Modelling with Hawkes Processes

Daniel O'Shea
Department of Mathematics and Statistics
University of Limerick

Supervisor
Prof. James Gleeson

Report for a Final Year Project
BSc in Mathematical Sciences
April 7, 2020

Abstract

In this project, I will start by giving an introduction to Hawkes Processes, including some use cases. I will then review the state of the art in the mathematics of Hawkes Process. After developing a good grounding in the theory and practice of this branch of mathematics, I will investigate the possibility of modelling time series data using Hawkes Processes. In this report, I will introduce some definitions and formulas for the purpose of the reader's understanding.

Hawkes Processes are a class of Stochastic Processes that have the interesting property that they are "self-exciting", meaning that the intensity of the process increases for some time after a new arrival. This mimics what we see in the stock market where bursts of buy/sell activity for a stock increase the probability of more of the same activity in the near future. Hawkes Processes can be difficult to analyse, but they show some considerable promise as concise models in a variety of different fields.

Keywords: *Hawkes Process, Self-Exciting, Stochastic Processes.*

Contents

List of Figures

Chapter 1

Introduction to Hawkes Processes

I will start this project by following a review article Laub *et al.* (2015) [1] which gave a comprehensive review of the current state of Hawkes Processes. I have also reviewed other literature including Swishchuk (2017) [2], Crowley (2015) [3], Laub (2014) [6], and Rizoiu *et al.* (2017) [11] for guidance on how to approach writing about this topic.

I will begin, in Chapter 1, by introducing some basic mathematical concepts that must be understood to gain a grasp on the Hawkes process. In Chapter 2, I will introduce the theory of the Hawkes process and its conditional intensity function, as well as its practical applications from the literature. It is then that I will move onto the excitation function and branching process in Chapter 3. In Chapter 4, I will derive the log-likelihood function for the Hawkes process, which is used to estimate the parameters of a Hawkes process realisation. Chapter 5 is where I introduce the method of estimating parameters for a Hawkes process in order to gain a better insight into this method before I introduce the algorithm that will calculate the equations needed to complete this task automatically in Chapter 6. In the same chapter, I simulate a Hawkes process for a chosen set of parameters (and for different n values) and then estimate said parameters as a practical introduction to these algorithms. This leads me to investigate how different n values effect the accuracy of the parameter estimates in Chapter 7. After gaining a better understanding in what to look out for when estimating parameters, I introduce a new method for predicting the future cascade size of a Hawkes process using these estimated parameters later in the same chapter. In Chapter 8, I apply everything I've learnt thus far to investigating the applications of Hawkes processes on some of the examples laid out in Chapter 2 using my algorithms. These applications are to Internet Virality, Crime Data and Finance. In my Conclusion, I lay out my hopes for the future application of these types of algorithms and why. I also lay out some possible use cases for these algorithms based on what I've learnt about this process.

1.1 What is a Hawkes Process

First introduced by Alan G. Hawkes in 1971; the Hawkes process has been shown to have wide applications in fields such as neuroscience, genome analysis, seismology, insurance and finance.

As mentioned, Hawkes processes are a class of stochastic process. These processes are systems that change in accordance with probabilistic laws. Hawkes processes are a very interesting class of stochastic process because of their defining characteristic as being self-exciting. This means that each arrival increases the probability of future arrivals for some period of time after a new arrival. It models arrivals over time, the future evolution of this self-exciting point process is influenced by the timing of past events. The Hawkes process depends on its entire past history and has a long memory. Thus, the process is a non-Markovian extension of the Poisson process.

The Hawkes process is a counting process on a simple point process that has self-exciting property, clustering effect and long memory. I will define a *counting process* and *point process*, following the approach taken in Laub *et al.* (2015) [1], for the reader's understanding.

1.2 Counting Process and Point Process

Definition: Counting Process
A *counting process* is a stochastic process $N(t)$, with $t \geqslant 0$, taking positive integer values and satisfying $N(0) = 0$. It is almost surely (a.s.) finite, and is a right-continuous step function with increments of size $+1$.
A counting process $N(t)$ can be thought of as a cumulative count of the number of arrivals into a system up to t, the current time. The history of the arrivals up to time t is a filtration, (an increasing sequence of $\sigma - algebras$). This is denoted by $\mathcal{F}^N(t)$, $t \geqslant 0$.

The *counting process* is defined by the sequence of random arrival times, denoted $(t_1, t_2, ...)$, at which the counting process $N(t)$ has jumped with increment $+1$. The process defined by these arrival times is called a *point process*.

Definition: Point Process
A sequence of random variables $(t_1, t_2, ...)$, taking values in $[0, +\infty)$, is called a point process if $P(0 \leqslant t_1 \leqslant t_2 \leqslant ...) = 1$, and the number of points in a bounded region is a.s. finite.

The terms *counting process* and *point process* are often used interchangeably in much of the literature in this field. It is often left up to the reader to infer which process is being discussed based on the context. However, I will give a summary of these terms:

- The series of random arrival times $\mathbf{T} = (t_1, t_2, t_3, t_4, ...)$ as seen on the $x - axis$ below is

called a *point process*.

- The graph with increments of +1 corresponding to each new arrival is the *counting process* associated with this *point process*.

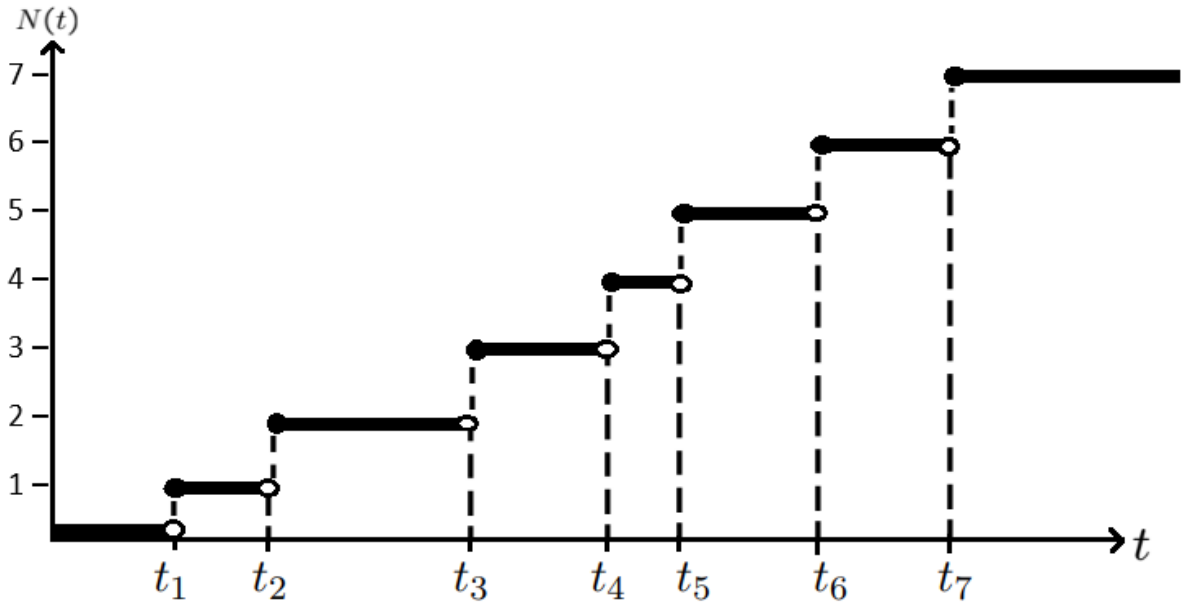


Figure 1.1: An example of a counting process $N(t)$ as derived from a point process realisation $(t_1, t_2, ...)$. (Diagram created by author, following the style of Fig. 1 of [1].)

Chapter 2

The Hawkes Process

2.1 Hawkes Conditional Intensity Function

I will define the *conditional intensity function* here for the readers understanding. Note, we will return to this topic in Section 2.3 for a more in-depth look.

Definition: Conditional Intensity Function [6]
Consider a counting process $N(t)$ with associated histories $\mathcal{H}(t)$. If a function $\lambda^*(t)$ exists such that

$$\lambda^*(t) = \lim_{h \downarrow 0} \frac{\mathbb{E}[N(t+h) - N(t)|\mathcal{H}(t)]}{h} \tag{2.1}$$

which only relies on information of the counting process in the past, then $\lambda^*(t)$ is called the *conditional intensity function* of $N(t)$.

The Hawkes Process is defined by its conditional intensity function,

$$\lambda^*(t) = \lambda + \sum_{t_i < t} \mu(t - t_i), \tag{2.2}$$

where λ is the background intensity, μ is the excitation function, and $(t_1, t_2, t_3, ..., t_k)$ denote the times of past arrivals into the system up to time t.

This is the standard and most intuitive form of the Hawkes conditional intensity function. The structure of the conditional intensity function $\lambda^*(t)$ is easy to understand as it has only two main components. λ is the background intensity, for some $\lambda > 0$ and μ is the excitation function, for some $\mu : (0, \infty) \to [0, \infty)$.

The background intensity, λ, is the constant stable intensity of arrivals on the system. Arrivals occur according to this intensity at the beginning of the process before any arrivals have occurred, then the system will see additional intensity brought on by the effect of new arrivals, the specifics of this additional intensity is determined by the second term.

The second term, $\sum_{t_i<t} \mu(t-t_i)$, is the additional influence on the intensity from the sum of excitations in the past up to time t. The function μ is defined such that the larger $(t-t_i)$ is, the smaller its effect is on $\lambda^*(t)$. The influence of previous arrivals on the intensity will wear off as they get older, this distance in time is accounted for in the term $(t-t_i)$. As this term grows, its effect on μ (and in turn, μ's effect on $\lambda^*(t)$) will diminish. The speed with which it diminishes, as well as the initial impact of a new arrival, will depend on your choice of excitation function. The configuration of the excitation function (sometimes called the response function) will be discussed further in Chapter 3.

Note:
One could have a process where $\lambda = 0$. So, you would have an initial burst, and then the background intensity will no longer carry the process forward and bring in new arrivals. In this process, the conditional intensity function drops to zero and stays there. This type of Hawkes process could be used to model viral trends, such as popular tweets and viral videos.

We will assume that $\mu \neq 0$ as this would change the conditional intensity function to a homogeneous Poisson process.

In summary, we can interpret equation 2.1 as follows: New arrivals occur according to a background intensity λ, and any new arrival into the system will cause the intensity to increase by $\mu(0)$. This additional intensity will fade away according to the choice of excitation function $\mu(t)$. The fact that an arrival causes the intensity of future arrivals to increase means that this process is a self-exciting process.

Notice how, in Fig. 2.1, the conditional intensity has dropped back down to the background intensity λ at the end of the interval between t_4 and t_5, where there were no new arrivals to excite the process. Also, the effect of previous arrivals on μ has decreased to zero due to the significant amount of time that has passed since the previous arrivals. At this point, we could say that the excitation function μ has no effect on the conditional intensity function. Thus, this process, at this point, is a Poisson process (defined in Appendix A.1). At this point, the history of arrivals has no effect on the next arrival.

Keep in mind; it can be difficult to identify how a new arrival came into the system. Be it from λ, like an arrival in line with a Poisson process, or from the excitation function μ, the effect of

recent previous arrivals. However, in this case, it is fair to say it came from λ.

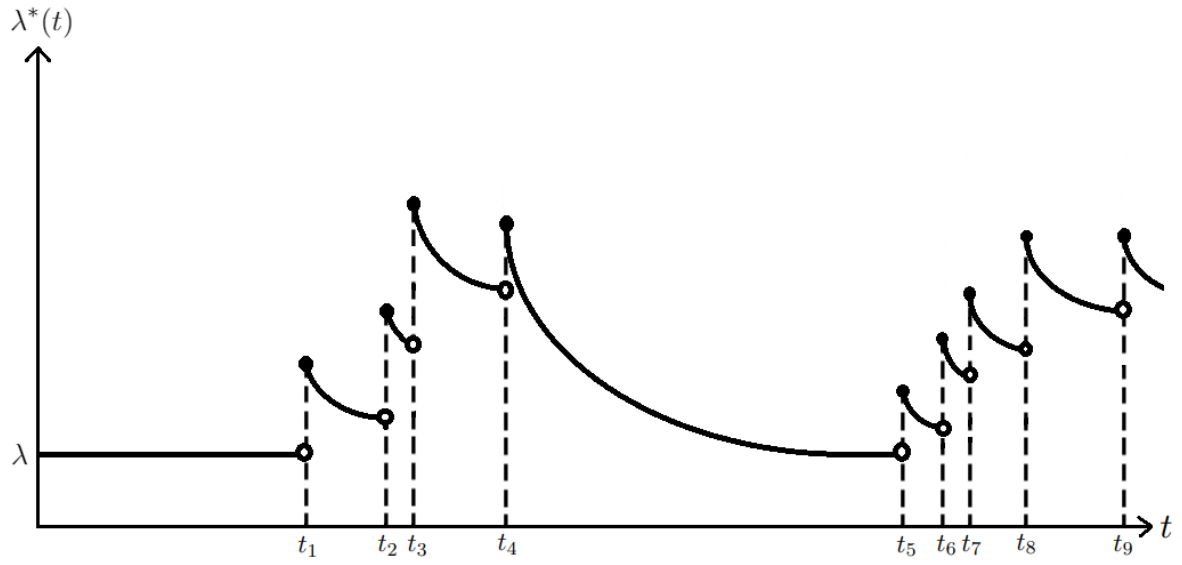


Figure 2.1: An example of a conditional intensity function $\lambda^*(t)$ with new arrivals into the system at times $(t_1, t_2, ...)$ displaying the self-exciting property. (Diagram created by author, following the style of Fig. 2 of [1].)

2.2 Deriving Hawkes Conditional Intensity Function

To facilitate the reader's understanding, I decided to first introduce the conditional intensity function in the form I will be implementing it, and leave the mathematics leading to its finished form until now. I hope this decision makes the Hawkes process easier to understand and gives those who are not interested in the mathematics behind deriving the conditional intensity function the option to skip the following two sections.

To begin our derivation of the conditional intensity function, we will continue on from our definition of the *counting process* and *point process* in Section 1.2. The conditional intensity function is related to the conditional arrival distribution. This is the distribution function of the next arrival time of the point process conditional on the timing of past arrivals. We will define the conditional c.d.f. and p.d.f. of the next arrival time after our final observed arrival k, denoted as T_{k+1}, conditional on the history up until the last arrival at time u, $\mathcal{H}(u)$ following [1] as

$$F^*(t|\mathcal{H}(u)) = \int_u^t \mathbb{P}(T_{k+1} \in [s, s+ds]|\mathcal{H}(u))ds = \int_u^t f^*(s|\mathcal{H}(u))ds. \tag{2.3}$$

This is the probability of the $k+1$ event happening in an infinitesimal time interval given the history of arrivals. Then integrating over time gives the cumulative distribution. We find the

joint p.d.f. using the chain rule as,

$$f(t_1, t_2, t_3, ..., t_k) = \prod_{i=1}^{k} f^*(t_i|\mathcal{H}(t_{i-1})). \tag{2.4}$$

The quantities $F^*(t)$ and $f^*(t)$ feed into the formulation of the conditional intensity function $\lambda^*(t)$, I will talk more on this below, but first I feel it may be helpful to gain a more intuitive understanding of these equations and their terms.

The intuitive understanding of equation 2.3 is not obvious at a glance. This equation describes a situation where we think of ourselves as being at time t, and we have observed arrivals up to time u ($u < t$), the time at which arrived our final observation T_k. We want to know the probability of the T_{k+1} event having arrived between times u and t. This probability is found by taking times s between u and t and asking what is the probability that T_{k+1} arrived between s and $s + ds$. This is a very small interval, so we are essentially looking at time s and finding the probability of the event occurring instantaneously at that time. Then we want the sum of the probabilities that the next event happened anywhere between u and t. These probabilities are independent, so this is achieved by taking the integral over u to t. Again, this is all conditional on the history of arrivals up to time u, $\mathcal{H}(u)$. So $f^*(s)$ is defined to be the probability, at time s, of observing the next event occurring instantaneously. (Note here I omitted the term specifying that s is conditional on the history, this is done for brevity and will continue in the following chapters.)

