MATH5004M: Assignment in Mathematics

Bayesian Sports Modelling

Supervisor: Dr. John Paul Gosling

Ryan Chan

200850644

May 4, 2018

Abstract

We consider the task of predicting football results in the Premier League and propose a Bayesian hierarchical model. In our model, we aim to estimate the characteristics of teams that contribute to a team either winning or losing a football match and use these to predict the score or outcome of future games. The model will be implemented using the Bayesian inference software Stan and the statistical programming language R. We also discuss several methods to test the predictive strength of our model and use these methods to compare with other models implemented previously. The data used throughout the report is taken from the *Football-Data* website, on `http://www.football-data.co.uk/`.

Contents

1 Introduction

1.1 Data in football today

Association football (also known as *soccer*) is one of the most popular sports in the world (Economist, 2011) and it is a team sport that is played between two teams, which is made up of eleven players. The main objective of the game is to score more goals than the opposing team and the team that scores more goals wins the match. The game ends in a draw if the number of goals scored by the two teams are equal.

Football is played in countries all around the world and there are many football clubs that compete for regional or national leagues. In addition, there are also many continental championships that are played, such as the UEFA Champions League, which is contested by top-division European clubs. In this report, we will be focusing on the English Premier League, which follows the *'double round robin tournament'* format. This means that each team plays each other twice (home and away). In the Premier League, there are 20 teams, meaning that over the course of the season each team plays 38 times. The Premier League is the top division in English football and every year the bottom 3 teams are relegated and replaced by 3 teams from the Championship, the league below the Premier League. The points system follows the standard 3-1-0 system, where a win is awarded with 3 points, a draw is awarded 1 point and no points are given for a loss.

Building a model to predict the outcomes of future matches and to the final league table has obvious applications to gambling and betting markets, where an accurate predictive model can be very profitable. The Premier League is now one of the richest league in the world (Deloitte, 2017) and is followed globally around the world. As in any sport, there is an interest in trying to predict the winner of any two teams. For instance, with the continuous rise of online gambling, betting companies try to do so for most professional football games and try to set their odds correctly and so there is motivation to build more accurate models. Decision analysis and Bayesian decision rules can be applied to determine which bets/actions should be taken after a model is made to give predictions to future outcomes. Moreover, other parties interested in building such models are football teams/clubs themselves to use as a tool to assess their performances, identify strengths or weaknesses and develop new tactics or strategies. In other sports outside football there have been a growth in incorporating the use of data in analysis. For example, in '*Moneyball*', Lewis (2003), discusses how the Oakland Athletics baseball team took advantage of more analytical gauges of performances to acquire players that were undervalued to compete against richer teams in the league.

Football is a complex sport, since teams can play various possible strategies, formations or tactics which are carried out by eleven players for each team, and so determining accurate probabilities for match outcomes is not simple. Reep and Benjamin (1968, p.585) looked at the number of passes made before shots or goals and observed there was a stochastic element in the number of goals coming from a particular number of shots in a match. They inferred then that an excess of shots by one team does not mean that the other side would not get more goals and win the match and as such concluded, "chance does dominate the game".

Anderson & Sally (2014, p.52) collected data from twenty betting exchanges along with final scores from the 2010/11 season in the NBA (Basketball), NFL (American Football), MLB (Baseball), Handball Bundesliga (in Germany) and the football scores from the top leagues in England, France, Spain, Italy and Germany, and the Champions League. They used this data to measure how well bookmakers were picking the favourites. From this data, they found that in football only little over half of the time the bookmakers chose the favourites. However, in handball, basketball and American football, the favourites would win around two-thirds of their games, while in baseball it is about 60%. In Handball, they found that more than 70% of games were won by the favourites. This may be the case for several reasons; the one suggested by Anderson & Sally (2014, p.52) is that generally in football, 'the favourite is not really much of a favourite', meaning that in many matches, the favourites were only narrowly favoured. Another possible reason is that in football, goals are more rare and draws are common. In contrast, basketball has more high scoring matches since the court is much smaller than football pitches so distance to the goal/hoop is smaller and in basketball, there is a rule which states that after a team gains possession of the ball, they must shoot within a given time. Since there is a high frequency of shots in basketball, it seems intuitive that the better team will generally be able to score more points than their opponents. Therefore, they found that compared to other sports such as basketball and baseball, football was the most uncertain of team sports.

A study by Ben-Naim, Vazquez & Redner (2005) supports this claim as they used computer simulations of artificial sports leagues using data looking back to the top flight of English football since 1888, the MLB since 1901, NHL (hockey) since 1917 and the NFL since 1922 (300,000 total games) and found that the likelihood of the underdog winning in football was around 45%, the highest of all the sports mentioned above.

This report will aim to tackle the problem of predicting football match outcomes and we will develop and implement a Bayesian hierarchical model in the programming languages R (The R Core Team, 2017) and Stan (Stan Development Team, 2017).

1.2 Overview of report

The report will be set out as follows: Section 2 provides an introduction to Bayesian statistics and its methods for inference. In the models that we discuss later in the report, we will use a Bayesian hierarchical model so Section 2 gives the theoretical framework that we will be using to obtain predictions for the outcomes of football matches. To implement these models, we will be using the Stan programming language (Stan Development Team, 2017) and the R Statistical language (The R Core Team, 2017). Stan is a programming language that allows us to implement a Markov Chain Monte Carlo method called *Hamiltonian Monte Carlo* to obtain random samples from a probability distribution where direct sampling is difficult. In Section 3, we will briefly explain the concepts of Markov Chain Monte Carlo and Hamiltonian Monte Carlo.

Prior to building our first model to predict football matches, we give a short literature review on past works and assess the assumptions made in these models in Section 4. Further, in Section 5 of the report, we discuss a Bayesian hierarchical model developed by Baio & Blangiardo (2010) that uses a Poisson distribution to model the number of goals scored by the home and away team and will implement the model in the Stan programming language and R.

In Section 6, we start to develop a different Bayesian hierarchical that uses a negative binomial distribution to model the goals scored by each team and present some results of the model.

To assess the models discussed, we use various methods such as Cross-Validation, the Brier score, Rank Probability Score (RPS) (all of these will be defined later) and we also assess model performance by attempting to predict a league table and by using it as a betting model. The results of these scores and assessments can be found in Section 7.

2 Short Introduction to Bayesian Statistics

A *statistical model* is a set of probability distributions on the sample space $\mathcal{S}$ (McCullagh, 2002, p.1225). For example, to model goals scored by football teams, the sample space would be $\mathcal{S} = \{0, 1, 2, 3, ...\}$. A model can be a useful imitation of a real world system or process. One important aim of modelling is to try and predict the effect of a change in the system or to predict future values or events. McCullagh (2002, p.1225) defines a *parametrised statistical model* as a parameter set Θ together with a function $\Pi : \Theta \to \mathcal{P}(\mathcal{S})$. This function Π assigns each parameter point $\theta \in \Theta$ to a probability distribution π_θ on $\mathcal{S}$, where $\mathcal{P}(\mathcal{S})$ is the set of all probability distributions on $\mathcal{S}$. For this project, we wish to create a statistical model to model the number of goals scored by each football team in a game with the aim to predict future game outcomes.

Bayesian statistics provides a framework within which uncertainties in real-world problems can be quantified. The frequentist (or classic) approach to statistics treats parameters as fixed and unknown quantities that are to be estimated, while the Bayesian approach treats parameters as random variables.

In a frequentist approach to statistics, probability is viewed as "a limiting ratio in a sequence of repeatable events" with "the ratio becoming ever more exact as the series is extended" (Howie, 2002, p.1). Hence, statistical models are based on frequencies of events. For instance, to assess the distributional fit of a model, *hypothesis tests* are generally used and *p-values* are calculated, which are defined to be the probability under the null hypothesis of obtaining a result equal or more extreme that was observed (Dahiru, 2008, p.22). Therefore, the p-value is a measure of "discrepancy between the data and the null hypothesis" (Goodman, 1999, p.1005). In Section 4, we use one of these tests, the *Chi-Squared Test* to assess the distributional fits to outcome data. However, we use these as a tool for exploratory analysis before building our own model rather than to accept or reject a potential distribution for our model.

The Bayesian approach to statistics takes its name from *Bayes's Theorem* (Box & Tiao, 1992, p.10). This theorem allows us to make probability statements about a parameter θ given some data x. Using that the joint probability mass or density function can be written as a product of the *prior distribution*, $\pi(\theta)$ and the *data distribution*, $\pi(x \mid \theta)$ (Gelman *et. al*, 2014, p.6),

$$\pi(\theta, x) = \pi(x \mid \theta)\pi(\theta),$$

then by the rule of conditional probability, we have *Bayes's Theorem*,

$$\pi(\theta \mid x) = \frac{\pi(x \mid \theta)\pi(\theta)}{\pi(x)}, \tag{1}$$

where, $\pi(\theta \mid x)$ denotes the conditional probability of observing θ given some data x and $\pi(x)$ is the marginal distribution.

