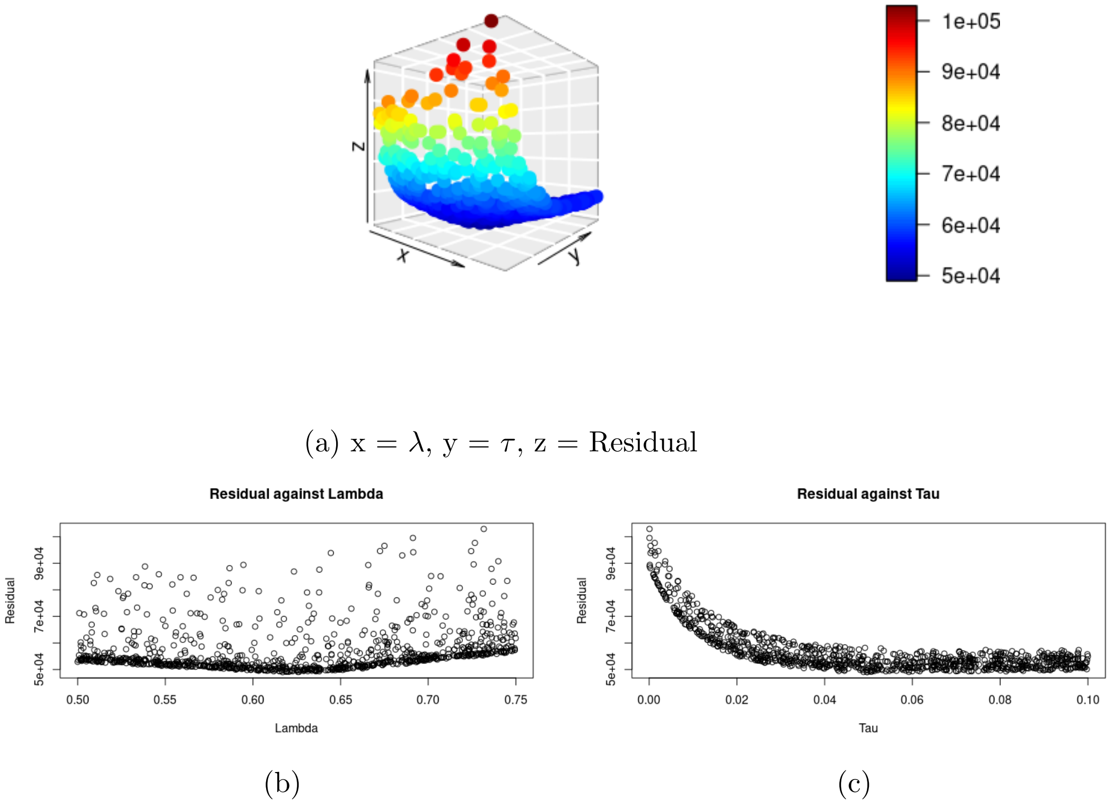


(a) x = λ, y = τ, z = Residual

(b)

(c)

Figure 7.2: First iteration of parmeter estimation. Parameters set to: λ between 0 and 2, τ between 0 and 0.5. Graphs show clear drop in residual where $\lambda \in [0.5, 0.75]$ and $\tau \in [0, 0.1]$.



(a) x = λ, y = τ, z = Residual

(b)

(c)

Figure 7.3: First iteration of parmeter estimation. Parameters set to: λ between 0.5 and 0.75, τ between 0 and 0.1. Graphs show minimum λ at around 0.625 and minimum τ at around 0.05.

Month	λ_1	λ_2	λ_3	τ_1	τ_2	τ_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 the table that while the values of τ 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 and apply more rigorous tests on the hmm only model.

Chapter 8

Results

This chapter finally brings together both the Hidden Markov only Model and the Generalised HMM for rainfall. We begin with discussing what data we have available before a graphical comparison of Cumulative Distribution Functions (CDFs) and analysing numerically using the Kolmogorov–Smirnov test 3.3.

8.1 Datasets

Before we begin testing, we summarise what data we have so far.

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 have 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 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 6 we also generated a simulation of size 1000 days. We will use this simulation for our testing now. From our parameter estimates in 7.3 we generated simulations of size 1000 using ”generator”, which can be found in the ”Model Generator” folder.

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

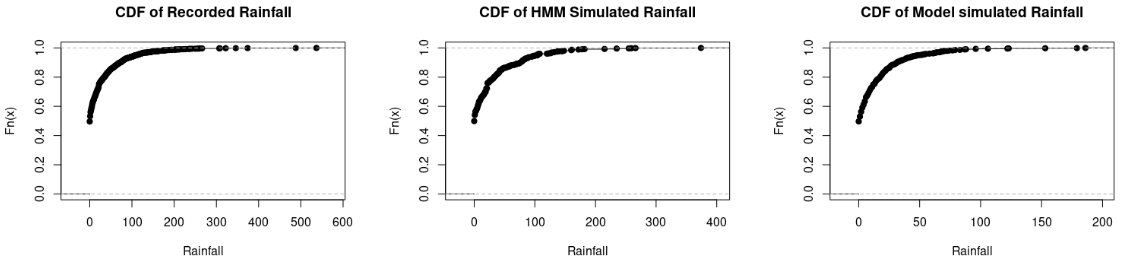
8.2 Analysis

In this section, we finally analyse both our simulations both graphically and numerically to help decide whether HMMs provide any use to rainfall modelling.