Equation 2.4 describes the probability of seeing a given realisation, i.e., the probability that we will have seen a given series of events calculated as follows: given the history up until t_1, what is the probability that I observe the next event at time t_2. These are observed to be independent, so each instance is multiplied to find the probability of the series of events, the point process realisation.

It is the relationship between $f^*(t)$ and $F^*(t)$ that defines the conditional intensity function, we can think of this as the expected rate of arrivals conditioned on the history $\mathcal{H}(u)$. The conditional intensity function, if it exists for a given point process, determines the probability structure of the point process uniquely, as described in Daley and Vere-Jones (2003) [4], and is defined as

$$\lambda^*(t) = \frac{f^*(t)}{1 - F^*(t)}. \tag{2.5}$$

This is the conditional intensity function in terms of $f^*(t)$ and $F^*(t)$. We can break down this equation to the form it is usually given in formal definitions following the approach taken in Rasmussen (2018) [5] to get equation 2.1 above and 2.6 below (same equation given twice for the reader's understanding).

2.3 Conditional Intensity Function and the Hawkes Process

I will now show how the conditional intensity function fits into our definition of a Hawkes process. A reminder of the formal definition of a conditional intensity function is as follows.

Definition: Conditional Intensity Function [6]
Consider a counting process $N(t)$ with associated histories $\mathcal{H}(t)$. If a function $\lambda^*(t)$ exists such that

$$\lambda^*(t) = \lim_{h \downarrow 0} \frac{\mathbb{E}[N(t+h) - N(t)|\mathcal{H}(t)]}{h} \tag{2.6}$$

which only relies on information of the counting process in the past, then $\lambda^*(t)$ is called the *conditional intensity function* of $N(t)$.

The expected number of events in an interval is proportional to the time interval as the size of the interval, h, shrinks to zero. This function shows the conditional intensity function as the instantaneous rate of change with respect to $N(t)$.

Definition: Hawkes Process
Consider a counting process with associated history $\mathcal{H}(t)$ that satisfies

$$\mathbb{P}(N(t+h) - N(t) = m|\mathcal{H}(t)) = \begin{cases} \lambda^*(t)h + o(h), & m = 1 \\ o(h), & m > 1 \\ 1 - \lambda^*(t)h + o(h), & m = 0 \end{cases}.$$

Intuitively, this is the probability that the number of arrivals that occur in the interval $(t, t+h)$ is m, given the history of the counting process up to t. We can see that the probability that $m = 1$ (one arrival) is given by the conditional intensity function weighted by h plus some error of h. The probability of the last case (having no arrivals) is given by the complementary probability of this. The chance of having more than one arrival is given by the error associated with h.

A Note on Non-Linear Hawkes Processes
The above is technically the definition of a *linear* Hawkes process. I will not touch on the *non-linear* case in this paper; I would, however, refer you to *Nonlinear Hawkes Processes* by Lingjiong Zhu (2013) [7] for further reading on this. Linear Hawkes processes are much better studied in the literature compared to the non-linear case. As mentioned by Zhu (2014) in [8]; the literature examines Hawkes processes almost exclusively in the linear case. Another reason mentioned for the lack of research on the non-linear Hawkes process is because of the lack of immigration-birth representation and computational tractability. For these reasons, this paper will focus exclusively on the linear form.

2.4 Applications of Self-Exciting Point Processes

Remembering that the Hawkes process has a long memory, clustering effect, self-exciting property and is, in general, non-Markovian, we can think of processes that follow these conditions for potential applications.

In the example of equity prices, we know the decision whether or not to buy a stock is dependent on where the stock's price is now, and its history of price movements. Market makers exhibit a herding behaviour, where one person makes a trade order to buy a stock, the price moves up, demand increases, and the process becomes self-fulfilling. In the same way, selling a large quantity of stock lowers the price of that stock which would lead to other traders (or trader's algorithms) to decide to sell before the price drops any lower. I will investigate the application of the Hawkes process on finance in Section 8.3.

After an earthquake, the resulting seismic disturbance leads to further tremors, called aftershocks, due to the earth's crust being disturbed. The distribution of their magnitudes and arrival times has been proven to be relatively consistent in Reinhart (2018) [9]. Self-exciting point processes have been used to model this behaviour.

In the same way cases of the flu can quickly spread through a school or office, epidemiologists have used self-exciting point processes to model epidemic forecasts; unfortunately, the data are rarely tracked with accurate time and location tags. In a specific example, Meyer *et al.* (2012) [10] introduced a self-exciting point process model for a form of meningitis. Their model accounted for unaffected carriers who could pass the disease on to new people, acting as a constant *background intensity* (λ) of new infections. They also modelled infections passed from an affected carrier to new victims. This case would be more contagious, acting as our *excitation function* (μ) in the model.

Similar to the virality of a disease, the virality of trends can be modelled in the same way. Twitter is a great example of this virality, when you see a tweet, you have the option to retweet it, i.e., to share it with your followers. Then, each of your followers will have the same option to retweet. If the tweet in question is particularly captivating, this could lead to a retweet cascade that bears all the same characteristics as a Hawkes process. Rizoiu *et al.* [11] modelled information diffusion through retweet cascades using a Hawkes process. I investigate this very idea in Section 8.1.

It is observed that crime data often exhibit clustering effects. A fight between rival gangs could cause more criminal retaliations. Burglars often return to burgle in areas where they have made successful burglary attempts before. In [12], Mohler *et al.* used self-exciting point process models from seismology studies and applied them to data from the Los Angeles Police Department to

illustrate the implementation of these models in the context of urban crime. This is the concept of my investigation in Section 8.2.

In [13], Reynaud-Bouret and Schbath used a Hawkes process to provide a new method to detect either avoided or favoured distances between genomic events along sequences of DNA. This was done by treating the DNA sequence as an integer number line and noticing instances along this discrete sequence.

In an alternative general example, consider the case where a driver gets a speeding ticket. In this case, the arrival of a speeding ticket for an individual driver could come at random intervals, but the driver would be much less likely to break the speed limit for some time interval after receiving a speeding ticket. This heightened caution and awareness of speed, reducing the probability of a new arrival (for some period of time), leads this process to be $self\text{-}regulating$, as opposed to self-exciting. This causes the overall intensity to drop and would slowly rise back up to the background intensity leading to arrivals appearing temporally regular. Given the area of study and the greater area of application, this paper will focus on self-exciting processes. The decay, rise and return to background intensity is all dependent on your choice of *excitation function* and your choice of parameters within it. The choice of parameters will be discussed in Chapter 4, while the specific type of excitation function will be discussed in the next chapter.

Chapter 3

The Mechanics of Hawkes Processes

What happens to the intensity when a new arrival occurs and how the intensity changes after which is governed by the excitation function. In the cases we will study (self-exciting processes) the intensity will rise immediately following an arrival and decay back to the background intensity. The initial rise and the speed of decay will depend on your choice of excitation function; it is these choices that we will discuss in the following section.

3.1 Specifying the Excitation Function

A reminder of the structure of the conditional intensity function $\lambda^*(t)$ given in Section 1.2 may be helpful here:

$$\lambda^*(t) = \lambda + \int_0^t \mu(t-u)dN(u), \tag{3.1}$$

which becomes

$$\lambda^*(t) = \lambda + \sum_{t_i<t} \mu(t-t_i), \tag{3.2}$$

where t_i represents the arrival times up to time t. The excitation function μ can take on multiple forms to manage the change of intensity over time following new arrivals. The forms that μ can take on are often called *kernels*.

A common choice for excitation function is one of exponential decay, as the name suggests, the intensity will decay exponentially following the initial increase after an arrival. An example of what this type of conditional intensity function would look like is given in Fig. 2.1. In this case $\mu(t) = \alpha e^{-\beta t}$, where your choice of $\alpha, \beta > 0$ will discussed in Chapter 4. In the conditional

intensity function, this becomes

$$\lambda^*(t) = \lambda + \int_{-\infty}^{t} \alpha e^{-\beta(t-s)} dN(s) = \lambda + \sum_{t_i<t} \alpha e^{-\beta(t-t_i)}. \tag{3.3}$$

As the term $(t - t_i) = 0$ at the time of the new arrival, the term $e^{-\beta(t-t_i)} = 1$, this means that each arrival into the system instantaneously increases the intensity by α, after which the intensity decays exponentially at a rate of β. The choice of this excitation function is often referred to as the Hawkes process with *exponentially decaying intensity*, and is probably the most common choice.

Another common choice is a *power-law function*, this gives us

$$\lambda^*(t) = \lambda + \int_{-\infty}^{t} \frac{k}{(c + (t-s))^p} dN(s) = \lambda + \sum_{t_i<t} \frac{k}{(c + (t-t_i))^p}. \tag{3.4}$$

This will usually decay less rapidly than the exponential kernel. This requires specification of three parameters which are positive scalars k, c, and p.

These are the two most commonly used excitation functions. For an introduction on some other choices of kernel, I would recommend Lima and Choi [14], or Liniger [15].

3.2 The Branching Process

We can understand the stability of a Hawkes process more easily if we view it as a branching process. The branching process is a stochastic process where particles arrive into a system, and each particle lives for a certain amount of time, and then creates children. The number of children had comes from the distribution that defines the branching process. In a standard branching process, we start with a single particle, and each event thereafter is a descendant particle of that first particle. The problem here is that it starts with one particle and does not allow new particles that are not descendants of the first to randomly arrive into the system.

So here we consider an immigration-birth representation where we can have particles enter the system by immigration (like a random arrival in line with a homogeneous Poisson process) or by birth (this would act like a response by the excitation function). Now we can return to our question on the stability of this process. We want to know if this process will grow or decay over time, this question is answered by the branching ratio; this is the average number of children per particle. It is easy to understand in a real-world context that if the average number of children per person is less than 1, the population would decay over time, and if the average is greater than 1, the population would grow over time.

We can gain an understanding of the stability of a particular branching process by investigating the branching ratio. This is calculated from the excitation function μ. We understand that here $\mu(s)ds$ is the probability that a child particle will be born at a time s after its parent's birth. Then the expected total number of children born from a single particle is

$$\int_0^\infty \mu(s)ds, \tag{3.5}$$

taken for the exponential kernel is

$$\int_0^\infty \alpha e^{-\beta s}ds, \tag{3.6}$$

this becomes

$$\frac{\alpha}{\beta}. \tag{3.7}$$

Now we can consider the stability of the branching process in terms of α and β as

$$\frac{\alpha}{\beta}\begin{cases} < 1 & Subcritical, \\ = 1 & Critical, \\ > 1 & Supercritical. \end{cases}$$

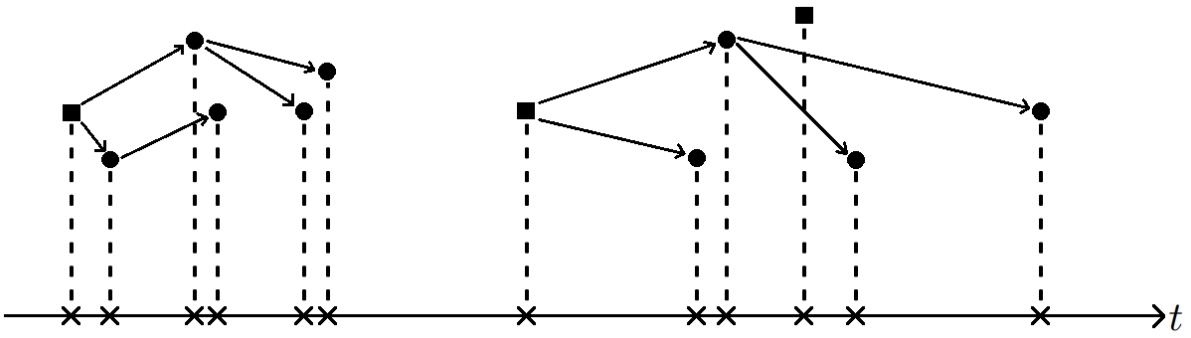


Figure 3.1: Hawkes process represented as an immigration-birth representation branching tree. Squares indicate arrivals by immigration, and circles indicate children/descendants of previous generations, the crosses at the bottom denote the resulting point process. (Diagram created by author, following the style of Fig. 4 of [1].)

In Chapter 4, we will find estimates for α and β. We can then use these measurements for stability to gain insights into the process we are working with. For example, we would expect the branching ratio α/β to be < 1 if the process is not exponentially exploding.

Chapter 4

The Mathematics of Parameter Estimation

In this chapter, I will discuss the problem of generating estimates for the parameters of the conditional intensity function. Given a data set that we assume could be modelled by a Hawkes process with a particular excitation function, we want to generate parameter estimates for λ, α and β. We will call these $\hat{\boldsymbol{\theta}} = (\hat{\lambda}, \hat{\alpha}, \hat{\beta})$, of course, the "hat" symbol here denotes an estimate. This will be estimated over some finite set of arrival times $\mathbf{t} = (t_1, t_2, t_3, ..., t_k)$.

4.1 Introduction to the Likelihood Function

As mentioned, an important part of building a Hawkes process model is fitting parameters to the conditional intensity function. The method we consider is by *maximum likelihood estimation*. This begins by deriving the likelihood function. Through an optimisation procedure, we can find the optimum parameters for the conditional intensity function by maximising the likelihood function, or equivalently, minimising the negative log-likelihood function. This is all with the aim of maximising the likelihood of seeing the given data given the estimated parameters. Hence the name *maximum likelihood estimation*.

4.2 Deriving the Likelihood Function

In this section, I will follow the approaches taken in [1] and [3] to derive the *likelihood function*. To avoid redundancy, I will omit the arguments $\hat{\boldsymbol{\theta}}$ and $\mathbf{t}$. This will give the following abbreviated functions: $L = L(\hat{\boldsymbol{\theta}}; \mathbf{t})$, $l = l(\hat{\boldsymbol{\theta}}; \mathbf{t})$, $\lambda^*(t) = \lambda^*(t; \mathbf{t}, \hat{\boldsymbol{\theta}})$ and $\Lambda(t) = \Lambda(t; \mathbf{t}, \hat{\boldsymbol{\theta}})$.

Let $N(t)$ be a point process on the interval $[0, T]$ for $T < \infty$ and let $(t_1, t_2, t_3, ..., t_k)$ denote a

set of event times associated with $N(t)$ during the period.