Equivalently, for fixed x, $\pi(x)$ can be considered a constant, hence, we have

$$\pi(\theta \mid x) \propto \pi(x \mid \theta)\pi(\theta). \tag{2}$$

Note that the *data distribution* is proportional to the *likelihood* Bolstad (2007, p.67), so $\pi(x \mid \theta) \propto l(\theta; x)$ and Bayes's Theorem can also be written as,

$$\pi(\theta \mid x) \propto l(\theta; x)\pi(\theta). \tag{3}$$

The constant of proportionality, $\frac{1}{\pi(x)}$, in equation 2 can be found through the law of total probability, since

$$\pi(x) = \int_{\Theta} \pi(x \mid \theta)\pi(\theta)d\theta = \int_{\Theta} \pi(x, \theta)d\theta.$$

Lee (2012, p.36) states that broadly speaking, in Bayesian inference, we take prior beliefs about various hypothesis and then modify these prior beliefs in the light of data to obtain posterior beliefs. Bayesian inference is based upon equation 2, as we take this to be our *posterior distribution*, which encodes what we believe after seeing some data. So after we observe some data x, we update our beliefs using equation 2. Note that in the current notation, we are implicitly assuming the conditioning in the model, whereas other models will alter these. In a more general model, the parameter θ may depend on another parameter μ and so equation 2 becomes

$$\pi(\theta \mid x, \mu) \propto \pi(x \mid \theta, \mu)\pi(\theta \mid \mu). \tag{4}$$

This is important to note, since later we propose a hierarchical model, where parameters will depend on another set of parameters called *hyperparameters*; see Section 2.6.

2.1 Prior distribution

Let θ be an uncertain parameter, then in Bayesian inference, where we treat parameters as random variables, we use the *prior density*, $\pi(\theta)$ (or $\pi(\theta \mid \mu)$ if θ depends on another parameter μ), to encode our uncertainty and prior beliefs about the parameter θ.

2.2 The Likelihood (or data distribution)

From equation 4, we see that the posterior inference is only affected by the data through $\pi(x \mid \theta, \mu)$. The *likelihood* tells us the probability of observing the data, x, if the model parameter is θ, and this encodes our beliefs about how the data is generated:

$$l(\theta; x, \mu) \propto \pi(x \mid \theta, \mu)$$

In contrast, in frequentist statistics, *p-values* are considered, which look at the probability of seeing such data, x, (or something more extreme) given the null hypothesis is true, rather than a value of a parameter θ, since parameters are fixed in the frequentist framework.

Gelman *et al.* (2014, p.8) notes that Bayesian inference obeys the *likelihood principle*, which states that for a given sample of data, any two probability models that have the same likelihood function should give the same inference for θ. This means that the inference should only be affected by the new data through the likelihood and that we should make the same inferences for θ after observing x or y if $l(\theta; x) \propto l(\theta; y)$.

2.3 Predictive distribution

After observing the data x, we are still uncertain about the parameter θ and so we are still uncertain about the future data $\tilde{x}$, say. Then the distribution of $\tilde{x}$ is known as the *predictive distribution*, sometimes known as the *posterior predictive distribution* and is defined as

$$\begin{aligned}
\pi(\tilde{x} \mid x, \mu) &= \int_\Theta \pi(\tilde{x}, \theta \mid x, \mu) d\theta \\
&= \int_\Theta \pi(\tilde{x} \mid \theta, \mu, x) \pi(\theta \mid \mu, x) d\theta \\
&= \int_\Theta \pi(\tilde{x} \mid \theta, \mu) \pi(\theta \mid x) d\theta \\
&= E_{\Theta \mid x}[\pi(\tilde{x} \mid \theta, \mu)]
\end{aligned}.$$

Lee (2012, p.39) notes that when there are no observations to take into account, this is called the *preposterior distribution*, which encodes our uncertainty about what value of x we will observe before any data is available to us yet. Hence, we have seen that in the Bayesian approach, we express our uncertainty for parameters by specifying *prior distributions* and then we use sample data and Bayes's Theorem to arrive at a *posterior probability distribution* in which we make predictions and inferences from.

2.4 Hypothesis testing

As stated above, in a frequentist approach to statistics, *hypothesis tests* are used and *p-values* are calculated to determine if they can accept a null hypothesis or not (a *null hypothesis* in a goodness-of-fit test is the hypothesis that there is no significant difference between specified populations). In the Bayesian approach to hypothesis testing, if we were forced to act accordingly to whether we believed in a hypothesis or not, we just need to calculate the posterior probabilities (Lee, 2004, p.118). Suppose that we have two hypotheses, $H_0 : \theta \in R$ (the *null hypothesis*), and $H_1 : \theta \notin R$, (the *alternative hypothesis*), then we just calculate the posterior probabilities,

$$\pi_0 = \pi(\theta \in R \mid x) \text{ and } \pi_1 = \pi(\theta \notin R \mid x),$$

and then decide between H_0 and H_1 accordingly.

2.5 Point and interval estimation

For point estimation, a single statistic is calculated from sample data and used to estimate an unknown parameter (Bolstad, 2007, p.163). As stated before, in frequentist statistics, unknown parameters are treated as fixed to be estimated. Popular methods to estimate parameters include *maximum likelihood estimation* or *method of moment estimates* (for more details about frequentist statistics and such methods, see Rice, 1995). From a Bayesian perspective, point estimation uses a single statistic to summarise the posterior distribution. An obvious posterior estimate of θ is the *posterior mean*, defined as

$$\theta_M = E_{\theta \mid x}[\theta \mid x].$$

Other posterior estimates include the median and the mode of the posterior distribution (also known as the *maximum a posteriori (MAP) estimate*). In this report, we will mainly use either the posterior mean or the MAP estimate as a Bayesian point estimate for an unknown parameter θ. However, in Section 4 where we perform some exploratory analysis on the dataset, for simplicity we will use maximum likelihood estimation where we can (the *maximum likelihood estimate (MLE)* for a parameter θ is the value of θ that maximises the likelihood (Rice, 1995, p.254)). The reason for this is because as the number of data increases, the MAP gets closer to the MLE.

Consider a set of data $\boldsymbol{x} = (x_1, ..., x_n)$ since the maximum likelihood estimate is defined as

$$\begin{aligned}
\theta_{MLE} &= \underset{\theta}{\operatorname{argmax}}\, l(\theta; \boldsymbol{x}) \\
&= \underset{\theta}{\operatorname{argmax}} \log\, l(\theta; \boldsymbol{x}) \\
&= \underset{\theta}{\operatorname{argmax}} \log \prod_i l(\theta; x_i) \\
&= \underset{\theta}{\operatorname{argmax}} \sum_i \log\, l(\theta; x_i)
\end{aligned}$$

Then for the maximum apriori estimate, since this is the mode of the posterior density given in equation 2, then this is defined as

$$\begin{aligned}
\theta_{MAP} &= \underset{\theta}{\operatorname{argmax}}\, l(\theta; \boldsymbol{x})\pi(\theta) \\
&= \underset{\theta}{\operatorname{argmax}} \log\, l(\theta; \boldsymbol{x})\pi(\theta) \\
&= \underset{\theta}{\operatorname{argmax}} \log \prod_i l(\theta; x_i)\pi(\theta) \\
&= \underset{\theta}{\operatorname{argmax}} \sum_i \log\, l(\theta; x_i)\pi(\theta)
\end{aligned}$$

By comparing these two expressions, the only difference is that in the MAP estimate, the prior density for θ is included. But for Bayesian approaches, under some sensible conditions (not discussed here), it can be shown that the asymptotic posterior distribution does not depend on the chosen prior distribution. For more details on the asymptotic behaviour of posterior distributions see Hartigan (1983) and Walker (1967). Hence for simplicity and for the use of exploratory analysis, we use the MLE to estimate parameters in Section 4.

For interval estimation, in the frequentist/classical statistical framework, *confidence intervals* are used. These are often misinterpreted to mean there is a $(\alpha \times 100)\%$, with $0 < \alpha < 1$, chance that the true value of a parameter θ is in the interval, however this is not the case. The correct interpretation is that $(\alpha \times 100)\%$ of the random intervals calculated in this frequentist way will contain the true value of the parameter (Bolstad, 2007, p.168), i.e. we are $(\alpha \times 100)\%$ *confident* that our interval will contain the true value. As Gill (2014, p.43) notes, a 95% interval covers the true value of the parameters 19/20 times, on average.

However, in a Bayesian setting, we use the posterior distribution to define *credible intervals*; a $(\alpha \times 100)\%$ *credible interval* for θ is given by (a, b) if $\pi(\theta \in (a, b)) = \alpha$. In this way to perform interval estimation, we are actually stating that we believe there is an $(\alpha \times 100)\%$ chance that the parameter θ falls within this interval. Hence, in this report, we will use credible intervals for interval estimation.

2.6 Bayesian hierarchical models and DAGs

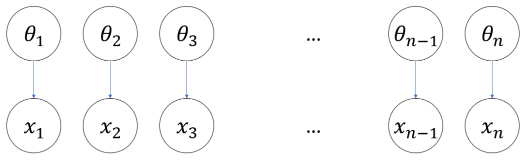
Gelman *et al.* (2014, p.101) note that in hierarchical models, observable outcomes are modelled conditionally on certain parameters, which themselves are also given probabilistic specification in terms of other parameters, which are called *hyperparameters*. In non-hierarchical models, observables are modelled conditionally on a set of parameters, but these parameters do not have any dependence on any other parameters. Hence in an hierarchical model, there is a multilevel parameter conditional structure. For such models, a *directed acyclic graph (DAG)* can be drawn to illustrate the dependency structure of the model.