8.2.1 CDFs

We begin by comparing the empirical cumulative distribution functions of the observed data and both simulations. For the recorded data, we combine the training and test data.

From Figure 8.1 we can instantly see that neither the HMM only nor our generalised model simulates any extreme values observed in the recorded data. However, the shape of the graphs does seem similar. It is also clear that the rate increases much faster for both



(a) (b) (c)

Figure 8.1

recorded and HMM only datasets. This result suggests our generalised model may not be a good fit.

8.2.2 Kolmogorov–Smirnov tests

To obtain numerical results, we apply the Kologmorov-Smirnoff (KS) test for two samples 3.3. We apply this to both of our models against both training and test data. Our hypotheses are given as follows:

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

	Train	Train	Test	Test
Month	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 we rejected the null hypothesis for 4 out of 12 months. To justify this result, we require further testing. 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, in fact, a suitable optimum.

For our generalised model, we rejected the null hypothesis whenever tested against both training and test sets. This result suggests the model is not describing the rainfall data well.

Chapter 9

Future Research

Our testing in 8 suggests the HMM on its own does have potential. The model matched both the training and test data well. While it did struggle to explain extreme rainfall, this could be an area for further research. Furthermore, for our testing, we used only three random starts of Baum-Welch. Considering the incredibly high complexity of the parameter space, due to large matrices, one can expect there to be many more local optima than just three. As such, increasing this number would likely increase the likelihood of achieving a better fit by finding a superior optimum. Thus we are inclined to believe the performance of HMM rainfall models could be further improved.

Our generalised model was not successful. The model itself was chosen based on purely natural phenomenon and thus does not dismiss the idea that a generalised HMM model may be helpful. If we can find a function of random variables that perfectly fit the rows of the observation matrix, then hypothetically, we can expect matching performance to that of the HMM only model. While the HMM fits the observed data more closely, a generalised model will likely allow for improvements in out-of-sample testing, particularly at the extremes.

We tested a three-state model. However, this was an arbitrary choice. Future research may consider increasing this number as it may allow for a superior fit to the data. This change, of course, must be done after careful considerations of computational cost as the algorithms used in fitting are extremely sensitive to the increase in value of this parameter.

To conclude, can hidden Markov models be used to simulate rainfall? From our testing, we have insufficient evidence to suggest not. Further research into the three areas listed above combined with testing performance against other models should determine whether HMMs provide any advantage.

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`.

Cowpertwait, Paul (2009). “A Neyman-Scott model with continuous distributions of storm types”. In: *ANZIAM Journal* 51, pp. C97–C108.

Cowpertwait, Paul, Valerie Isham, and Christian Onof (2007). “Point process models of rainfall: developments for fine-scale structure”. In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 463.2086, pp. 2569–2587.

Cowpertwait, Paul, Jim Salinger, and Brett Mullan (2009). “A spatial-temporal stochastic rainfall model for Auckland City: Scenarios for current and future climates”. In: *Journal of Hydrology (New Zealand)*, pp. 95–109.

Cowpertwait, Paul SP (1994). “A generalized point process model for rainfall”. In: *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences* 447.1929, pp. 23–37.

— (1995). “A generalized spatial-temporal model of rainfall based on a clustered point process”. In: *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences* 450.1938, pp. 163–175.

— (1998). “A Poisson-cluster model of rainfall: some high-order moments and extreme values”. In: *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 454.1971, pp. 885–898.

— (2006). “A spatial–temporal point process model of rainfall for the Thames catchment, UK”. In: *Journal of Hydrology* 330.3-4, pp. 586–595.

— (2010). “A spatial-temporal point process model with a continuous distribution of storm types”. In: *Water Resources Research* 46.12.

Cowpertwait, PSP, CG Kilsby, and PE O’Connell (2002). “A space-time Neyman-Scott model of rainfall: Empirical analysis of extremes”. In: *Water Resources Research* 38.8, pp. 6–1.

Daley, Daryl J and David Vere-Jones (2007). *An introduction to the theory of point processes: volume II: general theory and structure.* Springer Science & Business Media.

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.

Grando, Debora (2019). “Data-driven models and random generators of rainfall”. MA thesis.

Markov, Andre (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`.

Ross, Sheldon M (2004). *Introduction to probability and statistics for engineers and scientists.* Elsevier.

Sharma, Jay (2020). “Hidden Markov Models And Their Applications In Stock Market Prediction”. MA thesis.

Toni, Tina et al. (2009). “Approximate Bayesian computation scheme for parameter inference and model selection in dynamical systems”. In: *Journal of the Royal Society Interface* 6.31, pp. 187–202.

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.