Using what I've learned from Statistical Inference, I know that the likelihood, L, of the function $f(t)$ is given by

$$L = f(t_1, t_2, t_3, ..., t_k) = \prod_{i=1}^{k} f^*(t_i). \tag{4.1}$$

Therefore, I want to isolate $f^*(t)$ and take it from there, so I will write this function in terms of the *conditional intensity function* like in Section 2.2. In equation 2.3 in Section 2.2 I've shown how $f^*(t) = \frac{d}{dt}F^*(t)$, using this we now get

$$\lambda^*(t) = \frac{f^*(t)}{1 - F^*(t)} = -\frac{d}{dt}\log\Big(1 - F^*(t)\Big), \tag{4.2}$$

this is because

$$-\frac{d}{dt}\log\Big(1-F^*(t)\Big) = -\frac{1}{1-F^*(t)}\left(\frac{d}{dt}(1-F^*(t))\right) = -\frac{1}{1-F^*(t)}\left(-\frac{d}{dt}F^*(t)\right) = \frac{1}{1-F^*(t)}f^*(t). \tag{4.3}$$

I will denote the last known event time before t as t_k, then integrating both sides over (t_k, t) we get

$$\int_{t_k}^{t} \lambda^*(u)du = \int_{t_k}^{t} -\frac{d}{dt}\log\Big(1-F^*(t)\Big) = -\log\Big(1-F^*(t)\Big)\Big|_{t_k}^{t} = \log\Big(1-F^*(t)\Big) - \log\Big(1-F^*(t_k)\Big). \tag{4.4}$$

Here, $F^*(t_k) = 0$, as $T_{k+1} > t_k$. This is because multiple arrivals cannot occur at the same time in a simple point process given $\mathcal{H}(u)$ includes the event at t_k. Therefore,

$$\int_{t_k}^{t} \lambda^*(u)du = -\log\Big(1 - F^*(t)\Big). \tag{4.5}$$

Taking the exponential of both sides, we can isolate $F^*(t)$ like such

$$F^*(t) = 1 - \exp\left(-\int_{t_k}^{t} \lambda^*(u)du\right), \tag{4.6}$$

and differentiating both sides to find $f^*(t)$ in the fashion above (remembering that $f^*(t) = \frac{d}{dt}F^*(t)$) we find

$$\begin{aligned}
\frac{d}{dt}F^*(t) &= \frac{d}{dt}\Big(1 - \exp\Big(-\int_{t_k}^{t}\lambda^*(u)du\Big)\Big) \\
\implies f^*(t) &= 0 - \frac{d}{dt}\exp\Big(-\int_{t_k}^{t}\lambda^*(u)du\Big) \\
\implies f^*(t) &= -\Big((-\lambda^*(t))\exp\Big(-\int_{t_k}^{t}\lambda^*(u)du\Big) \\
\implies f^*(t) &= \lambda^*(t)\exp\Big(-\int_{t_k}^{t}\lambda^*(u)du\Big). \qquad (4.7)
\end{aligned}$$

Now that we have isolated $f^*(t)$, we can apply the likelihood, L, to the function in the way described in equation 4.1 as follows

$$\begin{aligned}
L &= \prod_{i=1}^{k} f^*(t_i) \\
&= \prod_{i=1}^{k} \lambda^*(t_i)\exp\Big(-\int_{t_{i-1}}^{t_i}\lambda^*(u)du\Big) \\
&= \lambda^*(t_1)\exp\Big(-\int_{t_0}^{t_1}\lambda^*(u)du\Big) * \lambda^*(t_2)\exp\Big(-\int_{t_1}^{t_2}\lambda^*(u)du\Big) * ..... \\
&= \Big[\prod_{i=1}^{k}\lambda^*(t_i)\Big]\exp\Big(-\int_{t_0}^{t_1}\lambda^*(u)du - \int_{t_1}^{t_2}\lambda^*(u)du - \int_{t_2}^{t_3}\lambda^*(u)du.....\Big). \qquad (4.8)
\end{aligned}$$

Notice that the integral limits are a telescoping series starting from t_0 and ending at t_k such that the limits of the integral can now be taken as from 0 to t_k

$$L = \Big[\prod_{i=1}^{k}\lambda^*(t_i)\Big]\exp\Big(-\int_{0}^{t_k}\lambda^*(u)du\Big). \qquad (4.9)$$

Finally, we will consider the case where the process is observed over some time period $[0, T] \supset [0, t_k]$. This means that the likelihood includes the possibility of observing no arrivals in the time period $(t_k, T]$, where we then have

$$L = \Big[\prod_{i=1}^{k} f^*(t_i)\Big]\Big(1 - F^*(T)\Big). \qquad (4.10)$$

As $F^*(T)$ is the probability an event happened by time t, $(1 - F^*(T))$ is the probability that no event happened by this time. Therefore equation 4.10 is this probability multiplied by the likelihood $\prod_{i=1}^{k} f^*(t_i)$.

Here I can use the equivalence established in equation 4.6 to get

$$1 - F^*(T) = \exp\Big(- \int_{t_k}^{T} \lambda^*(u)du \Big), \tag{4.11}$$

and as we've seen in equations 4.8 and 4.9

$$\prod_{i=1}^{k} f^*(t_i) = \Big[\prod_{i=1}^{k} \lambda^*(t_i) \Big] \exp\Big(- \int_{0}^{t_k} \lambda^*(u)du \Big), \tag{4.12}$$

therefore,

$$\begin{aligned} L &= \Big[\prod_{i=1}^{k} f^*(t_i) \Big] \Big(1 - F^*(T) \Big) \\ &= \Big[\Big[\prod_{i=1}^{k} \lambda^*(t_i) \Big] \exp\Big(- \int_{0}^{t_k} \lambda^*(u)du \Big) \Big] \Big(\exp\Big(- \int_{t_k}^{T} \lambda^*(u)du \Big) \Big) \\ &= \Big[\prod_{i=1}^{k} \lambda^*(t_i) \Big] \exp\Big(- \Big[\int_{0}^{t_k} \lambda^*(u)du + \int_{t_k}^{T} \lambda^*(u)du \Big] \Big) \\ &= \Big[\prod_{i=1}^{k} \lambda^*(t_i) \Big] \exp\Big(- \Big[\int_{0}^{T} \lambda^*(u)du \Big] \Big). \end{aligned} \tag{4.13}$$

As a result, we can see that equation 4.10 is equivalent to

$$L = \Big[\prod_{i=1}^{k} \lambda^*(t_i) \Big] \exp\Big(- \int_{0}^{T} \lambda^*(u)du \Big). \tag{4.14}$$

Thus, we've accounted for the likelihood including the possibility of observing no arrivals in the time period $(t_k, T]$. This extends the likelihood function derived in equation 4.9.

4.3 Log-Likelihood for Exponential Decay

Again, using what I've learned from Statistical Inference, I know that the log-likelihood, l, of the function $f(t)$ is given by

$$l = \log\Big(\prod_{i=1}^{k} f^*(t_i)\Big) = \sum_{i=1}^{k} \log\Big(f^*(t_i)\Big). \tag{4.15}$$

Using what we know here, we can take the likelihood function from equation 4.9 and derive the log-likelihood function for the interval $[0, t_k]$ as follows

$$l = \log\Bigg(\Big[\prod_{i=1}^{k} \lambda^*(t_i)\Big] \exp\Big(-\int_0^{t_k} \lambda^*(u)du\Big)\Bigg), \tag{4.16}$$

which becomes

$$l = \sum_{i=1}^{k} \log\left(\lambda^*(t_i)\right) - \int_0^{t_k} \lambda^*(u)du = \sum_{i=1}^{k} \log\left(\lambda^*(t_i)\right) - \Lambda(t_k). \tag{4.17}$$

The last term, Λ, is defined to be the *compensator* and will be discussed further in my final report. Note that the *compensator*, being the integral over $[0, t_k]$, can be separated into the segments $[0, t_1], (t_1, t_2], (t_2, t_3], ..., (t_k, t_{k+1}]$, and therefore

$$\Lambda(t_k) = \int_0^{t_k} \lambda^*(u)du = \int_0^{t_1} \lambda^*(u)du + \sum_{i=1}^{k-1} \int_{t_i}^{t_{i+1}} \lambda^*(u)du. \tag{4.18}$$

We will now consider the case where $\lambda^*(u)$ decays exponentially, and I will simplify the above. I will sub in the standard form of the conditional intensity function with exponential kernel in place of $\lambda^*(u)$. Note in the below that the excitation function does not apply where we have no arrivals in the integral range from 0 to t_1. Then working through the integrals and rearranging we get

$$\begin{aligned}
\Lambda(t_k) &= \int_0^{t_1} \lambda du + \sum_{i=1}^{k-1} \int_{t_i}^{t_{i+1}} \lambda + \sum_{t_j<u} \alpha e^{-\beta(u-t_j)} du \\
\Lambda(t_k) &= \lambda t_1 + \sum_{i=1}^{k-1} \Big[\lambda u\Big]_{t_i}^{t_{i+1}} + \alpha \sum_{i=1}^{k-1} \int_{t_i}^{t_{i+1}} \sum_{j=1}^{i} e^{-\beta(u-t_j)} du \\
\Lambda(t_k) &= \lambda t_1 + \sum_{i=1}^{k-1} \Big[\lambda t_{i+1} - \lambda t_i\Big] + \alpha \sum_{i=1}^{k-1} \int_{t_i}^{t_{i+1}} \sum_{j=1}^{i} e^{-\beta(u-t_j)} du.
\end{aligned} \tag{4.19}$$

Now we can expand the second term to make explicit its telescoping series nature

$$\Lambda(t_k) = \lambda t_1 + \Big[(\lambda t_2 - \lambda t_1) + (\lambda t_3 - \lambda t_2) + ... + (\lambda t_k - \lambda t_{k-1})\Big] + \alpha \sum_{i=1}^{k-1} \int_{t_i}^{t_{i+1}} \sum_{j=1}^{i} e^{-\beta(u-t_j)} du$$

$$\implies \Lambda(t_k) = \lambda t_1 + \Big[- \lambda t_1 + \lambda t_k\Big] + \alpha \sum_{i=1}^{k-1} \int_{t_i}^{t_{i+1}} \sum_{j=1}^{i} e^{-\beta(u-t_j)} du$$

$$\implies \Lambda(t_k) = \lambda t_k + \alpha \sum_{i=1}^{k-1} \int_{t_i}^{t_{i+1}} \sum_{j=1}^{i} e^{-\beta(u-t_j)} du. \tag{4.20}$$

Rearranging and working through the integral we are left with

$$\begin{aligned}
\Lambda(t_k) &= \lambda t_k + \alpha \sum_{i=1}^{k-1} \sum_{j=1}^{i} \int_{t_i}^{t_{i+1}} e^{-\beta(u-t_j)} du \\
\implies \Lambda(t_k) &= \lambda t_k + \alpha \sum_{i=1}^{k-1} \sum_{j=1}^{i} -\frac{1}{\beta} \Big[e^{-\beta(u-t_j)}\Big]_{t_i}^{t_{i+1}} \\
\implies \Lambda(t_k) &= \lambda t_k + \alpha \sum_{i=1}^{k-1} \sum_{j=1}^{i} -\frac{1}{\beta} \Big[e^{-\beta(t_{i+1}-t_j)} - e^{-\beta(t_i-t_j)}\Big] \\
\implies \Lambda(t_k) &= \lambda t_k - \frac{\alpha}{\beta} \sum_{i=1}^{k-1} \sum_{j=1}^{i} \Big[e^{-\beta(t_{i+1}-t_j)} - e^{-\beta(t_i-t_j)}\Big].
\end{aligned} \tag{4.21}$$

If we swap the order of the summations we can keep j fixed and vary i to $k-1$ and factor out the $e^{-\beta(-t_j)}$ terms. This way, it is much clearer to see the inner sum telescope. We can then pass the term in the square brackets through the summation which will allow us to cancel some terms

$$\begin{aligned}
\Lambda(t_k) &= \lambda t_k - \frac{\alpha}{\beta} \sum_{i=1}^{k-1} \Big[e^{-\beta(t_k-t_i)} - e^{-\beta(t_i-t_i)}\Big] \\
&= \lambda t_k - \frac{\alpha}{\beta} \sum_{i=1}^{k-1} \Big[e^{-\beta(t_k-t_i)} - 1\Big].
\end{aligned} \tag{4.22}$$

Finally, the final summand is often extended to k, this step is redundant as it means adding 0 to the final result, but I will include this out of tradition and mathematical aesthetics

$$\Lambda(t_k) = \lambda t_k - \frac{\alpha}{\beta} \sum_{i=1}^{k} \Big[e^{-\beta(t_k-t_i)} - 1\Big]. \tag{4.23}$$

This is now our final expression for $\Lambda(t_k)$. Now we can take this, along with the standard form

of Hawkes process with exponential decay,

$$\lambda^* = \lambda + \alpha \sum_{t_i<t} e^{-\beta(t-t_i)}, \tag{4.24}$$

and substitute these into equation 4.17 to give us the log-likelihood function for a Hawkes process with exponential decay

$$l = \sum_{i=1}^{k} \log \left[\lambda + \alpha \sum_{j=1}^{i-1} e^{-\beta(t_i - t_j)} \right] - \lambda t_k + \frac{\alpha}{\beta} \sum_{i=1}^{k} \left[e^{-\beta(t_k - t_i)} - 1 \right]. \tag{4.25}$$

Computing a summation within a summation will lead to $\mathcal{O}(k^2)$ complexity, this is computationally infeasible if k is sufficiently large, which may limit the number of events one can compute with. However, we can take the inner summation separately, which allows l to be computed with $\mathcal{O}(k)$ complexity. For $i \in \{2, ..., k\}$, let $A(i) = \sum_{j=1}^{i-1} e^{-\beta(t_{i-1} - t_j)}$, so that

$$A(i) = e^{-\beta t_i + \beta t_{i-1}} \sum_{j=1}^{i-1} e^{-\beta t_{i-1} + \beta t_j} = e^{-\beta(t_i - t_{i-1})} \left(1 + \sum_{j=1}^{i-2} e^{-\beta(t_{i-1} - t_j)} \right) = e^{-\beta(t_i - t_{i-1})} (1 + A(i-1)). \tag{4.26}$$

Along with the base case of $A(1) = 0$, l can be written as

$$l = \sum_{i=1}^{k} \log \left[\lambda + \alpha A(i) \right] - \lambda t_k + \frac{\alpha}{\beta} \sum_{i=1}^{k} \left[e^{-\beta(t_k - t_i)} - 1 \right]. \tag{4.27}$$

Using equations 4.26 and 4.27, we can compute estimations for the parameters λ, α and β in the fashion described in Section 4.1. Again, this is done by minimising the negative log-likelihood function, which is equivalent to maximising the likelihood function.

Chapter 5

An Introduction to Parameter Estimation in R

In this chapter, I will be using equations 4.26 and 4.27 to gain an intuitive understanding of the specifics of parameter estimation by varying elements of the data and procedure used on a small scale. I began by configuring the log-likelihood along with the function $A(i)$ by hand for a small number of events.

5.1 Calculating Log-Likelihood Equations

The artificial data I will use consists of the following arrival times: $t = \{1, 2, 4, 5, 7, 10, 11\}$.

With event $k = 1$, the base case, $A(1) = 0$, this results in

$$l = \log(\lambda) - \lambda.$$

k = 2

$$l = \log(\lambda) + \log(\lambda + \alpha e^{-\beta}) - 2\lambda + \frac{\alpha}{\beta}[e^{-\beta} - 1].$$

The subsequent log-likelihoods calculated are included as an appendix in Section A.2.

5.2 Applying Log-Likelihood Equations in R

In order to have R calculate the parameter estimates, I first gave the programme the vector of data, my arrival times. Then I built a function environment to hold my log-likelihood function.

The arguments in this function environment are made up of two things; the parameters of the log-likelihood function (α, β, λ), and the data to be inspected (my vector of arrival times). The function would then return the negative of the log-likelihood function. This is the function I am tasked with minimising. To do this, I use the R function Non-Linear Minimisation (*nlm*); this function takes my log-likelihood function along with the arrival data and an initial guess for each of the three parameters. R will then search in 3-dimensional space for improvements to these parameters similar to how the Newton–Raphson method works. Improvements to the parameters result in a further minimisation in the negative log-likelihood function. The code has been configured such that the *nlm* function will then display the value of the negative log-likelihood function and the final estimates for the three parameters. This process is repeated for each of my data sets as the number of arrivals will result in a different log-likelihood function.

For my first case where I have $k = 1$, R calculated a minimum value of 1 for the function and an optimal value of 9.9×10^{-1} for λ. α and β were not included in the resulting log-likelihood function, and so, their initial guesses did not change, their omission makes sense as these two terms are in the excitation function which only comes into play after the first arrival, and in this case, we had no further arrivals.

In the $k = 2$ case, R calculated a minimum value of 2 with the following estimates 2.7×10^{-4}, 1.2×10^{3}, and 1.0 for α, β, and λ respectively.

With $k = 3$, R finds a minimum value of 3.9, with estimates $\hat{\alpha} = 7.4 \times 10^{-4}$, $\hat{\beta} = 4.1 \times 10^{3}$, $\hat{\lambda} = 7.5 \times 10^{-1}$.

The $k = 4$ case results in a minimum of 4, with estimates $\hat{\alpha} = 7.0 \times 10^{-4}$, $\hat{\beta} = 7.3 \times 10^{3}$, $\hat{\lambda} = 1.0$.

The $k = 5$ case results in a minimum of 6.7, with estimates $\hat{\alpha} = 1.0 \times 10^{-4}$, $\hat{\beta} = 3.6 \times 10^{4}$, $\hat{\lambda} = 7.1 \times 10^{-1}$.

The $k = 6$ case results in a minimum of 9.1, with estimates $\hat{\alpha} = 4.9 \times 10^{-4}$, $\hat{\beta} = 9.7 \times 10^{3}$, $\hat{\lambda} = 6.0 \times 10^{-1}$.