Diestel (2006, p.2) defines a *graph* to be a pair $G = (V, E)$ of sets such that $E \subseteq [V]^2$. The elements of V are the *vertices* of the graph G, the elements of E are its *edges*. A *directed acyclic graph* (DAG) is a graph where the edges point in a particular direction between the two vertices and there are no cycles, meaning that there are no paths from a vertex to itself. Pearl (2009, p.14) notes that DAGs have been used represent causal or temporal relationships and are sometimes known as *Bayesian networks*, a term coined by Pearl (1985).

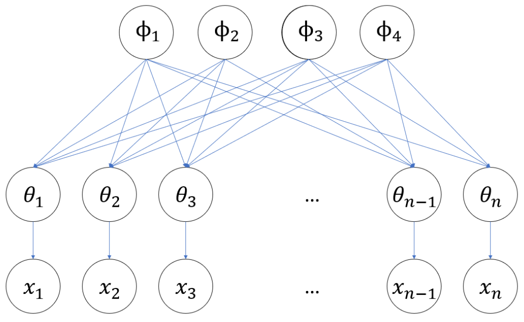
In a Bayesian hierarchical model, we might have some observations $x_1, ..., x_N$, that has a distribution which depends on an unknown set of parameters, $\boldsymbol{\theta} = \{\theta_1, ..., \theta_n\}$, which themselves have a distribution that depends on a set of hyperparameters $\boldsymbol{\phi} = \{\phi_1, ..., \phi_m\}$. Figure 1 shows the difference between a standard non-hierarchical model (1a) and a hierarchical model (1b), where the arrows represent relationships between variables. If there is an arrow from a variable X to Y then we say Y directly depends on X. In the example of the standard non-hierarchical model, each of the x_i for $i = 1, ..., N$, depends on a the parameter θ_i. However, in the hierarchical model, each of the x_i for $i = 1, ..., N$, depends on a the parameter θ_i, but now θ_i parameters depend on another set of parameters, $\boldsymbol{\phi}$ (in these examples, $N = n$ and $m = 4$ in the above notation). The DAG representation of these models show the hierarchical and dependency structure of the models.

With more complicated hierarchical models, we can carry on with this process, as $\boldsymbol{\phi}$ may also depend on one or more *hyper-hyperparameters* $\boldsymbol{\eta}$ and so on to create more levels of dependency. If $\boldsymbol{\eta}$ is unknown we can specify *hyper-hyperprior distributions* to represent our beliefs about $\boldsymbol{\eta}$. These are useful for modelling more complicated systems, as it involves multiple parameters that are regarded to be connected in some way.

In general, we consider constructing a hierarchical structure where parameters are conditional on some hyperparameters and we set plausible prior distributions on these. For modelling football, we may have the mean number of goals as a parameter to estimate and make this condition on factors that we believe this is dependent on, e.g. attack and defence skill of the teams.



(a) A non-hierarchical model



(b) A simple hierarchical model

Figure 1: Non-hierarchical and hierarchical models

Gelman *et al.* (2014, p.101) stated that hierarchical models are more important in practice, as non-hierarchical models are usually inappropriate for some data, as fewer parameters generally cannot fit large datasets accurately and with more parameters there may be problems with 'overfitting', which is the process where the model fit is too close to the data and has inferior predictive capabilities for new data. However, with hierarchical models, the model has enough parameters to fit the data well while having some dependency structure on the parameters to avoid problems with overfitting.

Note, while using a hierarchical model, assumptions are made about the dependency of different parameters. As stated above, in the area of *causality theory*, DAGs are used to represent causal relationships.

One definition of *independence* between two random variables X and Y is that if $\Pr(X, Y) = \Pr(X)\Pr(Y)$, then X is independent to Y (Rice, 1995, p.35). Further, suppose we have three random variables X, Y and Z, then we say that X and Y are *conditionally independent* given Z if and only if $\Pr(X, Y \mid Z) = \Pr(X \mid Z)\Pr(Y \mid Z)$ (Pearl *et al.*, 1989, p.33). By using DAGs, we can use several graphical criteria such as *d-separation*, the *Back-* and *Front-door criterion*; see Pearl (1985 & 2009) to extract information implied by DAGs about the conditional dependencies between random variables. This is important to us, since in previous works on predicting football games, there has been a debate whether or not to assume independence between goals scored by the home and away team (see Section 4). However, by using a hierarchical model, we do not need to use a bivariate distribution to model the goals scored. Later in Sections 5 and 6, where we discuss two different Bayesian hierarchical models to model the goals scored by the home and away team, we will see the dependency structure of these models imply a form of correlation between the observed variables (i.e. the goals scored by teams), as they depend on a common set of unobservable hyper-parameters.

Therefore, correlation between the goals scored by the home and away team are taken into account without the use of a more complicated bivariate distribution. Although we will not discuss the methods or criteria to show conditional independence between random variables from DAGs, we note that the Bayesian hierarchical models shown in Sections 5 and 6 will assume independent distributions for the home and away goals scored in a match, but the dependency structure implied by the hierarchical model will imply there is a conditional independence between the two observed variables. For more information on causal relationships induced by DAGs, see Pearl (1985, 2009) and Pearl *et al.* (1989).

3 Markov Chain Monte Carlo

In this section, we describe methods for computing posterior distributions that are otherwise intractable to solve analytically. In Bayesian inference, we use *Bayes's Theorem* to obtain the posterior density, which requires integration to find the constant of proportionality (or normalisation constant), since

$$\begin{aligned}\pi(\theta \mid x) &= \frac{\pi(x \mid \theta)\pi(\theta)}{\pi(x)} \\ &= \frac{\pi(x \mid \theta)\pi(\theta)}{\int_\Theta \pi(x \mid \theta)\pi(\theta)d\theta}.\end{aligned}$$

For more complicated posterior distributions, this often becomes more difficult to work with via analytic examination (Ravenzwaaij *et al.*, 2018, p.144) and so alternative methods to estimate this are needed. Further, Barp et al. (2017) state that even when the value of the normalisation constant is known, sampling from π is challenging, particularly in high dimensions. Markov Chain Monte Carlo (MCMC) introduced by Metropolis *et al.* (1953), allows sampling from distributions with intractable normalisation. The Metropolis algorithm (discussed in Section 3.3.1) was introduced to tackle problems in physics with the aim to compute the expected value of physical quantities. This algorithm was later generalised by Hastings (1970) to give the Metropolis-Hastings algorithm (discussed in Section 3.3) and can be used to obtain a sample from a target distribution. The normalisation constant is a key quantity in Bayesian statistics, since it is needed to obtain other quantities, such as the posterior mean and the predictive distribution. In this report, we will be using MCMC sampling methods, in particular Hamiltonian Monte Carlo (HMC), which is used in the Stan modelling language (Stan Development Team, 2017) to automatically apply HMC to a given Bayesian model. This section starts off with some definitions of Markov chains and general Monte Carlo methods to develop the framework for MCMC sampling and then we will present the HMC algorithm.

3.1 Markov Chains

Markov Chain Monte Carlo is a method based on drawing values of θ from approximate distributions and correcting these to have a better approximation to the target posterior distribution (Gelman *et al.*, 2014, p.275). The simulation sampling is sequential, meaning that the distribution of the sampled draws only depends on the last value drawn - hence the draw forms a *Markov chain*.

The definition of a *Markov process* is given by Cook and Upton (2014): "a *Markov process* is a stochastic process $X = \{X_1, X_2, ...\}$ for which the value taken by random variable X_t, for all $t > 2$, is independent of $X_1, X_2, ..., X_{t-2}$, but may depend on X_{t-1}."

This is the *Markov property* and so a sequence of random variables $X_1, X_2, ...$ form a *Markov process* if

$$X_{t+1} \perp\!\!\!\perp \{X_{t-1}, X_{t-2}, ..., X_1\} \mid X_t, \text{ for all } t.$$

If the random variables X_t for $t = 1, 2, ...$ are discrete and take values in some set $\mathcal{S}$, then the process is a *Markov chain.*

The *transition probability*, p_{ij}, of a Markov chain is the probability that a Markov chain in state $i \in \mathcal{S}$ at time $t-1$, is in state $j \in \mathcal{S}$ at time t (Cook and Upton, 2014) and so the transition probabilities can be associated with a matrix P (called the *transition matrix*) having elements p_{ij} defined as

$$p_{ij} = \Pr(X_t = j \mid X_{t-1} = i) \text{ for } i = 1, ..., n \text{ and } j = 1, ...n.$$

A Markov chain is *irreducible* if the state space $\mathcal{S}$ is all connected, meaning for all $i, j \in \mathcal{S}$, there exists some t such that $p_{ij}(t) \geq 0$, where $p_{ij}(t)$ denotes the probability of transitioning from i to j in t steps. Cook and Upton (2014) says that a state is *recurrent* (or *persistent*) if, starting from that state, the probability of ever returning to it is equal to 1. A recurrent state with a finite expected time until return is called a *positive recurrent* (or an *ergodic state*). Roberts (1996, p.46) states that an irreducible chain X is called *aperiodic* if for all $i \in \mathcal{S}$,

$$\textit{greatest common divider } \{t \mid p_{ii}(t) > 0\} = 1,$$

where $p_{ii}(t)$ denotes the probability of returning to state i in t number of steps. If a Markov chain is aperiodic and positive recurrent, then it is called *ergodic.*

Geyer (2011, p.5) states that a stochastic process $X = \{X_1, X_2, ...\}$ is *stationary* if for every positive integer k, the distribution of $(X_{t+1}, ..., X_{t+k})$ does not depend on t. For a Markov chain, if there exists a vector $\boldsymbol{\pi} = (\pi_i \geq 0, 0 \leq i \leq n)$, where $\sum_{i=1}^{n} \pi_i = 1$, such that $\boldsymbol{\pi} P = \boldsymbol{\pi}$, then $\boldsymbol{\pi}$ is called the *stationary distribution* of the Markov chain (Fishman, 1996, p.339). Note that in some Markov chains, there can be a number of different stationary distributions, but if a Markov chain is *irreducible* and *positive recurrent*, then $\boldsymbol{\pi}$ is the unique stationary distribution of the Markov chain. Roberts (1996, p.47) notes that for an aperiodic positive-recurrent Markov chain, the stationary distribution is the *limiting distribution* of successive iterates from the chain and this is true regardless of the starting value of the chain. Fishman (1996, p.340) notes that this is also sometimes called the *equilibrium* or *steady-state distribution* of the chain.

The idea behind Markov chain Monte Carlo simulation (Bart *et al.*, 2017, p.2) is to generate samples from the target density, $\pi(\theta \mid x)$, which are approximately i.i.d. (independent and identically distributed) by creating a Markov chain whose stationary distribution is the target distribution with density $\pi(\theta \mid x)$, and the simulation is run enough times so the distribution of the draws is close enough to this stationary distribution. Robert & Casella (2004, p.268) defines a MCMC method for simulation of a distribution π as any method to produce an ergodic Markov chain whose stationary distribution is π. Hence, if we can find such a Markov chain, we can start the chain at any starting point and make sure the simulation is run for long enough that the probability of being in state θ is given by the target density. Gelman *et al.* (2014, p.275) states that the key to the success of MCMC to approximate target distributions is not the Markov property, but rather that the approximate distributions are improved at each step in the simulation, in the sense of converging to the target distribution.

3.2 Monte Carlo methods

Numerical integration is the study of how the numerical value of an integral can be found (Davis & Rabinowitz, 2007, p.1). In numerical integration, the integral of a function f is evaluated by computing the value of the function at a finite number of points. The estimate becomes more accurate by increasing the number of points where the function f is evaluated. As Gelman *et al.* (2014, p.261) note, numerical integration methods can be divided into stochastic (or simulation) methods such as Monte Carlo, which we will discuss here, and deterministic methods (see Davis & Rabinowitz, 2007). Now, consider the evaluation of the following integral,

$$\mathrm{E}[f(\theta)] = \int_{\Theta} f(\theta)\pi(\theta)d\theta, \tag{5}$$

where $\pi(\theta)$ is some probability density, and $f(\theta)$ is some function. Note that this integral is the expectation of the function $f(\theta)$. In a Bayesian setting, if $\pi(\theta)$ is the posterior density, then an integral of this form is the posterior expectation of the function $f(\theta)$. Rasmussen & Chahramani (2002, p.1) states the Classical Monte Carlo approximation to the above integral:

$$\int_{\Theta} f(\theta)\pi(\theta)d\theta \approx \frac{1}{T}\sum_{t=1}^{T} f\left(\theta^{(t)}\right) = \mathcal{F}_T,$$

where $\theta^{(t)}$ are random simulated draws from $\pi(\theta)$, and this converges to the true value of the integral in the limit of large numbers of samples, T. This estimate is stochastic, since it depends on generated random numbers and so stochastic methods are based on obtaining random samples, θ^t, from the desired distribution $\pi(\theta)$.

The variance of the Monte Carlo approximation can be estimated from the sample $(\theta^{(1)}, ..., \theta^{(T)})$ through the following formula given by Weinzierl (2000, p.12):

$$\mathrm{Var}(\mathcal{F}_t) \approx \frac{1}{T-1} \sum_{t=1}^{T} (f(\theta^t) - \mathcal{F}_T)^2 \tag{6}$$

There are several Monte Carlo methods to produce independent samples, such as direct simulation, rejection sampling and importance sampling. The following subsections describes such methods and are given by Gelman *et al.* (2014), Murray (2007) and Robert (2010). Here, we describe these methods to produce a random sample of size 1 and note that these can be repeated to draw larger samples.

3.2.1 Rejection Sampling

Suppose we want to obtain a single draw from a density, $\pi(\theta \mid x)$, in *rejection sampling* (also known as *acceptance-rejection sampling* (Lee, 2004, p.267)), we draw samples using another distribution, $g(\theta)$, from which we can already sample. We require $g(\theta)$ to be a positive function defined for all θ where $\pi(\theta \mid x) > 0$. The method also requires the ability to evaluate the probability density of points under both of the distributions up to proportionality. Moreover, the *importance ratio* is given by $\frac{\pi(\theta \mid x)}{g(\theta)}$ and it is required that it must be bounded above, that is, there is some constant M for which $\frac{\pi(\theta \mid x)}{g(\theta)} \leq M$ for all θ (this implies that $\pi(\theta \mid x) \leq Mg(\theta)$).

Then to obtain a random draw from $\pi(\theta|x)$, the rejection sampling algorithm is as follows:

1. Sample θ at random from the probability density proportional to $g(\theta)$.
2. Calculate the importance ratio, $\frac{\pi(\theta \mid x)}{Mg(\theta)}$.
3. Draw a uniform number, u, from $(0, 1)$.
4. If $u < \frac{\pi(\theta \mid x)}{Mg(\theta)}$, accept θ as a draw from $\pi(\theta \mid x)$, otherwise, reject θ as a draw and return to step 1.

Murray (2007, p.22) identifies that the more closely $g(\theta)$ matches $\pi(\theta \mid x)$, the lower the rejection rate can be made and in the ideal situation where g is proportional to π, then every draw is accepted.

To show how rejection sampling works, suppose that we want to obtain an independent sample for θ, where $\theta \sim$ Beta(4,2). Then for rejection sampling, we need to draw samples from another distribution, $g(\theta)$, where we can find a constant M such that $\pi(\theta) \leq Mg(\theta)$ for all θ. Here, we can use a uniform density between 0 and 1, so $g(\theta) = 1$ for $\theta \in (0, 1)$. Figure 2 shows a plot of the density for a Beta(4,2) distribution in red and $g(\theta)$ in blue.

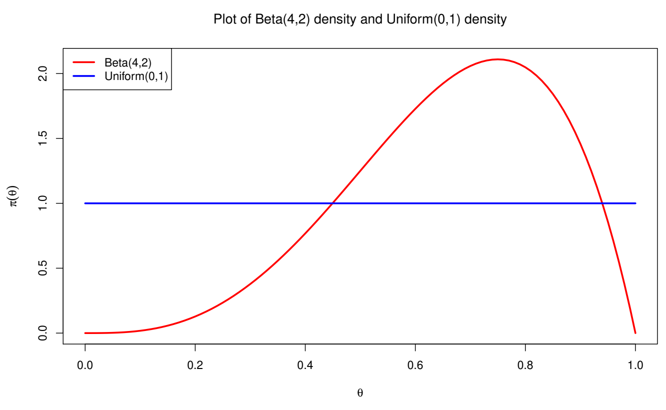


Figure 2: Plot of Beta(4,2) and Uniform(0,1) density

To use rejection sampling, we can use any M such that $\pi(\theta) \leq Mg(\theta)$ for all θ. In this case, M can be any value greater than or equal to 2.109375, since that is the mode of the Beta(4,2) distribution. Having $M = 2.109375$ will have the highest acceptance rate (which is calculated by using Number of samples / Number of iterations required). For instance, when $M = 2.109375$, then the acceptance rate was 0.481 to 3dp, whereas when $M = 100$, then the acceptance rate was 0.01 to 3dp and this is because the importance ratio is larger if M is smaller, meaning samples are accepted more often. However, using different values of M returned similar results - Figure 3 shows a histogram of the accepted θ values when $M = 100$. The histogram of accepted θ values when $M = 2.109375$ was very similar, so the choice of M only really affects the efficiency of the algorithm. In this plot, the density for the Beta(4,2) distribution is plotted in red for a comparison.

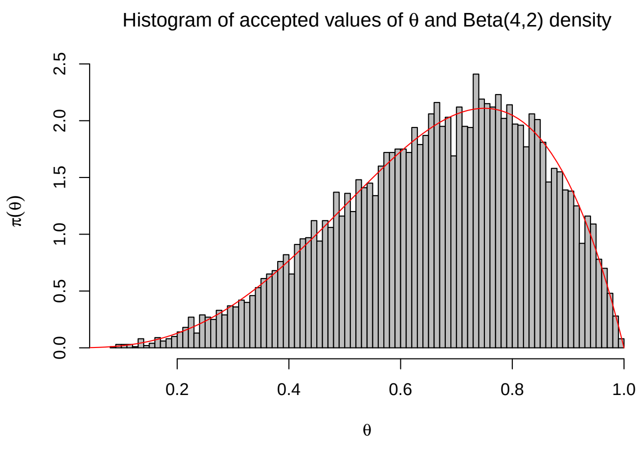


Figure 3: Histogram of accepted values of θ with $M = 100$

The R code used to implement rejection sampling for this example can be found in the Appendix (Section A).