The $k = 7$ case results in a minimum of 10.2, with estimates $\hat{\alpha} = 5.2 \times 10^{-4}$, $\hat{\beta} = 8.5 \times 10^{3}$, $\hat{\lambda} = 6.4 \times 10^{-1}$.

To explore how the three parameters vary under different data, I took the k = 5 case and constructed data with a small inter-event time $t = \{1, 2, 3, 4, 5\}$. This yielded a minimum of 5 with estimates $\hat{\alpha} = 3.0 \times 10^{-4}$, $\hat{\beta} = 5.7 \times 10^{3}$, $\hat{\lambda} = 1.0$. Then, in the same way, I constructed data with a much larger inter-event time $t = \{5, 10, 15, 20, 25\}$. This yielded a minimum of 13.0 with estimates $\hat{\alpha} = 3.0 \times 10^{-5}$, $\hat{\beta} = 1.9 \times 10^{5}$, $\hat{\lambda} = 2.0 \times 10^{-1}$. As you can see, the value for α has

reduced by a factor of 10 as a result of this change. This is understandable as α is thought of as the parameter setting the strength of the interactions. β also changed significantly between the two tests, β is thought of as the parameter that controls the relaxation of the intensity as time passes after an arrival. The larger number for β in the second test indicates that this model predicts that the intensity drops off much faster when the inter-event times are much larger. We see a similar change with λ, as the model sees that it takes much longer for a new arrival to enter the system.

5.3 Discussion

As the *nlm* method is very similar to the Newton–Raphson root-finding algorithm, changes to the initial guesses can lead to very different final results. I found this to be especially true in the cases I coded above, I suspect this is because the data I used is very small and may result in many local minima where the *nlm* algorithm would settle. If the data were bigger, these inconsistencies could be smoothed out, and the algorithm could more easily find a global minimum.

Another item to keep in mind when applying this process to real-world data is the constraint that arrivals cannot occur at the same time. If your data includes multiple arrivals in the same time interval, you may think it would be appropriate to split these arrivals evenly between this interval (for example if five arrivals occur in a one-second interval, to separate these arrivals by 0.2 seconds each). However, this method may result in the optimisation function noticing a pattern of arrivals after 0.2 seconds intervals, and thus, the parameters would reflect that, and try to replicate this.

Another point I am aware of is that the data sets of arrival times I have worked with are very artificial. An ideal test at this point would be to simulate a Hawkes process, which would return a vector of arrival times. I could then use these arrival times to estimate values for α, β and λ in a similar way to how I've done this before. The test would be to examine if the parameter estimation procedure could estimate the true parameter values accurately; I will approach this very task in Section 6.2 and Section 6.3 in the next chapter. However, the problems I've had with this method is that the vector of arrival times is given in continuous time, and would require much more work to compute the log-likelihood and $A(i)$ functions. More importantly, given such a small number of arrival times, the accuracy of these estimates would be very unstable. It is clear that if I am to explore parameter estimation further, I will need a way of computing equations 4.26 and 4.27 in R for a given data set, as opposed to working this out by hand. I will introduce a method for completing this task in the next chapter.

Chapter 6

Simulation and Estimation

In the previous chapters, I have tackled Hawkes processes (which I will sometimes call HPs, or a HP) from a theoretical basis. Now that we have an intuitive understanding of Hawkes processes and a grounding in the mathematical concepts that underpin this process, we can apply this learning to real-world data. As mentioned, to do this, I will be using functions in R to calculate equations 4.26 and 4.27. A great resource for this has been "MLE of Hawkes' Self-Exciting Point Process" by Radha Krishna [17]. This paper gives a short introduction to Hawkes processes and includes R functions for use in simulating and estimating Hawkes process realisations. This code performs parameter estimation in a similar way to what I have done in Chapter 5; however, most importantly, this code automates the calculation of equations 4.26 and 4.27. Using this code, I can apply the *nlm* function directly to the log-likelihood function laid out in this code to calculate parameter estimates without having to calculate the terms of the log-likelihood function individually.

6.1 How This Code Works

The log-likelihood function from this paper can be understood from looking at equation 4.27. I have included a screenshot of the function below where we can see that it starts by setting out the parameters α_i, β_i and λ_i. The function then calculates the terms of the log-likelihood function separately as three terms, as explained in Fig. 6.2. This function, of course, also calculates $A(i)$ from equation 4.26 and then includes it in "term_3". The function then returns the negative of the three calculated terms to make up the negative log-likelihood function which I can then apply the *nlm* function to in order to find the optimal parameter estimates.

```r
# Log Likelihood Function
loglik <- function(params, arrivals){
  alpha_i <- params[1]
  beta_i <- params[2]
  lambda_i <- params[3]
  term_1 <- -lambda_i*arrivals[n]
  term_2 <- sum(alpha_i/beta_i*(exp( -beta_i * (arrivals[n] - arrivals)) - 1))
  Ai <- c(0, sapply(2:n, function(z) {
    sum(exp( -beta_i * (arrivals[z]- arrivals[1:(z - 1)])))
  }))
  term_3 <- sum(log( lambda_i + alpha_i * Ai))
  return(-term_1- term_2 -term_3)
}
```

Figure 6.1: R Code taken from [17] used to calculate equations 4.26 and 4.27.

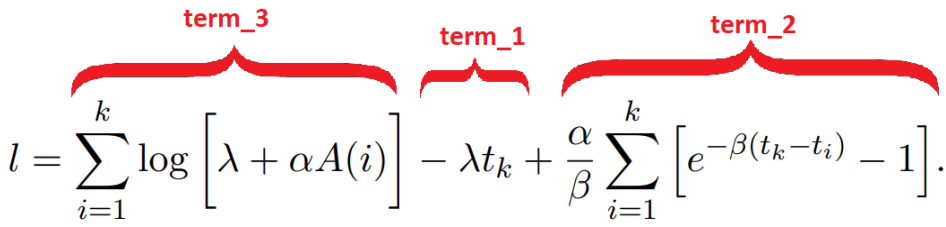


Figure 6.2: Figure showing how the code in Fig. 6.1 calculates equation 4.27.

Krishna [17] also included a very useful function that can be used to simulate a Hawkes process realisation for a given set of α, β, λ parameters and the desired number of arrivals n. This function will generate a vector of arrival times in line with a Hawkes process with exponential kernel subject to certain checks to ensure the process doesn't explode. This is done by ensuring the difference in time between the theoretical previous arrival and the theoretical next arrival is greater than e^{-6}. The function will then add the value in between these two theoretical arrivals as the next arrival and add that to the vector of arrival times, and repeat this process until the number of arrivals calculated is equal to the desired number of arrivals n specified when calling the function. The random component in this process comes from the random uniform number called to be used in the calculation of the next arrival time. This random component can be set such that the results can be reproduced by adding the *set.seed* function, which I have removed in favour of a different approach which I will discuss in Section 8.1.1.

```r
# HP Simulation Function
simulate_hawkes <- function(alpha, beta, lambda, n){
  arrivals <- numeric()
  eps <- 1e-6
  # recurrence function
  S <- function(k){
    if(k==1) return(1)
    return( 1 + S(k-1)* exp(-beta * (arrivals[k]- arrivals[k-1])) )
  }
  # the function to be solved to obtain, u
  fu <- function(u, U, k){
    return(log(U) + lambda*(u - arrivals[k]) +
             alpha/beta * S(k) * (1 - exp(-beta * (u - arrivals[k]))))
  }
  fu_prime <- function(u, U , k){
    return( lambda + alpha * S(k) * (exp(-beta * (u - arrivals[k]))))
  }
  # iterative procedure to solve f(u)
  solve_u <- function(U,k) {
    u_prev <- arrivals[k] - log(U)/lambda
    u_next <- u_prev - fu(u_prev, U, k)/ fu_prime(u_prev, U, k)
    while( abs(u_next - u_prev) > eps){
      u_prev <- u_next
      u_next <- u_prev - fu(u_prev, U, k)/fu_prime(u_prev, U, k)
    }
    return(0.5 * (u_prev + u_next))
  }
  #set.seed(1)
  t1 <- -log(runif(1))/lambda
  arrivals <- c(arrivals, t1)
  k <- length(arrivals)
  while(k < n) {
    U <- runif(1)
    t_next <- solve_u( U, k)
    arrivals <- c(arrivals, t_next)
    k <- length(arrivals)
  }
  return(arrivals)
}
```

Figure 6.3: R Code taken from [17] used to create a Hawkes process simulation.

6.2 Simulation (n = 500)

To introduce these methods, I will start by simulating a Hawkes process with 500 arrivals using some reasonable parameters of $\alpha = 4$, $\beta = 5$ and $\lambda = 0.5$. Plotting these arrivals gives the left graph in Fig. 6.4 below.

I can then test how well the parameter estimation process works by applying the *nlm* function to the log-likelihood function with this vector of arrival times to estimate the known true parameters of this Hawkes process realisation. Doing this has given me estimates of $\hat{\alpha} = 4.07$, $\hat{\beta} = 5.51$, $\hat{\lambda} = 0.57$. These estimates are very close to the actual values of the Hawkes process.

I then decided to resimulate a Hawkes process of 500 arrivals using the estimated parameters as my α, β, λ parameters. If I then plot this resimulation on the same graph as the original simulation, I get the right graph in Fig. 6.4 below.

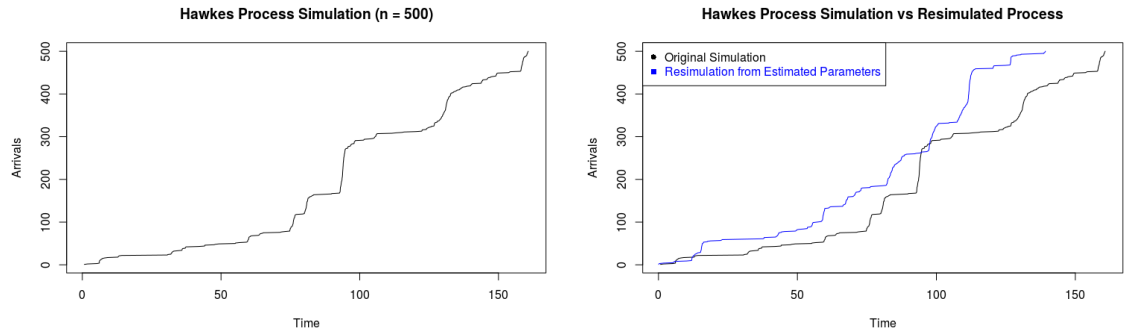


Figure 6.4: Left: HP simulation (n = 500). Right: Resimulated HP with original simulation.

We can see that the simulations follow a similar path and the overall realisations of both Hawkes processes don't differ by a large amount at any time point, in fact, they cross paths a number of times.

Using the $A(i)$ term in the simulation function, I can plot the intensity of this process over time. If I plot this on top of the original simulation, we can see that spikes in the intensity correspond with rapid increases in the counting process as we would expect from the explanation given in Section 2.1. It is also clear that the intensity drops back down (even as low as the background intensity λ) at times when there are much fewer arrivals in the process.

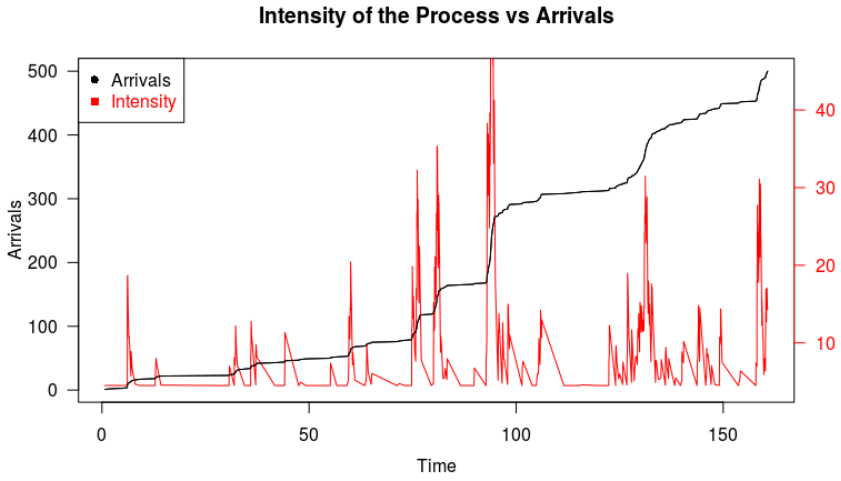


Figure 6.5: Hawkes Process Simulation with Intensity (n=500).

6.3 Simulation (n = 1000)

In order to investigate how this process can change, I performed the same process again, this time with $n = 1000$ arrivals and parameter values of $\alpha = 3$, $\beta = 4$ and $\lambda = 1$. Plotting this realisation gave the left graph in Fig. 6.6 below.

In the same way as before, I estimated the parameters of this process and the parameter estimates were $\hat{\alpha} = 2.85$, $\hat{\beta} = 3.98$, $\hat{\lambda} = 0.99$. These estimates are much closer to the true values than they were in the $n = 500$ case. Again, I resimulated a Hawkes process using the parameter estimates, plotting this resimulation on the same graph as the original gives the right graph in Fig. 6.6 below.

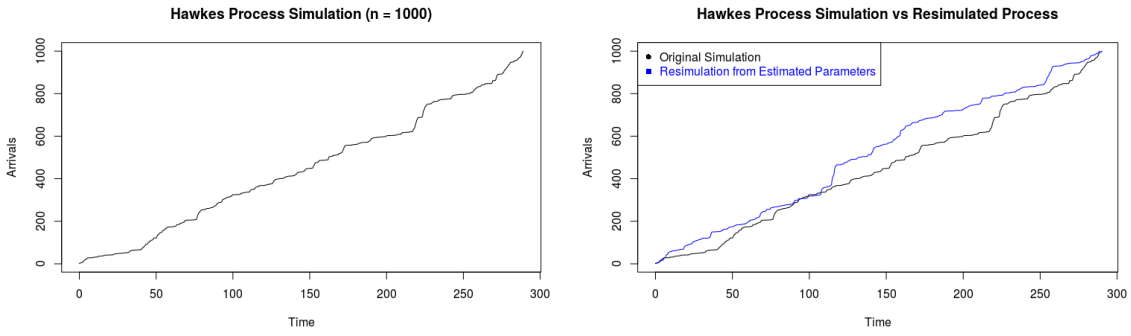


Figure 6.6: Left: HP simulation (n = 1000). Right: Resimulated HP with original simulation.

As you can see, these two simulations follow even closer to one another than in the $n = 500$ case, and the two processes converge on almost the exact same point. Out of interest, I plotted

the intensity of the process over the simulation, which yielded this graph.

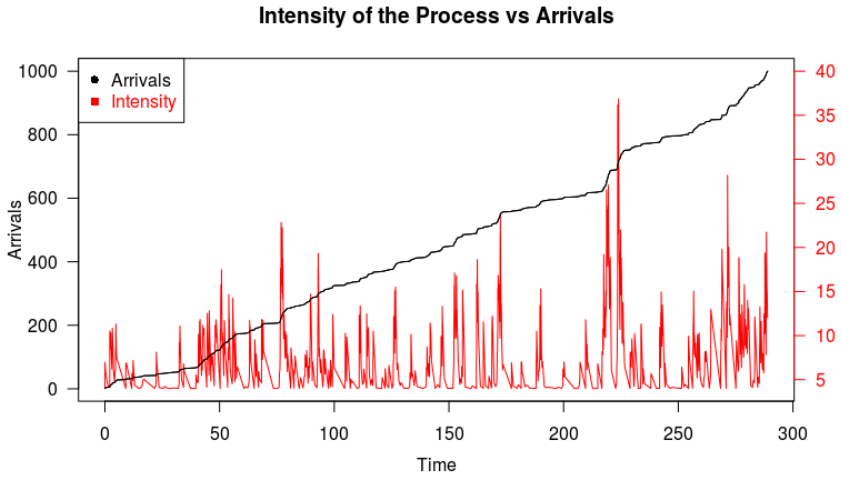


Figure 6.7: Hawkes Process Simulation with Intensity (n=1000).