3.2.2 Importance Sampling

Now suppose that we are interested in the posterior expectation of some function $f(\theta)$, but are unable to generate random draws of θ from $\pi(\theta \mid x)$, so we are unable to evaluate the integral by a simple average of simulated values, used in the Classical Monte Carlo approximation, then we can use importance sampling instead.

Consider a probability density $g(\theta)$ for which we can generate random draws from, then the posterior expectation of the function $f(\theta)$ can be written as

$$\begin{aligned} \mathrm{E}(f(\theta \mid x)) &= \frac{\int_\Theta f(\theta)\pi(\theta \mid x)d\theta}{\int_\Theta \pi(\theta \mid x)d\theta} \\ &= \frac{\int_\Theta [f(\theta)\pi(\theta \mid x)/g(\theta)]g(\theta)d\theta}{\int_\Theta [\pi(\theta \mid x)/g(\theta)]g(\theta)d\theta}. \end{aligned}$$

Then an approximation to this can be estimated using S draws $\theta^1, ..., \theta^T$ from $g(\theta)$ using

$$\begin{aligned} \mathrm{E}(f(\theta \mid x)) &= \frac{\frac{1}{T}\sum_{t=1}^T f(\theta^t)\omega(\theta^t)}{\frac{1}{T}\sum_{t=1}^T \omega(\theta^t)} \\ &= \frac{\sum_{t=1}^T f(\theta^t)\omega(\theta^t)}{\sum_{t=1}^T \omega(\theta^t)}, \end{aligned}$$

where the factors, $\omega(\theta^t) = \frac{\pi(\theta^t \mid x)}{g(\theta^t)}$, are known as the *importance ratios* or *importance weights.*

Murray (2007, p.24) states that:

Both rejection sampling and importance sampling require a tractable surrogate distribution $[g(\theta)]$. Neither method will perform well if $\max_\theta[\frac{\pi(\theta)}{g(\theta)}]$ is large: rejection sampling will rarely return samples and importance sampling will have large variance.

Hence, rather than using the methods described in this section, Markov Chain Monte Carlo methods can be used instead to sample from distributions that are more complex. One of these is the Metropolis-Hastings algorithm.

3.3 The Metropolis-Hastings algorithm

The Metropolis-Hastings algorithm was developed by Metropolis *et al.* (1953) and after was generalised by Hastings (1970). The algorithm provides a method for simulating a Markov chain whose stationary distribution is the specified target distribution, $\pi(\theta)$. We can use this algorithm to sample from any probability distribution provided that we can specify the density up to proportionality. We first present the Metropolis algorithm, then the Metropolis-Hastings algorithm, which generalises the basic Metropolis algorithm.

3.3.1 The Metropolis algorithm

The Metropolis algorithm also requires us to find an appropriate *proposal distribution* (or *jumping distribution*, which is used by Gelman *et al.* (2014)). In the Metropolis algorithm, the proposal distribution must be *symmetric*, which means that it satisfies the condition, $f(x, y) = f(y, x)$ or $f(x|y) = f(y|x)$. For example, consider the normal density evaluated at x with mean y and variance σ^2,

$$f(x \mid y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x-y)^2},$$

where $x, y \in \mathbb{R}$. Then the normal density is symmetric since,

$$f(x \mid y) = f(y \mid x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(y-x)^2}.$$

Then the Metropolis algorithm for a density $\pi(\theta|x)$ with a proposal density $q(\theta|\phi) = q(\phi|\theta)$ is initialised by setting the number of required samples, N, and starting point, θ_0, for which $\pi(\theta_0 \mid x) > 0$, then for $t = 1, 2, ..., N$,

1. Sample a proposal θ^* from the proposal distribution at time t, $q(\theta \mid \theta_{t-1})$.

2. Calculate the *acceptance ratio*,

$$A = \frac{\pi(\theta^* \mid x)}{\pi(\theta_{t-1} \mid x)}$$

3. Draw a uniform number, u, from $(0, 1)$.

4. If $u < A$, set $\theta_t = \theta^*$, otherwise set $\theta_t = \theta_{t-1}$

So when $\theta_t = \theta_{t-1}$, this means that the proposal is rejected, but it still counts as an iteration in the algorithm. This means that the choice of the proposal distribution is important in the Metropolis algorithm, since if the variance of the proposal distribution is too large, the chain may get stuck and will not explore the distribution well. But if the variance is too small, the chain may take very long to converge. The details of why the Metropolis algorithm works to provide sequence of iterations $\theta_1, \theta_2, ...$ that converges to the target distribution, $\pi(\theta \mid x)$, is given by Voss (2014, pp.114-115) and Gelman *et al.* (2014, p.279).

3.3.2 Discarding early iterations and effective sample size

If the chain is not sampling from the specified distribution efficiently, then it is said to be *mixing poorly.* For MCMC algorithms, the idea is to sample a Markov chain whose stationary distribution is the target distribution and so a problem with not discarding any early iterations is that at the start of the chain, the chain may not yet have reached this stationary distribution (Geyer, 1992, p.480). For the Metropolis and Metropolis-Hastings algorithms, it does not matter where the starting point is, given the proposal distribution has large enough variance and the number of iterations is large enough so the chain can explore the distribution well. To remove the influence of starting values, we generally discard a proportion of the first few values in the chain. The practice of discarding early iterations in the Markov chain simulation is referred to as the *warm-up* (Gelman *et al.*, 2014, p.282) and the number of iterations needed for the chain to converge is called the *burn-in period.*

Note that the sample is not an independent sample. The *effective sample size*, N^*, of the chain of length N can be estimated using:

$$N^* = \frac{N}{1 + 2\sum_{k=1}^{N-1} \rho_k},$$

where ρ_k is the estimated autocorrelation at lag k.

3.3.3 The Metropolis-Hastings algorithm

In the Metropolis algorithm it was required that an appropriate proposal distribution was found and the proposal density must be symmetric. Now for the Metropolis-Hastings algorithm, the proposal density, $q(\theta \mid \phi)$, need no longer be symmetric. Hence for the Metropolis-Hastings algorithm remains the same as the Metropolis except the acceptance ratio is now

$$A = \frac{\pi(\theta^* \mid x)q(\theta_{t-1} \mid \theta^*)}{\pi(\theta_{t-1} \mid x)q(\theta^* \mid \theta_{t-1})}.$$

To understand the Metropolis-Hastings algorithm and why it works, we first need to define the concept of *detailed balance*, which is an important property of certain Markov chains. Fishman (1996, p.341) gives the following definition: a Markov chain with state space $\mathcal{S}$ and transition probabilities p_{ij} satisfies the *detailed balance property* (with respect to $\boldsymbol{\pi}$) if

$$\pi_i p_{ij} = \pi_j p_{ji}, \text{ for all } i, j \in \mathcal{S},$$

If a Markov Chain satisfies the detailed balance condition, then it is called $\boldsymbol{\pi}$*-reversible.* Further, Voss (2014, p.114) notes that satisfying the detailed balance property is a stronger condition for a Markov chain than having a stationary distribution and proves that if a

Markov chain, X, is $\boldsymbol{\pi}$-*reversible*, the $\boldsymbol{\pi}$ is a stationary distribution of X. In addition, Voss (2014, pp.114-115) proves that the process X constructed in the Metropolis-Hastings algorithm is a $\boldsymbol{\pi}$-*reversible* Markov chain with stationary distribution $\boldsymbol{\pi}$.

Note that in this notation, we assume that the state space of X of the Markov chain and our desired target density is discrete, but the algorithm works also in continuous state spaces also (see Voss 2014, p.110).

3.3.4 An example of the Metropolis-Hastings algorithm in R

To illustrate the implementation of the Metropolis-Algorithm, we will use an example that can be found analytically so we can assess the accuracy of the algorithm. For this example, consider the following model

$$\begin{aligned} Y \mid \lambda &\sim \text{Poisson}(\lambda), \\ \lambda &\sim \text{Gamma}(2, 2). \end{aligned}$$

Since $\lambda \sim \text{Gamma}(2, 2)$, then

$$\pi(\lambda) \propto \lambda e^{-2\lambda}.$$

Additionally, consider that we have some data, $\boldsymbol{y} = (2, 8, 7, 1, 0)$, then as $\pi(\boldsymbol{y} \,|\, \lambda) \propto l(\lambda; \boldsymbol{y})$ and the likelihood here is

$$\begin{aligned} l(\lambda; \boldsymbol{y}) &= \frac{\lambda^2 e^{-\lambda}}{2!} \frac{\lambda^8 e^{-\lambda}}{8!} \frac{\lambda^7 e^{-\lambda}}{7!} \frac{\lambda^1 e^{-\lambda}}{1!} \frac{\lambda^0 e^{-\lambda}}{0!} \\ &\propto \lambda^{18} e^{-5\lambda}, \end{aligned}$$

then by Bayes's Theorem, the posterior density is

$$\begin{aligned} \pi(\lambda \mid \boldsymbol{y}) &\propto \pi(\boldsymbol{y} \mid \lambda)\pi(\lambda) \\ &\propto \lambda^{19} e^{-7\lambda}. \end{aligned}$$

In Bayesian analysis, a *conjugate analysis* is where the posterior density has the same form as the prior. Hence by conjugacy, the posterior distribution is given by

$$\lambda \,|\, \boldsymbol{y} \sim \text{Gamma}(20, 7),$$

and therefore, the posterior mean for λ is given by $\hat{\lambda} = \frac{20}{7} \approx 2.8571$ to 4dp (since the mean of a Gamma(α, β) distribution is given by $\frac{\alpha}{\beta}$).