You may be wondering if it was the choice of parameters α, β, λ, or the n value that has resulted in the increased accuracy in this test, however, I have tested this with multiple choices of parameters and found similar results. I will explore how the choice of n affects the accuracy of parameter estimates in the next chapter.

Chapter 7

Accuracy of Estimation and Predictability of Cascade Size

7.1 Accuracy of Estimation

To investigate how the number of arrivals affects the accuracy of the parameter estimates I created a function that will simulate a Hawkes process (for a given set of parameters), then estimate the parameters of this process. The function will then take the absolute value of the estimate minus the original parameter, then divide this number by the original value for each parameter α, β, λ. This gives the error in the parameter estimate, the function then adds the three parameters and divides by three to find the average parameter error. I then put this function in a *for loop* allowing me to vary the number of arrivals across 37 n values ranging from 10 arrivals to 2500 arrivals. I first ran this for parameters $\alpha = 2.25$, $\beta = 3$ and $\lambda = 1$. This gave the following graph.

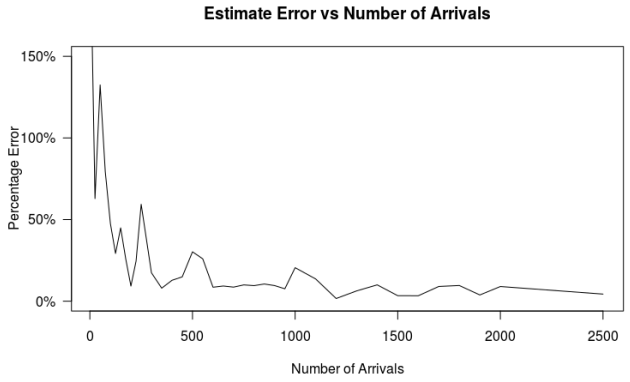


Figure 7.1: Average Error of Parameter Estimates across n values.

I then decided I should run this for different parameter values to avoid the bias in accuracy caused by not varying these values. I ran this for eight different sets of α, β, λ parameters and plotted them all together:

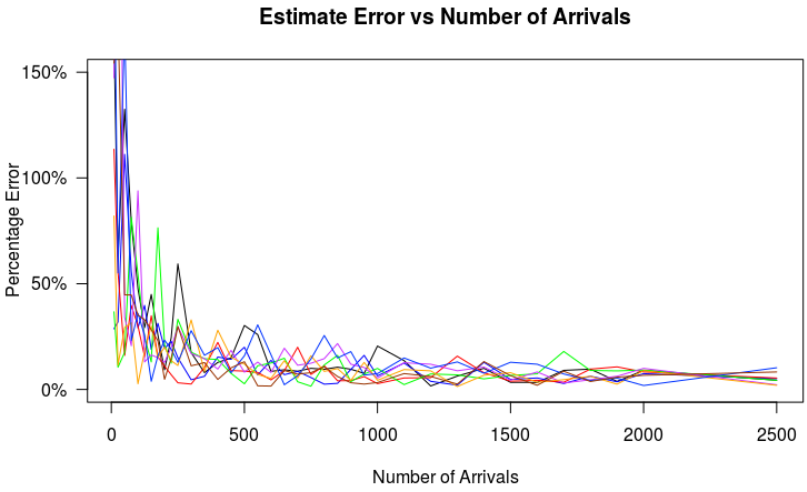


Figure 7.2: Average Error of Parameter Estimates across n values (varying parameters).

If I then take each vector of error values and find the average error at each n value, I can then plot the average parameter estimate error across n values, while varying parameters.

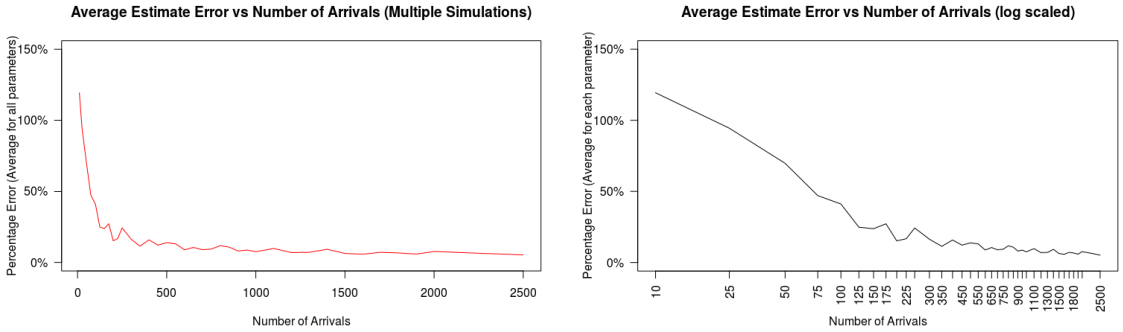


Figure 7.3: Left: Average Parameter Estimate Error across n values. Right: Same Data, log scaled.

From looking at these graphs, I can make some conclusions about how I should approach parameter estimation as I continue with my research. When the number of arrivals is $n = 25$, the average parameter estimate will be wrong by approx. 94.4%. When the number of arrivals is $n = 100$, this error drops to approx. 41.1%. When I estimate parameters with $n = 500$, I can expect my estimates to be wrong by about 13.9%. But I can achieve an average parameter estimate error of 7.5% when I work with $n = 1000$ arrivals, which is much more acceptable.

Therefore, based on this test, I will conclude that I can make confident statements about my findings when I work with over 1000 data points.

7.1.1 Accuracy of Estimation (varying parameters)

I then wanted to investigate how the accuracy of estimates could change when I vary my parameters α, β, λ by an order of magnitude. I will start with the parameter values used in Fig. 7.1 of $\alpha = 2.25$, $\beta = 3$ and $\lambda = 1$. I will vary each parameter down by an order of magnitude individually, and then all together. In the below graph, the **black** line is the left graph in Fig. 7.3, the blue line is the average parameter estimate error using the above parameter values, but with $\alpha = 0.225$ instead of $\alpha = 2.25$. In a similar way, the red line is using the Fig. 7.3 parameters, but with $\beta = 0.3$. The green line varies λ such that $\lambda = 0.1$. And finally, the orange line varies all of the parameters down by an order of magnitude.

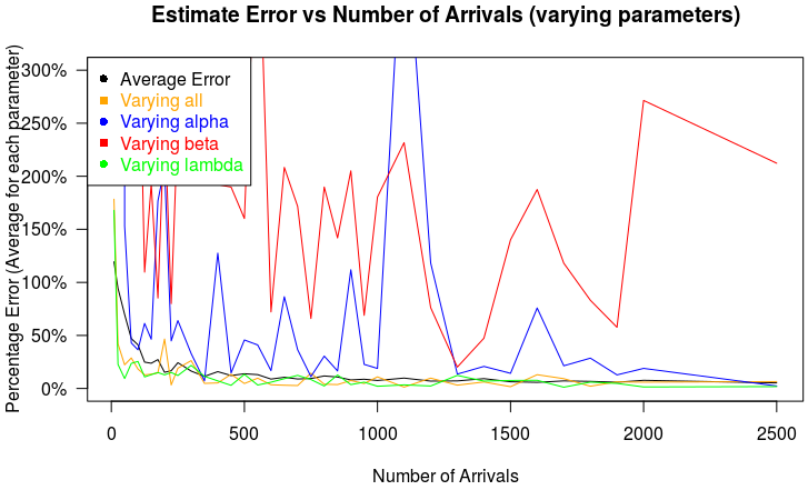


Figure 7.4: Average Error of Parameter Estimates across n values (varying parameters down by an order of magnitude).

As we can see, varying lambda, the background intensity, gave very similar results to the average error line, as did the case varying all parameters down by an order of magnitude. However, the most interesting point to note in this graph is the extremely varied parameter estimate errors when we vary α and β independently. The interaction between α and β has been shown to be extremely important in the HP with exponential kernel as we've seen in The Branching Process in Section 3.2.

To see if this variation is due to the *interaction* between α and β, or just because their respective interaction with λ, I will vary both α and β down by an order of magnitude together to see how that affects the estimated parameter error, this is illustrated in the graph below.

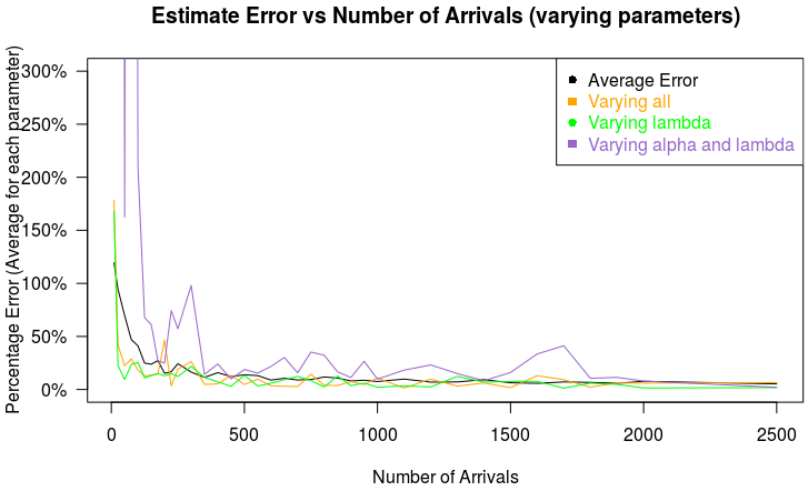


Figure 7.5: Average Error of Parameter Estimates across n values (varying parameters down by an order of magnitude).

It's clear that a huge amount of the error at higher n values has been reduced by restricting α and β to a ratio near 1 ($0.225/0.3 = 0.75$). This is an interesting point to note; we know that if the branching ratio is > 1, the process becomes *supercritical* and is very unstable (Section 3.2). Initially, I had thought that this means that a branching ratio near zero had caused a different type of instability in the process, resulting in reduced accuracy of parameter estimates, and therefore, reduced certainty in the resulting conclusions made. This isn't something that I've seen in the literature around Hawkes processes.

To test this, I then allowed the parameters to vary up by an order of magnitude. This reduced the size of the error where the branching ratio was near zero (see Fig. 7.6). (Note: I did not include the case where I vary α up by an order of magnitude as the supercritical nature of this process was much more extreme than in the previous case of varying the parameters down by an order of magnitude.) When looking at the red line, varying beta up by an order of magnitude, we see that the instability in estimates is reduced somewhat. This seems to show that a branching ratio near zero ($2.25/30 = 0.066\dot{6}$) is only a problem when α is close to zero too.

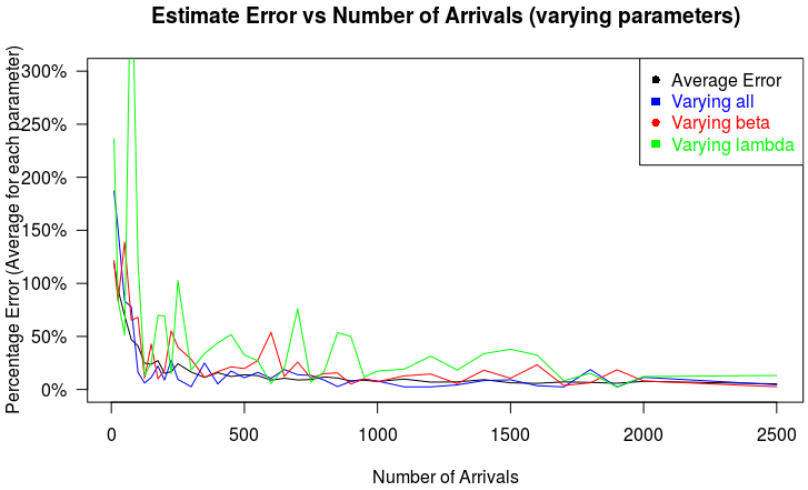


Figure 7.6: Average Error of Parameter Estimates across n values (varying parameters down by an order of magnitude).

Now that we have a better idea on what to look out for when estimating parameters, we can use this to predict future cascades by resimulating the process through future events. Alternatively, we can also use these parameter estimates in a different method to predict future cascades. I would like to briefly explore this method of predicting future cascade sizes using the parameters that we can now confidently estimate. I will introduce and explore this method in the next section.

7.2 Predictability of Cascade Size

An interesting question to ask is how one could predict the size of the counting process at a given point in the future. Let's say we observe a given counting process from a certain point (call this time $\tau = 0$) up to the most recent arrival (call this time T). We would call this time interval the *observation window*. One could then ask, given the distribution and number of arrivals seen in the observation window, how many arrivals could we expect to see at a certain time in the future?

A method that finds a solution to this question is laid out in O'Brien *et al.* [18], and I will describe the basic concept behind this method here.

In the observation window $[0, T]$, we say we've observed n arrivals at times $\{\tau_0, \tau_1, \tau_2, ..., \tau_{n-1}\}$. We will consider how this process evolves up to a certain time in the future, and call this future time Ω. We will call the time interval $(T, \Omega]$ the *prediction window*. The formula developed to predict this cascade size is

$$m_I(r) = n + \frac{\lambda_0}{1-\xi}\left[r + \frac{\xi}{\beta(1-\xi)}\left(e^{-\beta(1-\xi)r} - 1\right)\right] + \sum_{i=1}^{n-1} \frac{\xi e^{-\beta a_i}}{1-\xi}\left[1 - e^{-\beta(1-\xi)r}\right]. \tag{7.1}$$

Again, n is the number of observations in the prediction window, λ_0 and β are the parameters estimated, and $\xi = \alpha/\beta$, where α will also be an estimated parameter. $a_i = T - \tau_i$ for each arrival time τ_i in the observation window. $r = \Omega - T$, the length of the prediction window.

In order to calculate the predicted cascade size using this formula, I built a function in R that will calculate each part of this formula and put it together to find the value for $m_I(r)$. This function is shown below in Fig. 7.7.

It is at this point that I can test the accuracy of this function. In the same way that I've done in Section 6.2, I will simulate a HP realisation with $n = 500$. I will allow the first 100 elements of this arrival vector to be the observation window. The 500^{th} element of this vector is the time of the 500^{th} arrival; I will set this value to be Ω. When I apply these parameters to my predicted cascade size (PCS) function, it predicts a cascade size of 472.412. I was very impressed with this result as the function had only 100 data points to work with and still managed to predict the next 400 data points with great accuracy. A plot of what this prediction looks like is shown in the left plot in Fig. 7.8.

I then repeated this test for a HP simulation of $n = 1000$. I set the first 500 arrivals as the observation window, and predicted the cascade size at the time of the actual 1000^{th} arrival. This

```
# Predicted Cascade Size = PCS
PCS <- function(lambda,beta,alpha,T,tao,omega){
  # create required variables
  xi=alpha/beta
  r=omega-T
  # create the "a" vector
  a=c()
  n=length(tao)
  for (i in 1:n){
    a[i]=T-tao[i]
  }
  sum=0
  m=0
  # loop to find the second part of the function
  for (i in 1:(n-1)){
    newsum=(xi*exp(-beta*a[i]))/(1-xi)*(1-exp(-beta*(1-xi)*r))
    sum=sum+newsum
  }
  # add the second part of the function to the rest
  m = n + (lambda/(1-xi))*(r+xi/(beta*(1-xi))*(exp(-beta*(1-xi)*r)-1)) + sum
  return(m)
}
```

Figure 7.7: Function built to find the predicted cascade size of a HP.

gave me a PCS of 1013.94 arrivals, which is extremely accurate given the prediction window was the same length as the observation window. A plot of what this prediction looks like is shown in the right plot in Fig. 7.8.

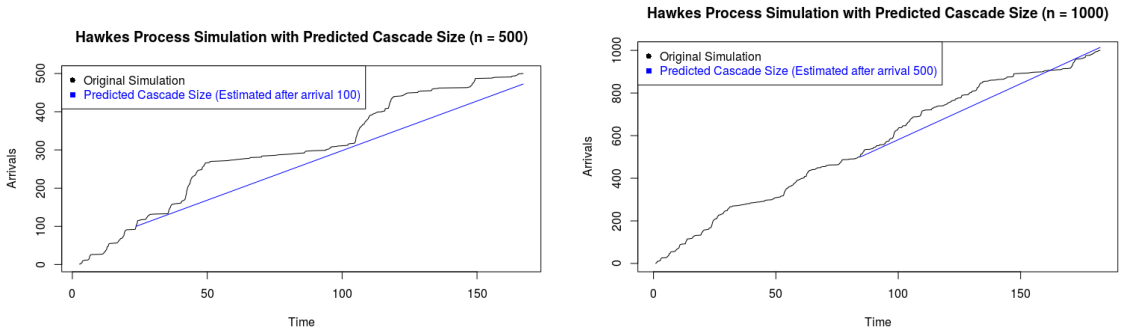
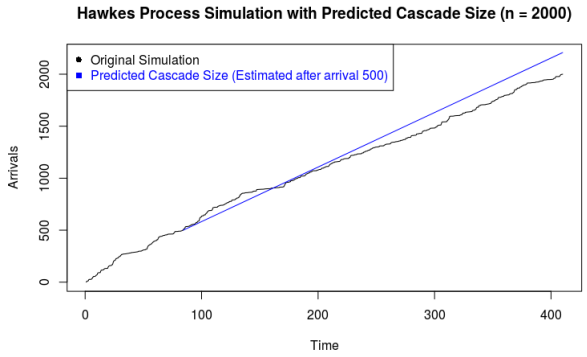


Figure 7.8: Left: PCS of an $n = 500$ HP simulation. Right: PCS of an $n = 1000$ HP simulation.

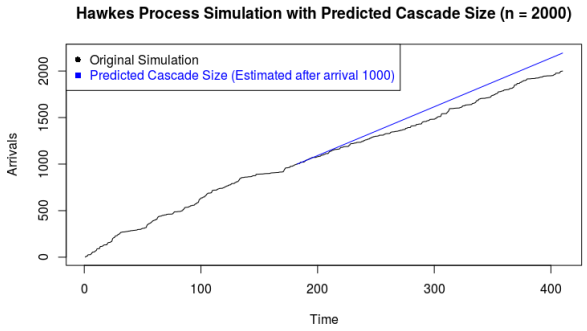
To further test the accuracy of this method, I will run a simulation of $n = 2000$ and predict the final cascade size using three different observation windows. One observation window will be $[0, 500]$, another will be $[0, 1000]$, and the final observation window will be $[0, 1500]$.

In the first case, an observation window of $[0, 500]$ (1/4 of the full cascade) yielded a PCS of 2208.175, which is wrong by 10.4%. Observation window $[0, 1000]$ (1/2 of the full cascade) gave a PCS of 2192.629, this is wrong by 9.6%, this isn't as much of an improvement over the $[0, 500]$ case as I was expecting. Finally, an observation window of $[0, 1500]$ (3/4 of the full cascade) yielded a PCS of 2073.75, which is a sizable improvement over the first two cases, having an error of only 3.7%. This test, along with other results from this chapter, seems to suggest that

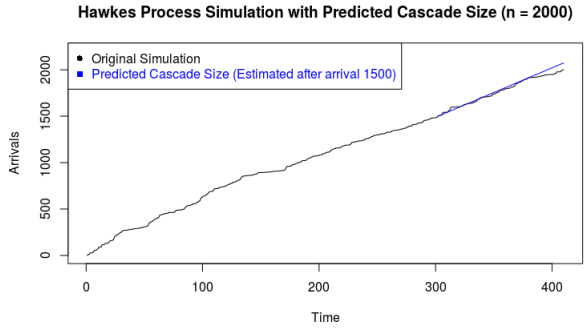
short prediction windows with large observation windows can yield very accurate results, but the error in cases with larger prediction windows does not increase linearly. The plots illustrating these predictions are included below.



(a)



(b)



(c)

Figure 7.9: PCS of an $n = 2000$ HP simulation with observation windows (a) $[0, 500]$, (b) $[0, 1000]$ and (c) $[0, 1500]$.

7.2.1 Discussion

One very important point to note with this method of predicting the future of a HP cascade is that the prediction is linear, so this method doesn't display the natural clustering of arrivals present in all HPs. So, in the case of its application to finance, this wouldn't be the best method for predicting fluctuations in the market far into the future, however, it could be a very useful method for short term micro-market movements. I will look into this concept more in Section 8.2.

Chapter 8

Practical Applications of Hawkes Processes

8.1 Hawkes Processes Application to Internet Virality

As mentioned in Section 2.4, the virality of trends can follow a self-exciting point process and bear all the same characteristics as a Hawkes process. One good example of online trends following this type of process is instances of retweets on twitter.

In my interim report, I stated that I would like to try and replicate the results in Rizoiu *et al.* [11] as a proof of concept. In this paper, Rizoiu *et al.* applied Hawkes processes to social media data, more specifically, to predict the popularity of a tweet quoting the New York Times as measured by retweets. This set of retweets of an initial tweet is defined as the retweet cascade. This application will tie together many of the concepts I have discussed, including the branching process structure introduced in Section 3.2. This retweet cascade is made up of only one immigrant event, the original tweet, and all of its offspring, the subsequent retweets. This offspring is made up of child events coming directly from the original immigrant event, as in, somebody retweeting the original tweet they saw from following the original author, and also offspring from subsequent generations where someone may be the recipient of this retweet and also decide to retweet.

Figure 8.1: Tweet Announcing the Death of "Mr. Spock".

8.1.1 Replication Procedure

I would like to replicate the results from this paper's parameter estimation. My method differs from the method used in this paper, in that, the process was modelled as a *marked* Hawkes process. This mark or magnitude is a representation of the user's influence, as measured in followers, the more followers a person has, more people will potentially see their retweet. Therefore, I don't expect the parameters I estimate to match the parameters estimated in [11]. However, I would hope to be able to replicate the process itself. I would hope that using a different method would add to the robustness of the results obtained.

I started by attempting to replicate the process obtained by counting the first 600 seconds of retweet instances. Taking each instance of a retweet as an element in the vector of arrival times, I was able to estimate the parameters for this process as $\alpha = 0.03$, $\beta = 0.07$ and $\lambda = 0.04$. These estimates, of course, differ from the parameters found in [11], being $\alpha = 0.0003$, $\beta = 1.0156$ and $\lambda = 0.04$. This result is expected due to the different methods being implemented. I then turned to replicating the realisation of the process. Using these parameters I've estimated, I can then simulate a HP for the same number of arrivals as the original 600 seconds of retweets, this number of arrivals being $n = 43$. I then plotted this realisation on the same graph as the true realisation here.

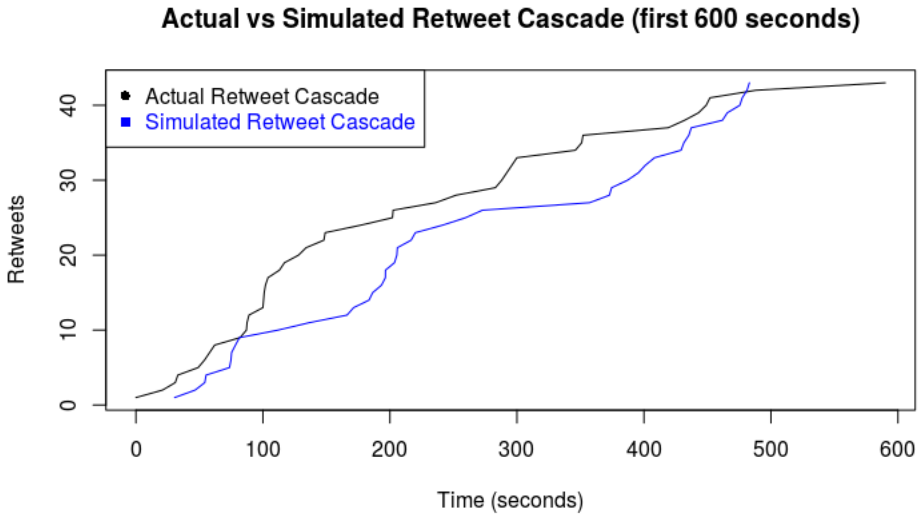


Figure 8.2: Replication of Retweets Events (600 seconds).

As we've since learnt in Chapter 7, when working with data containing $n = 100$ arrivals, we can expect an average error in our estimated parameters of about 41.1%. As we are only working with 43 arrivals, the simulation using our (probably) poorly estimated parameters is unlikely the be an accurate replication of the actual retweet cascade.

As a result, I decided to return to the data source and attempt to replicate the same results, however, using a larger version of the same data set. I didn't want to use the full data set, as this would include a number of retweet instances that have occurred months or even years apart. The problem with this is that these models don't account for the fact that, like all things, this tweet has fallen into the eternal abyss of the internet. Therefore, the sample of data points I've decided to use is the first hour of retweet instances. This data set contains 160 data points, and when plotted, gives the following graph.

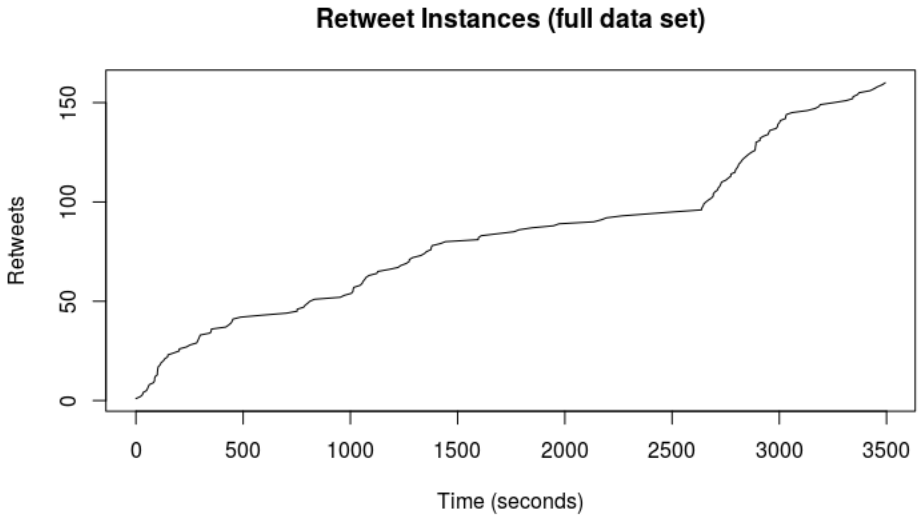


Figure 8.3: Actual Retweets Events (1 hour).

When I attempted to estimate the parameters of this process, I encountered a new problem with parameter estimation. The vector of arrival times looks like $\{0, 21, 31, 33, 49, 54, ..., 3480, 3494\}$, and my model has a hard time dealing with arrival times that are as spread out as these are. To resolve this issue, I was able to divide this vector of arrival times by an appropriate constant (in this case, 100), which made it much easier for my algorithm to estimate the parameters of the process. I then estimated the parameters from my new vector of arrival times. After resimulating the process using these parameters, I could then multiply these simulated arrival times by 100 to bring me back to a similar vector of arrival times that I began with. I also noticed another problem here; when I removed the $set.seed(1)$ function from the simulation algorithm, thus adding an extra random component into the algorithm, I would often obtain varying resimulated realisations. This means that each time I ran the resimulation process, I would be given a slightly different realisation each time. To find a better way of resimulating, I created a function that takes the parameters α, β, λ and n, then produces ten simulations. Then it runs each simulated arrivals vector through a $for\ loop$ which will take each element of the ten vectors one-by-one and find the average simulation. When this was done, I plotted the final average simulation on the same plot as the original vector of arrival retweets to obtain the

graph below.

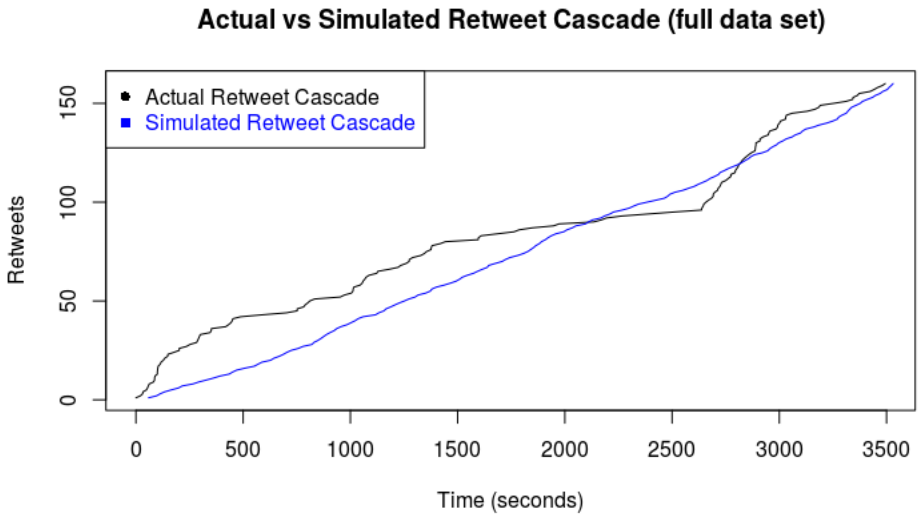


Figure 8.4: Replication of Retweets Events (1 hour).

As we can see from the graph, this process has been very successful in replicating the actual retweet cascade.

8.1.2 Discussion

In a similar way to the predicted cascade size method in Section 7.2, finding the average of all simulations takes a lot of the non-linear nature of the HP out of the final graph. This is because finding the average of all simulations, each with various clusters of arrivals, can smooth out the final average simulation. This being such that, the final average simulation has smoothed out these clusterings to an almost linear realisation. This method tends to give a process that converges to the same endpoint as the original, as the parameters take the entirety of the original process into account. This method of finding the average of multiple simulations could be very useful as it still keeps some of the non-linear nature of the HP, and is very robust in converging to an accurate terminal point in the process.

Another point to mention is that the arrivals are in seconds, therefore in numerical time format starting at zero. Our data won't always be as conveniently formatted. In future chapters, we will encounter data with much different starting points and increment values; I will explain the data cleansing procedure required for this in Section 8.2.1.

8.2 Hawkes Processes Application to Crime Data

As I've discussed in Section 2.4, the distribution of crime data can follow a self-exciting point process and bear all the same characteristics as a Hawkes process. Highly clustered event sequences in crime data are pervasive in police-reported figures. The reasons for this could come down to crime-specific patterns of criminal behaviour or due to the fact that an increased sense of lawlessness in an area can lead to further encouragement of antisocial behaviour. In the case of burglary, it has been shown that after there has been a burglary at a residence, there is an increased risk of burglary for both the burgled house and neighbouring houses (Short *et al.* [19]).

Luckily for my research, there are many well maintained and accurate databases for crime data that are publicly available. I will be working with crime data from two U.S. cities; *Baltimore*, Maryland [20] and *Kansas City*, Missouri [21].

8.2.1 Baltimore Crime Data

I will start with Baltimore crime data, this data includes all police-reported crimes for the city of Baltimore for 11 days, from 2020-01-22 06:30:00 UTC to 2020-02-01 23:45:59 UTC, and it is here that I encountered my first problem with this data. This data is stored in date format in one column, and the times these crimes were reported are in the next column. Through manipulation of this data, I was able to format these columns such that *Excel* could recognise the date and time parameters. Then multiplying these recognised date cells by 86400 will convert these dates into *Unix Time* format, that being, the number of seconds passed since 01/01/1970.

Now that my time-series data is in a format that my algorithms can work with, there still remained two more issues, one of which I faced in the previous section. By using this time format, it means that my algorithm doesn't recognise that an event has occurred for the first 1579682700 seconds. This large gap would mean that my parameter estimation algorithm is searching for a λ value low enough to model a process that wouldn't encounter an arrival for most of the life of the process, and then see 999 arrivals over the next 950400 seconds (11 days). This simply meant that the parameter estimation algorithm gave seemingly arbitrary parameters that had no tangible use, and sometimes gave no answer at all. Along with this problem, this time format also meant that many events were thousands of seconds apart; this caused a very similar problem for the parameter estimation algorithm. To remedy both of these issues, I needed to subtract a suitable constant from each arrival time, then divide each arrival time by another suitable constant. In order to produce usable results, I required a suitable combination of these two constants that would translate the arrival times into a different series that could be worked with.

I will first show a plot of the Baltimore crime counting process below.