The proposal density used in this example is chosen to be the Gamma$(4, 2)$ distribution, but note that this could be any proposal density in theory as long as the variance is appropriate and the chain runs long enough. The R code used to implement the Metropolis algorithm for this example can be found in the Appendix (Section B).

In order to use the Metropolis algorithm, we first sample a proposal λ^* from the proposal distribution and calculate the acceptance ratio. Here, the acceptance ratio for this example would be of the form,

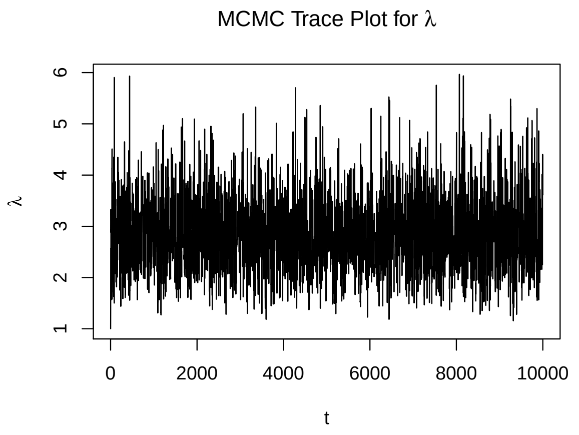
$$A = \frac{{\lambda^*}^{19} e^{-7\lambda^*} q(\lambda_{t-1})}{\lambda_{t-1}^{19} e^{-7\lambda_{t-1}} q(\lambda^*)}$$

where $q(\lambda)$ is the proposal density for λ, which is the Gamma$(4, 2)$ density here. Note that in order to avoid computational overflows or underflows, we can use logarithms of densities wherever it is possible and we should only use exponentials only when necessary. Hence, when implementing this Metropolis algorithm, when we draw a uniform number, u, from $(0, 1)$, we now compare $\log(u)$ with $\log(A)$ to decide whether or not to keep the proposed value of λ, where $\log(A)$ is given by

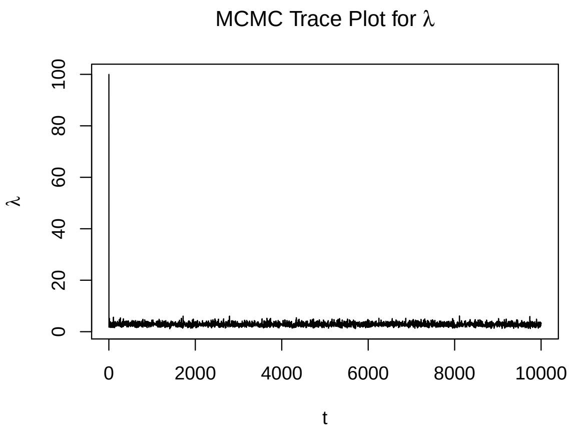
$$\log(A) = 19\log(\lambda^*) - 7\lambda^* + \log(q(\lambda_{t-1})) - (19\log(\lambda_{t-1}) - 7\lambda_{t-1} + \log(q(\lambda^*)))$$

Figure 4a shows the trace plot for the parameter λ, which shows the value of λ at each iteration of the algorithm. As we can see, this plot looks like random noise as it is sampling from the posterior density effectively. Here, we set the number of required samples, $N = 10000$, so by looking at the mean of the values between 1000 and 10000 (using 1000 as the burn-in period), then the posterior mean was found to be 2.853 (to 3dp), which is very close to the analytical solution. Note that the starting point does not matter for the Metropolis-Hastings algorithm, as long as the proposal distribution has large enough variance and the number of iterations (length of the chain) is long enough. For instance, in Figure 4b, we start the algorithm from 100, and we can see that it quickly moves downwards towards the correct value of λ. When the starting point was set to 100, the posterior mean calculated was 3.870 (to 3dp). By using the values of λ after discarding early iterations, we are able to obtain various summary statistics about λ, such as the posterior mean, posterior median and credible intervals.

To illustrate the importance of finding an appropriate proposal distribution, then suppose we use a folded normal distribution with mean λ_{t-1} for $t = 1, 2, ..., N$ and variance σ^2 that we will choose different values for (a folded normal distribution is just the absolute value of the regular distribution so only has positive values, since $\lambda > 0$ for a Poisson distribution).



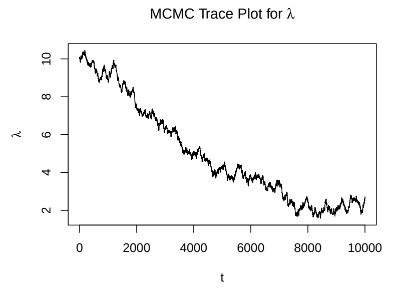
(a) Starting point set to 1



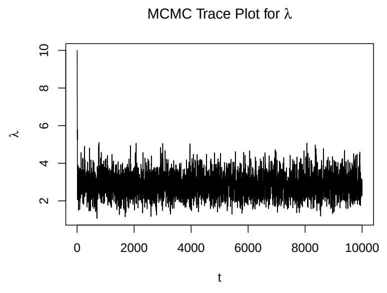
(b) Starting point set to 100

Figure 4: Metropolis-Hastings MCMC trace plots for λ with proposal distribution a Gamma$(4, 2)$ distribution

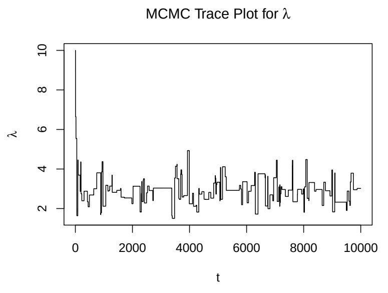
Figure 5 shows the different trace plots for λ for different values of σ^2 with all chains starting from at 10. In Figure 5a, $\sigma^2 = 0.001$ and we can see that the chain takes a long time ($\sim$ 7000 iterations) to reach the posterior distribution. Hence, from having a smaller variance for the proposal, then the warm-up period is longer here. If we only discard the first 1000 values, the posterior mean from this chain is 4.435 (to 3dp). However, if we set the warm-up period to 6000 then the posterior mean of values between 6000 and 10000 in the chain was 2.549 (to 3dp). While this is still not close enough to the true value of the posterior mean (2.8571), this is a better approximation by taking only the last 4000 values in the chain.



(a) $\sigma^2 = 0.001$



(b) $\sigma^2 = 1$



(c) $\sigma^2 = 10000$

Figure 5: Metropolis-Hastings MCMC trace plots for λ with proposal distribution a folded normal distribution with mean 0 and variance σ^2

Now, if we choose $\sigma^2 = 1$, then we can see from Figure 5b, the chain reaches the posterior distribution much quicker and the warm-up period is much shorter. If the variance is too large, as it is in Figure 5c, where $\sigma^2 = 10000$, the chain can get stuck at various points and the acceptance rate is much lower, as the proposal distribution will propose values that are far away from the previous value. Therefore, we can see that when the variance is too large, the chain does not mix well and it does not explore the posterior distribution effectively.

3.4 Hamiltonian Monte Carlo

The Stan modelling language (Stan Development Team, 2017, p.390) utilises the *Hamiltonian Monte Carlo* (HMC) algorithm and its adaptive variant, the *no-U-turn sampler* (NUTS) (see Gelman & Hoffman, 2011). Betancourt & Girolami (2013, p.2) state that although the Metropolis algorithm and the Gibbs sampler (not discussed here, but see Voss, 2014, p.141) are straightforward to implement in many models, their performance are limited by their incoherent exploration. This means that an inefficiency of these algorithms is their random walk behaviour, which can explore the target distribution extremely slowly. This can be seen in Figure 5a, where the chain can take a long time moving through the target distribution, since the variance of the proposal density was too small. The HMC algorithm is described by Gelman *et al.* (2014) and Betancourt and Girolami (2013) but is translated into the notation that we have been using so far and more steps are added where it is not immediately obvious where values or equations come from. Before presenting the algorithm, we first introduce some ideas from Hamiltonian dynamics.

3.4.1 Hamiltonian dynamics

Hamiltonian Monte Carlo uses Hamiltonian dynamics to suppress the local random walk behaviour exhibited in the Metropolis algorithms and this allows it to move much more rapidly through the target distribution. The Metropolis-Hastings algorithm presented in Section 3.3.3 is useful since it will always converge to the correct target distribution, given that the chain is long enough. However, as illustrated in the example shown in Section 3.3.4, if the variance of the proposal density is too small, the chain makes very small steps and takes a long time to explore the target distribution. But if the variance is too large, the chain may get stuck at points since there are more samples being rejected. Hence the issue with the Metropolis-Hastings algorithm is its random-walk behaviour and this is very important in high-dimensional cases. Suppose that the target distribution has 100 dimensions, then at each step the algorithm must choose a direction to go in. Intuitively, we can see that guessing a good direction in so many directions is very difficult and therefore sampling high-dimensional distributions using the Metropolis-Hastings algorithm becomes very inefficient.