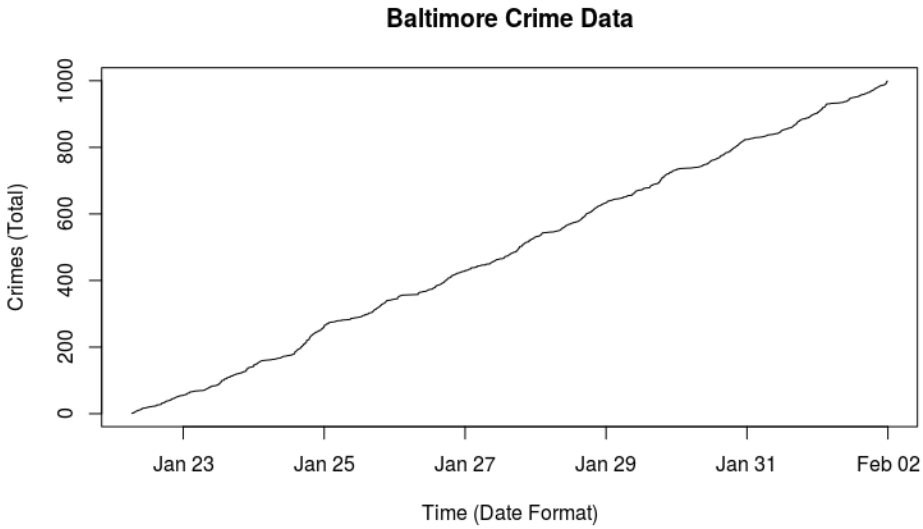


Figure 8.5: Baltimore Crime Data.

Then implementing the parameter estimation procedure and resimulating the process from the beginning, using my multiple simulations method, R outputs a series of arrival times which I can then manipulate back to the format I started with. I do this by multiplying the simulated arrival vector by the same constant the actual arrival vector was multiplied by, then also adding the constant from before. Using the $as.POSIXct$ R function, I can turn this vector of arrival seconds back into dates and times. When I plot this on the same graph as the original, the results match almost exactly. This graph is shown in Fig. 8.6.

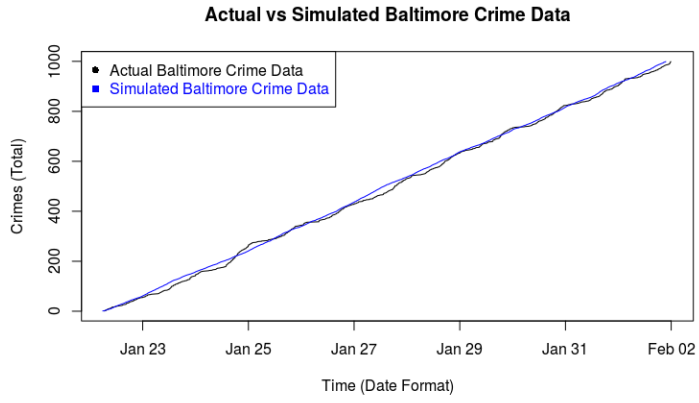


Figure 8.6: Baltimore Crime Data Resimulation.

For me, the most impressive result from this test is that, despite heavy manipulation of the data, both prior and post estimation and simulation, there was no loss of accuracy in the results, as evidenced by the above graph.

Using this type of method, one can predict the future clusterings of crimes in local areas. I believe the best use for an algorithm like this could be as a constant process updating on the fly. If a given cluster starts to emerge, this algorithm could predict, based on the history of crime data, if a larger cluster will appear in the near future or if we are seeing the end of a small cluster of crimes. With this type of analysis at hand, police departments could more efficiently allocate officers to at-risk areas, and reduce the risk of a series of crimes turning into a larger cluster.

It is this idea that I will now test. I will implement the predicted cascade size method laid out in Chapter 8 on a smaller subset of the data. I will find a point in the data that resembles the beginning of a cluster, the entire data set, up to this point, will be the observation window. I want to see if this method can predict the cascade size at the end of this cluster accurately, such that this method could be proven useful for police departments. The left graph in Fig. 8.7 below shows the experiment outline. At the beginning of this cluster, at 18:01 on Friday, there were 220 arrivals. I will predict as far out as 06:00 on Saturday, at this time, the actual number of crimes was at 280. (Note: this time format is in UTC. Therefore, in local time, the prediction window is 2 pm to 2 am, which would be a logical time window for the analysis of criminal activity.) Impressively, the predicted cascade size function estimated that at 06:00 on Saturday (UTC) there would be 280.0757 arrivals. This estimation is extremely accurate and could prove to be very useful in its practical application. The realisation of this prediction is shown in Fig. 8.7 below (right graph).

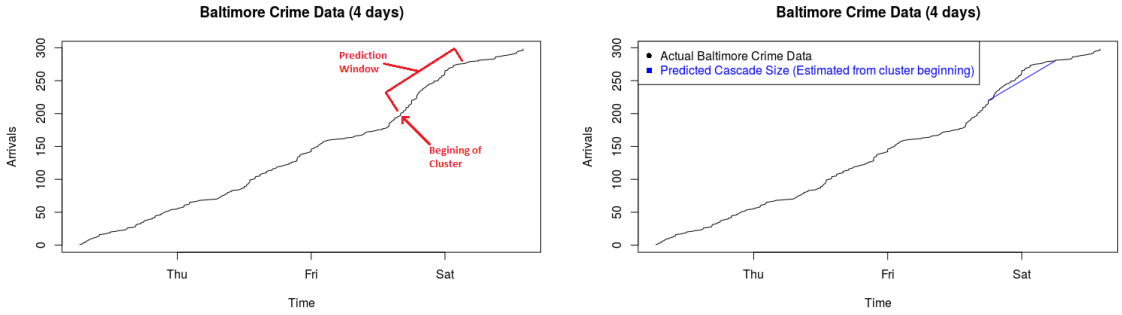


Figure 8.7: Baltimore Crime Data - Prediction Outline and Prediction.

8.2.2 Kansas City Crime Data

I took the same approach when investigating Kansas City crime data. When looking at the distribution of crimes in this city, it was much more clustered. Because of this, I wanted to see if my estimation and simulation algorithm could model this distribution accurately. I again used my *multiple simulations algorithm* to replicate these results. The crime data is plotted below, along with the replication realisation.

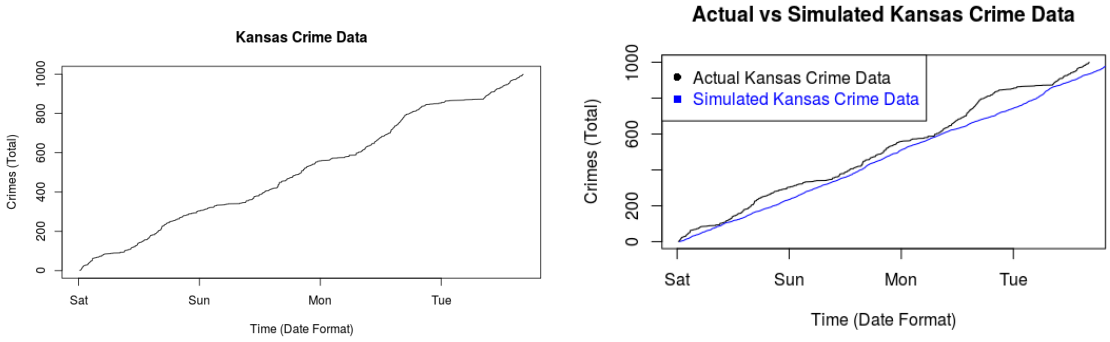


Figure 8.8: Kansas City Crime Data and Replication.

I will again implement the predicted cascade size method laid out in Section 7.2 on this data. Again, I identify a point in the data that resembles the beginning of a cluster, the entire data set, up to this point, will be the observation window. I then predict the cascade size at the end of this cluster. I have chosen the point at which the 450^{th} arrival occurred as the beginning of a cluster, this time being Sunday 16:18 (UTC) (Sunday 11:18 am local time). Predicting as far out as Monday 07:30 (UTC) (Monday 2:30 am local time), at this time, the actual number of crimes was at 600. Very impressively, the predicted cascade size function estimated that at Monday 07:30 (UTC) there would be 600.0063 arrivals. Again, this prediction algorithm has shown itself to be very accurate and shows great use for practical application. The realisation of this prediction is shown in Fig. 8.9 below.

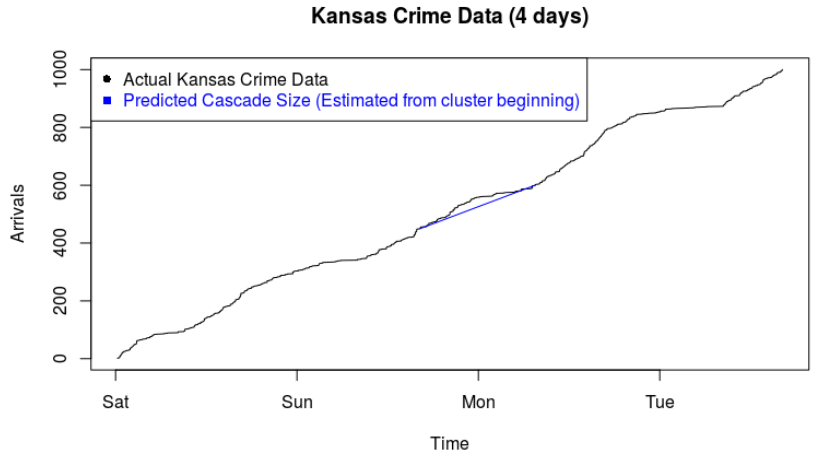


Figure 8.9: Kansas City Crime Data Prediction.

8.3 Hawkes Processes Application to Finance

My final investigation into the practical applications of Hawkes processes is on finance. For this, I decided to explore S&P500 data, specifically, to model volatility in the financial markets.

This data set is the market index price of the S&P500 over 35 years, beginning in 02/01/1985 and spans as far as 31/12/2019. The specific "event" I modelled is the occurrence of "volatile days", I have defined a volatile day as a trading day where the S&P500 open price differs from the closing price by over 1% (either up or down). My reason for choosing the value of 1% as the cutoff point is backed up by a common consensus that in the case where a market moves by greater than 1% over a sustained period of time, one day in this case, this is considered a "volatile market" [23]. The reason I chose to use the S&P500 index as opposed to any other index is because I see it as a much better metric for market movement, better than, say, the price-weighted index of the Dow Jones Industrial Average. After calculating the volatility of each of the 8822 trading days, there were 2054 volatile dates, this number of data points will again ensure an acceptable level of accuracy in my analysis as we've seen from Section 7.1. Plotting the counting process of these dates gives the following graph.

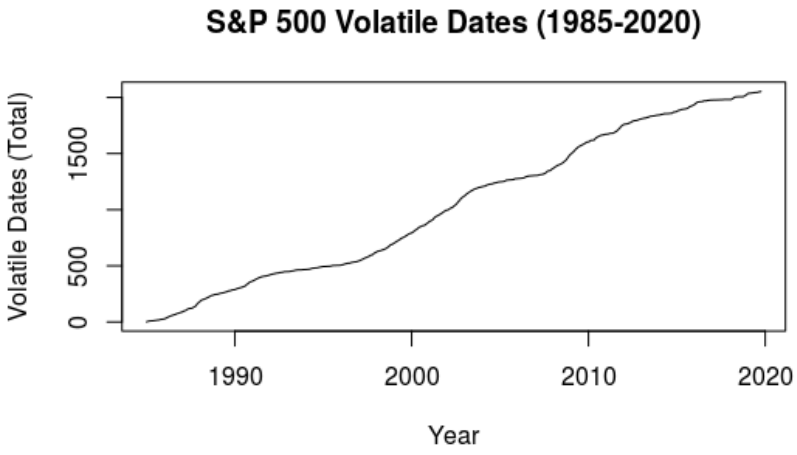


Figure 8.10: S&P500 Volatile Dates.

It is clear to see a large amount of clustering in this data; this would be expected as volatility tends to incite further volatility (a perfect example of a self-exciting process). Based on the distribution of volatile dates, this data looks to be very well suited to be modelled by a Hawkes process. Estimating the parameters of this process gave $\alpha = 2.04$, $\beta = 2.41$ and $\lambda = 2.52$. Using these parameters in the multiple simulations algorithm gave me a model of the overall trend of this data. This trend plotted on the same graph as the original data yielded the following.

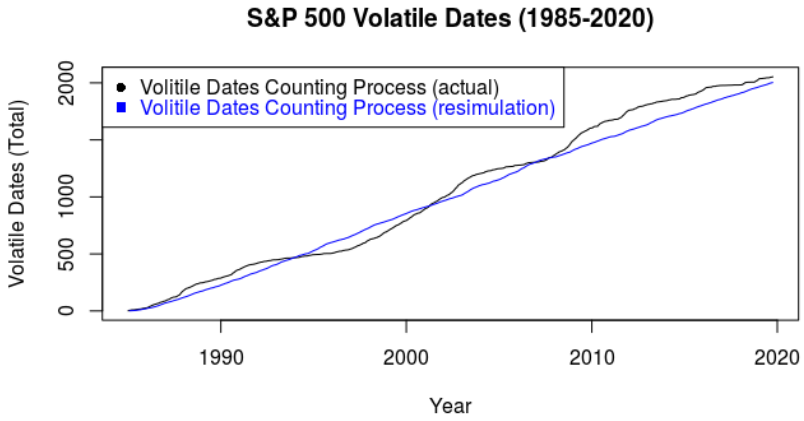


Figure 8.11: S&P500 Volatile Dates.

I then implemented the predicted cascade size algorithm on this data using the first 1300 arrivals (up to 15/06/2006) as the observation window. I wanted to predict as far out as 04/10/2019, the date at which the 2054^{th} and final volatile trading day in this data set occurred. This gave me a predicted cascade size of 2076.926, which is very close to the actual cascade size of 2054. The plot below illustrates the accuracy of this prediction.

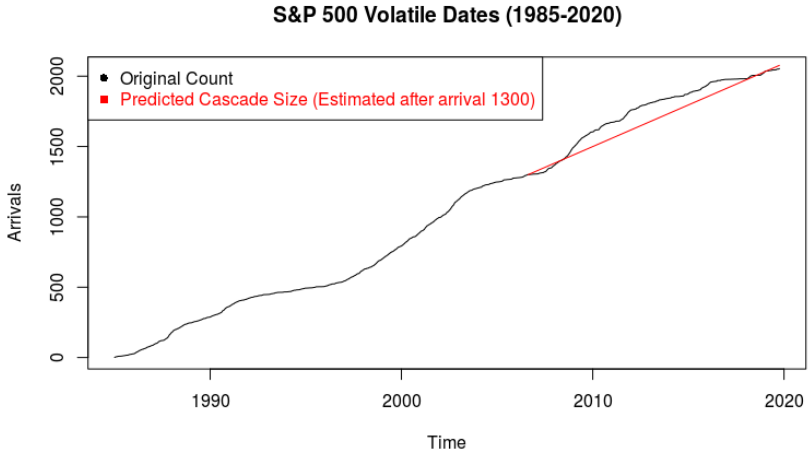


Figure 8.12: S&P500 Volatile Dates and Predicted Cascade Size.

I believe the best use for this analysis would again be as a rolling calculation to predict short-term future volatility, much like the case with short-term bursts in crime data that I've modelled in Section 8.2. This can be achieved using a method like the one described above.

One of the most important analyses implemented by an equity trader is the analysis of risk, and as all traders know; where there is risk, there is opportunity for reward. Risk plays such a

huge part in the trading of financial instruments that the motto of "turn[ing] risk into a path for growth" is used by leading global advisory, broking and solutions companies such as Willis Towers Watson [24]. Therefore, the ability to forecast risk in the financial markets is one of the most important weapons in the arsenal of an equity trader. That is why I will turn my attention to the prediction of risk, again, taking risk to mean, in this case, volatility in the financial markets. Using the predicted cascade size algorithm, I can predict the volatility in the S&P500 out to any time point. I decided to predict as far as the end of the year 2039 as this would give a clear trajectory for volatility in the future, given the previous 35 years of data. This analysis predicted an additional 1186.847 volatile trading days to occur by the date 05/10/2039. This means that out of the 5160 trading days, 23% of them will be "volatile". A visualisation of this prediction is shown below.