In contrast, the methods in HMC are built upon a rich theoretical foundation that makes it uniquely suited to the high-dimensional problems of applied interest (Betancourt, 2017, p.3). However, since the theoretical underpinnings of HMC are formulated in terms of differential geometry, we will only present the general ideas and intuition of Hamiltonian dynamics and some reasons for the success of Hamiltonian Monte Carlo.

Neal (2011, p.114) gives the physical analogy of a hockey puck sliding on a frictionless surface of varying height to explain Hamiltonian dynamics. The state of this system consists of the *position* of the puck, given by a two-dimensional vector $\boldsymbol{\theta}$, say, and the *momentum* of the puck (mass $\times$ velocity), given by a two-dimensional vector $\boldsymbol{\phi}$. Moreover, the *potential energy* (the stored energy in an object due of its position or its configuration) of the puck is proportional to the height of the surface at the given position and the *kinetic energy* (the energy which the puck possesses because of its motion) is equal to $\frac{|\boldsymbol{\phi}|^2}{2m}$, where m is the mass of the puck. If the surface is level, then the puck moves at constant velocity, equal to the momentum divided by the mass, $\frac{\boldsymbol{\phi}}{m}$. If the puck encounters a rising slope, then the puck's momentum allows it to continue to move, with the kinetic energy decreasing and potential energy increasing until the point where the kinetic energy of the puck is equal to zero. At this point, the puck will slide back down and the kinetic energy increases and potential energy decreases.

It is noted by Betancourt (2017, p.3) that Hamiltonian Monte Carlo was initially called *Hybrid Monte Carlo* and the method was used to tackle calculations within Lattice field theory simulations of quantum chromodynamics, which is a field focused on understanding the structure of protons and neutrons. However, in non-physical applications of Hamiltonian Monte Carlo such as Bayesian analysis, in the above analogy, the position $\boldsymbol{\theta}$ of the puck corresponds to the variables of interest, $\boldsymbol{\theta} = (\theta_1, ..., \theta_d)$, where d is the number of parameters of interest. The potential energy will be minus the log probability density for these variables and a momentum variable, $\boldsymbol{\phi} = (\phi_1, ..., \phi_d)$, will be introduced to the problem artificially.

This analogy can be used to help understand the idea behind HMC, but these can also be understood by using a set of differential equations called *Hamilton's equations*.

3.4.2 Hamilton's equations

Suppose we have a d-dimensional *position* vector $\boldsymbol{\theta}$ and a d-dimensional *momentum* vector, $\boldsymbol{\phi}$, and so the full state space has $2d$ dimensions. A function of $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ that describes the system is known as the *Hamiltonian*, $H(\boldsymbol{\theta}, \boldsymbol{\phi})$. Then the partial derivatives of the Hamiltonian, H, determine how $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ change over time, t, according to *Hamilton's equations* (Neal, 2011, p.115):

$$\begin{aligned} \frac{d\theta_i}{dt} &= \frac{\partial H}{\partial \phi_i} \\ \frac{d\phi_i}{dt} &= -\frac{\partial H}{\partial \theta_i}, \end{aligned} \tag{7}$$

for $i = 1, ..., d$.

For Hamiltonian Monte Carlo, we usually use *Hamiltonian functions* (or the *Hamiltonian*) that can be written as

$$H(\boldsymbol{\theta}, \boldsymbol{\phi}) = T(\boldsymbol{\phi}) + V(\boldsymbol{\theta}),$$

where $T(\boldsymbol{\phi})$ is the kinetic energy, which is defined as minus the log probability density of the multivariate normal distribution with mean vector $\mathbf{0}$ and covariance matrix Σ and $V(\boldsymbol{\theta})$ is the potential energy, which will be defined as minus the log probability density of the distribution for $\boldsymbol{\theta}$ that we wish to sample from.

3.4.3 Hamiltonian Monte Carlo algorithm

Neal (2011, p.113) notes, Hamiltonian dynamics can be applied to most problems with continuous state spaces by simply introducing a 'momentum' variable. In HMC, for each component θ_j in the target space, a momentum variable, ϕ_j, is added and are both updated together in a Metropolis algorithm, where $j = 1, ..., d$. The proposal distribution for $\boldsymbol{\theta}$ is now largely determined by the momentum variable, $\boldsymbol{\phi}$. Hence, in HMC, we introduce an auxiliary momentum variable, $\boldsymbol{\phi}$, that is introduced only to enable the algorithm to move faster through the parameter space. The aim of HMC is to obtain draws from a density, $\pi(\boldsymbol{\theta})$, for parameters $\boldsymbol{\theta}$. In a Bayesian setting, this target density is usually the posterior density, $\pi(\boldsymbol{\theta} \mid x)$ given some data x. HMC draws from a joint density

$$\begin{aligned} \pi(\boldsymbol{\phi}, \boldsymbol{\theta} \mid x) &= \pi(\boldsymbol{\phi} \mid \boldsymbol{\theta}, x)\pi(\boldsymbol{\theta} \mid x) \\ &= \pi(\boldsymbol{\phi})\pi(\boldsymbol{\theta} \mid x), \text{ since } \boldsymbol{\phi} \text{ is independent to } \boldsymbol{\theta}. \end{aligned}$$

In this algorithm, we simulate from this joint distribution, but we are only interested in the simulations of $\boldsymbol{\theta}$.

In Stan and most applications of HMC, $\boldsymbol{\phi}$ is given a multivariate normal distribution with mean vector $\mathbf{0}$ and covariance matrix Σ ($\boldsymbol{\phi}$ has the same dimension as $\boldsymbol{\theta}$, since every parameter θ_j has a momentum variable ϕ_j for $j = 1, ..., d$),

$$\boldsymbol{\phi} \sim N_d(\mathbf{0}, \Sigma), \text{ where } d \text{ is the dimension of } \boldsymbol{\theta}.$$

In Stan, the covariance matrix Σ is commonly set to be diagonal or the identity matrix but it can be estimated from the warm-up samples.

The negative logarithm of the joint density for $\pi(\boldsymbol{\phi}, \boldsymbol{\theta} \mid x)$ defines a Hamiltonian (Betancourt and Girolami, 2013, p.4),

$$\begin{aligned}
H(\boldsymbol{\phi}, \boldsymbol{\theta} \mid x) &= -\log\, \pi(\boldsymbol{\phi}, \boldsymbol{\theta} \mid x) \\
&= -\log\, (\pi(\boldsymbol{\phi})\pi(\boldsymbol{\theta} \mid x)) \\
&= -\log\, \pi(\boldsymbol{\phi}) - \log\, \pi(\boldsymbol{\theta} \mid x) \\
&= T(\boldsymbol{\phi}) + V(\boldsymbol{\theta} \mid x),
\end{aligned}$$

where $T(\boldsymbol{\phi}) = -\log\, \pi(\boldsymbol{\phi})$ is the *kinetic energy* and $V(\boldsymbol{\theta} \mid x) = -\log\, \pi(\boldsymbol{\theta} \mid x)$ is the *potential energy* of the parameter (or particle/puck), $\boldsymbol{\theta}$.

A transition to a new state of parameters are generated in two steps before accepting or rejecting the new step using the Metropolis acceptance ratio, defined in Section 3.3.1. Firstly, we sample from the auxiliary momenta,

$$\boldsymbol{\phi} \sim N_d(\mathbf{0}, \Sigma).$$

Following this, current parameter values $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ evolve via *Hamilton's equations*, which are given in equation 7,

$$\begin{aligned}
\frac{d\boldsymbol{\theta}}{dt} &= \frac{\partial H}{\partial \boldsymbol{\phi}} = \frac{\partial T}{\partial \boldsymbol{\phi}}, \\
\frac{d\boldsymbol{\phi}}{dt} &= -\frac{\partial H}{\partial \boldsymbol{\theta}} = -\frac{\partial T}{\partial \boldsymbol{\theta}} - \frac{\partial V}{\partial \boldsymbol{\theta}}.
\end{aligned}$$

The momentum density for $\boldsymbol{\phi}$ is independent of the target density and so Hamilton's equations become,

$$\begin{aligned}
\frac{d\boldsymbol{\theta}}{dt} &= \frac{\partial T}{\partial \boldsymbol{\phi}}, \\
\frac{d\boldsymbol{\phi}}{dt} &= -\frac{\partial V}{\partial \boldsymbol{\theta}}.
\end{aligned}$$

This leaves a system of partial derivatives to be solved. For a computer implementation of this, Hamilton's equations must be approximated by discretising time (Neal, 2011, p,120) and by using some small stepsize, ε. In the Stan programming language, the numerical integration algorithm used to provide stable results for Hamiltonian systems of equations is the *leapfrog algorithm* (or *leapfrog integrator*) (Stan Development Team, 2017, p.392).