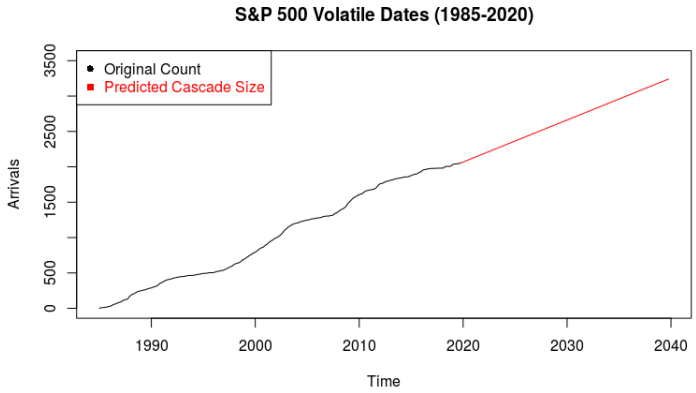


Figure 8.13: S&P500 Volatile Dates and Predicted Future Cascade Size.

The code and data used in my analysis throughout my report are available at
`http://danieloshea.ie/more/modelling-with-hawkes-processes/`

Chapter 9

Conclusion

To illustrate my conclusion on the modelling of Hawkes processes, I will implement a final test on the S&P500 volatile dates data set.

I decided to compare how these different methods of future cascade size prediction differ on a large scale. To do this, in the case of the predicted cascade size algorithm (m), I took the entire data set as the observation window and predicted the cascade size for another 35 years. This will mean that the prediction window is the same size as the observation window. In the case of the resimulation algorithm, I predicted, using the estimated parameters, for another 2054 arrivals, starting at the end of the actual last observed arrival in this data set. This will also predict arrivals for the same amount of time as the observed data set. Fig. 9.1 below shows the realisation of these two prediction algorithms.

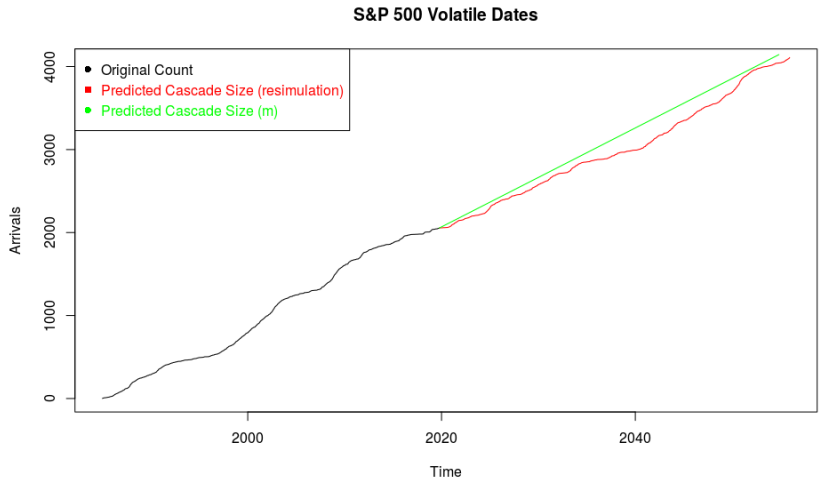


Figure 9.1: S&P500 Volatile Dates and Predicted Future Cascade Size.

The most important item to note is the fact that they converge on almost the exact same point, but take different paths to get there. The predicted cascade size (m) algorithm has shown itself to be extremely accurate in many cases over a short prediction window. The downfall of this algorithm is the fact that the realisation of its prediction is linear, and therefore does not predict clusterings of arrivals over a large prediction window. In the case of the resimulation algorithm, I see its main advantage as its ability to predict the occurrence of clustered arrivals over a long term prediction window, given the data observed. The disadvantage of this method, in my opinion, is that the clusters of arrivals, when predicting their occurrence in the very near future, seem to be almost arbitrary in their manifestation.

9.1 Research and Business Use

For this reason, when recommending further research and the practical application of Hawkes process modelling algorithms, I would recommend the implementation of an algorithm like the one described in Section 8.2 using the predicted cascade size (m) method. The reasons for this recommendation, along with the ones described in the paragraph above, come down to the current state of scientific computation and the latest in the trends of data analysis. The super-fast and efficient collection of data (of all kinds) leads to a greater need for fast analysis of this data in order to take full advantage of the recency of the information. An example of this being used is in social media attention engineering algorithms, which are developed by companies whose business model is based on advertising, the efficiency of which increases with retention time. Maximising retention time is achieved by selectively ranking and arranging content to fulfil cravings of instant gratification based on the behaviour of the user, the data describing which, is collected instantly with any interaction with the content previously shown. These algorithms, when they work effectively, can reap untold actionable results and make vast amounts of profit, and companies continue to spend millions on developing them.

This type of business demand has led to the acceleration in the development of computer hardware and software that can manage to both collect and analyse this data efficiently and quickly.

9.2 Software

The focused development of efficient cloud server software to run these processes and algorithms is evident across the tech sector. For example, Microsoft, developer of Windows OS, uses a kernel of the open-source operating system *Linux* to run Azure, their cloud computing service. This kernel has been developed to be extremely efficient in its computation and completely stripped of all unnecessary background processes and tasks so that the full power of the hardware available can be used. The fact that there is so much work being put into efficient cloud computing software, that the developer of the most popular operating system on earth uses a different

operating system to run their cloud computing services, is evidence of this effort.

9.3 Hardware

Moore's Law may be slowing down, but the current state of computer hardware still shows growth year after year. Just ten years ago, the most powerful supercomputer was the *Tianhe-IA* which had a peak speed of 2.566 petaflops (1 petaflop is 10^{15} floating-point operations per second). Today, the most powerful supercomputer, the *IBM Summit* has a peak speed of 122.3 petaflops. This represents an almost 50-fold increase in ten years.

9.4 Further Application

I believe the predicted cascade size algorithm could be applied to many self-exciting processes in the manner I've described. Along with the case of efficient police dispatchment from Section 8.2 and on-the-fly financial market risk management from Section 8.3, I will outline some other ideas that come to mind. An algorithm like this could be applied in cases such as the following:

The business of Amazon Web Services (AWS) is to allocate server space to manage the internet traffic, of companies like Netflix and Spotify, in such a way that customers can stream their content without interruption or slowing data speed. When consumers request data packets from these servers, it can happen quite suddenly and then all at once. An algorithm that uses the observed internet traffic and then anticipates spikes in demand could be used by companies like AWS to effectively manage sever space allocation.

In cases of a sudden disease pandemic, the ability to model the spread of the disease in real-time and predict areas of future contraction can be extremely beneficial. The actionable results coming from this would be to advise on best practices and for the allocation of scarce and vital resources to at-risk areas, or soon-to-be at-risk areas.

In the case of supply management, when a retail trend starts to emerge in the current world of social media marketing, retail companies can see sudden influxes of orders for popular products. This is before the wholesaler is even aware that this product has become a hit in the mind of the consumer. In this case, it is extremely important for the seller of the product to have their orders in place from the manufacturer in the common case where the manufacturer cannot meet this unexpected demand. In this situation, if a seller has the bulk of the orders for this hot product, this distributor can then corner the market and gain a huge advantage in future name recognition over other retailers.

9.5 Discussion

We have seen that Hawkes processes have been used to model real-world self-exciting point processes, however, using these models to find immediately actionable results, isn't something that I've seen much of in the literature around Hawkes processes.

I am happy to have explored this topic as I would like to see further research in the development of these types of algorithms. I hope to have given a comprehensive review of the current state of Hawkes processes and also to have inspired some sort of confidence in the reliability and accuracy in the application of Hawkes processes to model self-exciting point processes. It is clear to me that many processes in the current world of business seem to follow that of a Hawkes process, and could be profitably analysed by these algorithms.

Bibliography

[1] Patrick J. Laub, Thomas Taimre, Philip K. Pollett. Hawkes Processes, arXiv preprint, arXiv:1507.02822, 2015.

[2] Anatoliy Swishchuk. Hawkes Processes and their Applications in Finance and Insurance. University of Calgary, Calgary, Alberta, Canada, 2017.

[3] Stephen Crowley. Exponential Hawkes Processes. viXra preprint, viXra:1211.0094v9, 2015.

[4] D. Daley, D. Vere-Jones. An Introduction to the Theory of Point Processes: Volume I: Elementary Theory and Methods, 2003.

[5] J.G. Rasmussen. Temporal point processes: the conditional intensity function, arXiv preprint, arXiv:1806.00221v1, 2018.

[6] Patrick J. Laub. Hawkes Processes: Simulation, Estimation, and Validation. Bachelors Thesis, University of Queensland, Queensland, Australia, 2014.

[7] Lingjiong Zhu. Nonlinear Hawkes Processes, arXiv preprint, arXiv:1304.7531v3, 2013.

[8] Lingjiong Zhu. Central Limit Theorem for Nonlinear Hawkes Processes, *Journal of Applied Probability* 2013, Vol. 50, No. 3, 760-771. arXiv preprint, arXiv:1204.1067v3, 2014.

[9] Alex Reinhart. A Review of Self-Exciting Spatio-Temporal Point Processes and Their Applications, *Statistical Science* 33 (2018), no. 3, pp. 299-318. arXiv preprint, arXiv:1708.02647v2, 2018.

[10] Sebastian Meyer, Johannes Elias and Michael Höhle. A Space—Time Conditional Intensity Model for Invasive Meningococcal Disease Occurrence. *Biometrics* Vol. 68, No. 2, pp. 607-616, 2012.

[11] Marian-Andrei Rizoiu, Young Lee, Swapnil Mishra, Lexing Xie. A Tutorial on Hawkes

Processes for Events in Social Media, arXiv preprint, arXiv:1708.06401v2, 2017.

[12] G. O. Mohler, M. B. Short, P. J. Brantingham, F. P. Schoenberg, G. E. Tita. Self-Exciting Point Process Modeling of Crime. *Journal of the American Statistical Association.* 106. 100-108. 10.1198/jasa.2011.ap09546, 2011.

[13] Patricia Reynaud-Bouret, Sophie Schbath. Adaptive Estimation for Hawkes Processes; Application to Genome Analysis. *Annals of Statistics* 2010, Vol. 38, No. 5, 2781-2822. arXiv preprint, arXiv:0903.2919v4, 2010.

[14] Rafael Lima, Jaesik Choi. Hawkes Process Kernel Structure Parametric Search with Renormalization Factors. arXiv preprint, arXiv:1805.09570v3, 2018.

[15] Thomas Josef Liniger. Multivariate Hawkes Processes, 2009. `https://www.research-collection.ethz.ch/handle/20.500.11850/151886`

[16] Gerold Alsmeyer. University of Münster. `https://www.uni-muenster.de/Stochastik/lehre/WS1314/BachelorWT/Daten/StPro_Ross1.pdf` [Accessed 13 Jan. 2020].

[17] Radha Krishna. MLE of Hawkes' Self-Exciting Point Process. `https://radhakrishna.typepad.com/mle-of-hawkes-self-exciting-process.pdf` [Accessed 06 Mar. 2020].

[18] Joseph D. O'Brien, Alberto Aleta, Yamir Moreno, James P. Gleeson. Quantifying Uncertainty in a Predictive Model for Popularity Dynamics. arXiv preprint, arXiv:2001.09490v1, 2020.

[19] M. B. Short, M. R. D'Orsogna, P. J. Brantingham, G. E. Tita. Measuring and Modeling Repeat and Near-Repeat Burglary Effects, 2009. `https://doi.org/10.1007/s10940-009-9068-8` [Accessed 09 Mar. 2020].

[20] `https://data.baltimorecity.gov/Public-Safety/BPD-Part-1-Victim-Based-Crime-Data/wsfq-mvij` [Accessed 11 Feb. 2020].

[21] `https://data.world/data-society/kansas-city-crime-data` [Accessed 09 Feb. 2020].

[22] `https://finance.yahoo.com/quote/%5EGSPC/history/` [Accessed 18 Feb. 2020].

[23] `https://www.investopedia.com/terms/v/volatility.asp` [Accessed 14 Mar. 2020].

[24] `http://www.willis.ie/About/Careers` [Accessed 15 Mar. 2020].

Appendices

Appendix A

Some Appendix

A.1 Poisson Process

Definition: Poisson Process [16]
A counting process $N(t)$ is said to be a *Poisson process* with rate (or intensity) λ (> 0), if:

- $N(0) = 0$.
- The process has independent increments.
- The number of events in any time interval of length t is Poisson distributed with mean λt. That is, $N((s,t]) \equiv Poi(\lambda t)$ for all s, $t \geq 0$:

$$\mathbb{P}\Big(N((s,t]) = n\Big) = e^{-\lambda t}\frac{(\lambda t)^n}{n!}, n \in \mathbb{N}_0 \tag{A.1}$$

A.2 Calculating Log-Likelihood Equations

I will include here the log-likelihood calculations for $k = 3$ onward:

k = 3

$$l = \log(\lambda) + \log(\lambda + \alpha(e^{-\beta})) + \log(\lambda + \alpha(e^{-2\beta} + e^{-3\beta})) - 4\lambda + \frac{\alpha}{\beta}[(e^{-2\beta} - 1) + (e^{-\beta} - 1)].$$

k = 4

$$l = \log(\lambda) + \log(\lambda + \alpha(e^{-\beta})) + \log(\lambda + \alpha(e^{-2\beta} + e^{-3\beta})) + \log(\lambda + \alpha(e^{-\beta} + e^{-3\beta} + e^{-4\beta})) - 5\lambda + \frac{\alpha}{\beta}[(e^{-4\beta} - 1) + (e^{-3\beta} - 1) + (e^{-\beta} - 1)].$$

k = 5

$$l = \log(\lambda) + \log(\lambda + \alpha(e^{-\beta})) + \log(\lambda + \alpha(e^{-2\beta} + e^{-3\beta})) + \log(\lambda + \alpha(e^{-\beta} + e^{-3\beta} + e^{-4\beta})) + \log(\lambda + \alpha(e^{-2\beta} + e^{-3\beta} + e^{-5\beta} + e^{-6\beta})) - 7\lambda + \frac{\alpha}{\beta}[(e^{-6\beta} - 1) + (e^{-5\beta} - 1) + (e^{-3\beta} - 1) + (e^{-2\beta} - 1)].$$

k = 6

$$l = \log(\lambda) + \log(\lambda + \alpha(e^{-\beta})) + \log(\lambda + \alpha(e^{-2\beta} + e^{-3\beta})) + \log(\lambda + \alpha(e^{-\beta} + e^{-3\beta} + e^{-4\beta})) + \log(\lambda + \alpha(e^{-2\beta} + e^{-3\beta} + e^{-5\beta} + e^{-6\beta})) + \log(\lambda + \alpha(e^{-3\beta} + e^{-5\beta} + e^{-6\beta} + e^{-8\beta} + e^{-9\beta})) - 10\lambda + \frac{\alpha}{\beta}[(e^{-9\beta} - 1) + (e^{-8\beta} - 1) + (e^{-6\beta} - 1) + (e^{-5\beta} - 1) + (e^{-3\beta} - 1)].$$

k = 7

$$l = \log(\lambda) + \log(\lambda + \alpha(e^{-\beta})) + \log(\lambda + \alpha(e^{-2\beta} + e^{-3\beta})) + \log(\lambda + \alpha(e^{-\beta} + e^{-3\beta} + e^{-4\beta})) + \log(\lambda + \alpha(e^{-2\beta} + e^{-3\beta} + e^{-5\beta}) + e^{-6\beta})) + \log(\lambda + \alpha(e^{-3\beta} + e^{-5\beta} + e^{-6\beta} + e^{-8\beta} + e^{-9\beta})) + \log(\lambda + \alpha(e^{-\beta} + e^{-4\beta} + e^{-6\beta}) + e^{-7\beta} + e^{-9\beta} + e^{-10\beta})) - 11\lambda + \frac{\alpha}{\beta}[(e^{-10\beta} - 1) + (e^{-9\beta} - 1) + (e^{-7\beta} - 1) + (e^{-6\beta} - 1) + (e^{-4\beta} - 1) + (e^{-1\beta} - 1)].$$