3.4.4 The Leapfrog algorithm

The leapfrog algorithm takes discrete steps of size ε and alternates half-step updates of the momentum vector, $\boldsymbol{\phi}$, and full-step updates of the position vector, $\boldsymbol{\theta}$:

$$\begin{aligned}\boldsymbol{\phi} &\leftarrow \boldsymbol{\phi} + \frac{\varepsilon}{2}\frac{d\boldsymbol{\phi}}{dt},\\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \varepsilon\frac{d\boldsymbol{\theta}}{dt},\\ \boldsymbol{\phi} &\leftarrow \boldsymbol{\phi} + \frac{\varepsilon}{2}\frac{d\boldsymbol{\phi}}{dt}.\end{aligned}$$

Suppose that a d-dimensional random variable X follows a multivariate normal distribution with mean vector $\boldsymbol{\mu}$ and covariance matrix Σ, then its joint probability density is given by Chatfield & Collins (1980, p.28) and is of the form,

$$f(\boldsymbol{x}) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}}\exp\Big[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\Big].$$

Then since $T(\boldsymbol{\phi}) = -\text{log}\,(\pi(\boldsymbol{\phi}))$ and $\boldsymbol{\phi} \sim N_d(\mathbf{0}, \Sigma)$, then

$$\pi(\boldsymbol{\phi}) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}}\exp\Big[-\frac{1}{2}\boldsymbol{\phi}^T\Sigma^{-1}\boldsymbol{\phi}\Big],$$

and so by taking logs and partially differentiating with respect to $\boldsymbol{\phi}$, we find that

$$\frac{d\boldsymbol{\theta}}{dt} = \frac{\partial T}{\partial\boldsymbol{\phi}} = \Sigma^{-1}\boldsymbol{\phi}.$$

Further, since $V(\boldsymbol{\theta} \mid x) = -\text{log}\,\pi(\boldsymbol{\theta} \mid x)$, then we just have

$$\frac{d\boldsymbol{\phi}}{dt} = -\frac{\partial V}{\partial\boldsymbol{\theta}} = \frac{d\text{log}\,\pi(\boldsymbol{\theta} \mid x)}{d\boldsymbol{\theta}}.$$

Therefore, the leapfrog algorithm takes discrete steps of some small time interval ε and updates the momentum and position using

$$\begin{aligned}\boldsymbol{\phi} &\leftarrow \boldsymbol{\phi} + \frac{\varepsilon}{2}\frac{d\text{log}\,\pi(\boldsymbol{\theta} \mid x)}{d\boldsymbol{\theta}},\\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \varepsilon\Sigma^{-1}\boldsymbol{\phi},\\ \boldsymbol{\phi} &\leftarrow \boldsymbol{\phi} + \frac{\varepsilon}{2}\frac{d\text{log}\,\pi(\boldsymbol{\theta} \mid x)}{d\boldsymbol{\theta}}.\end{aligned}$$

The algorithm is called a 'leapfrog', since it is splitting the momentum updates into half steps and is a discrete approximation to physical Hamiltonian dynamics where the position and momentum evolve in continuous time (Gelman *et al.*, 2014, p.302).

We then apply L leapfrog steps so a total of $L\varepsilon$ time is simulated. Now label the value of the momentum and parameter vectors at the start of the leapfrog process as $\boldsymbol{\theta}_{t-1}$, $\boldsymbol{\phi}_{t-1}$, then the state of the simulation after L such steps is denoted as $(\boldsymbol{\phi}^*, \boldsymbol{\theta}^*)$. In the accept-reject step, we compute the acceptance ratio (Neal, 2011, p.125):

$$A = \exp\left(-H(\boldsymbol{\phi^*}, \boldsymbol{\theta^*} \mid x) + H(\boldsymbol{\phi_{t-1}}, \boldsymbol{\theta_{t-1}} \mid x)\right)$$

Then since $H(\boldsymbol{\phi}, \boldsymbol{\theta} \mid x) = T(\boldsymbol{\phi}) + V(\boldsymbol{\theta} \mid x) = -\log\, \pi(\boldsymbol{\phi}) - \log\, \pi(\boldsymbol{\theta} \mid x)$, then

$$\begin{aligned} A &= \exp\Big(- T(\boldsymbol{\phi^*}) - V(\boldsymbol{\theta^*} \mid x) + T(\boldsymbol{\phi_{t-1}}) + V(\boldsymbol{\theta_{t-1}} \mid x)\Big) \\ &= \exp\Big(\log\, \pi(\boldsymbol{\phi^*}) + \log\, \pi(\boldsymbol{\theta^*} \mid x) - \log\, \pi(\boldsymbol{\phi_{t-1}}) - \log\, \pi(\boldsymbol{\theta_{t-1}} \mid x)\Big) \end{aligned}$$

Then by rearrangement, the acceptance ratio for the HMC algorithm is given by

$$A = \frac{\pi(\boldsymbol{\theta}^* \mid x)\pi(\boldsymbol{\phi}^*)}{\pi(\boldsymbol{\theta}_{t-1} \mid x)\pi(\boldsymbol{\phi}_{t-1})},$$

where π is the density for the momentum vector $\boldsymbol{\phi}$. Therefore, in this algorithm, by using Hamiltonian dynamics, we propose a new state for the momentum and position, $(\boldsymbol{\phi^*}, \boldsymbol{\theta^*})$,

Then as we had in the Metropolis and Metropolis-Hastings algorithm in Section 3.3, then we draw a uniform number, u, from $(0, 1)$ and if $u < A$, we set the next value in the iteration, $\boldsymbol{\theta}_t = \boldsymbol{\theta}^*$, otherwise we set $\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1}$. Therefore, since we do not care about the momentum $\boldsymbol{\phi}$ itself, it gets immediately updated at the beginning of the next iteration so there is no need to keep track of the values of $\boldsymbol{\phi}$ at each step.

3.4.5 Summary of the Hamiltonian Monte Carlo algorithm

Hamiltonian Monte Carlo is an MCMC algorithm that makes use of gradient information to avoid the random walk behaviour exhibited in other popular MCMC algorithms such as the Metropolis-Hastings algorithm. The method is based on ideas from Hamiltonian dynamics, which allows the chain to move quicker towards the target distribution.

To summarise, the Hamiltonian Monte Carlo algorithm is initialised by setting the number of required samples, N, and starts with a specified initial set of parameters $\boldsymbol{\theta_0}$. In Stan, this value can be specified by the user or is generated randomly. Then for $t = 1, 2, ..., N$, the algorithm is as follows:

1. Sample a new momentum vector from $\boldsymbol{\phi} \sim N_d(\mathbf{0}, \Sigma)$.

2. Update the current value of the parameters, $\boldsymbol{\theta}_{t-1}$ and $\boldsymbol{\phi}_{t-1}$, is updated using the leapfrog algorithm with discretisation time ε and number of steps L using:

$$\begin{aligned}
&\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} + \frac{\varepsilon}{2}\frac{d\log\,\pi(\boldsymbol{\theta} \mid x)}{d\boldsymbol{\theta}},\\
&\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \varepsilon\Sigma^{-1}\boldsymbol{\phi},\\
&\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} + \frac{\varepsilon}{2}\frac{d\log\,\pi(\boldsymbol{\theta} \mid x)}{d\boldsymbol{\theta}}.
\end{aligned}$$

3. Label the value of the parameters after L leapfrog steps as $\boldsymbol{\theta}^*$ and $\boldsymbol{\phi}^*$.

4. Calculate the acceptance ratio,

$$A = \frac{\pi(\boldsymbol{\theta}^* \mid x)\pi(\boldsymbol{\phi}^*)}{\pi(\boldsymbol{\theta}_{t-1} \mid x)\pi(\boldsymbol{\phi}_{t-1})},$$

5. Draw a uniform number, u, from $(0, 1)$.

6. If $u < A$, set $\boldsymbol{\theta}_t = \boldsymbol{\theta}^*$, otherwise set $\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1}$

3.4.6 Tuning parameters in HMC and Adaptive HMC

The Hamiltonian Monte Carlo algorithm can be tuned in three places:

1. The probability distribution of the momentum vector $\boldsymbol{\phi}$ (usually assumed to be multivariate normal),

2. The scaling factor, ε, of the leapfrog steps,

3. The number of leapfrog steps, L.

The sampling efficiency of the algorithm can be very sensitive to these tuning parameters (for more information about these tuning parameters and the effect they have on HMC, see Hoffman & Gelman, 2011; Gelman *et al.*, 2014; Neal, 2011).

Stan is able to automatically optimise ε, estimate Σ based on the warm-up sample iterations and adapt the number of leapfrog steps, L, at each iteration (Stan Development Team, 2017, p.394). Therefore, rather than fixing the number of steps L, the number of leapfrog steps can be adapted at each iteration. This is done by using the *no-U-turn sampling* (NUTS) algorithm (for more details about NUTS, see Gelman & Hoffman, 2011).

3.4.7 An example of the Hamiltonian Monte Carlo algorithm in R

We now illustrate an example of Hamiltonian Monte Carlo in R (see Appendix, Section C for R code) and apply it to the model discussed in Section 3.3.4. Consider the following model,

$$\begin{aligned} Y \mid \lambda &\sim \text{Poisson}(\lambda), \\ \lambda &\sim \text{Gamma}(2, 2), \end{aligned}$$

and consider we have some observed data, $\boldsymbol{y} = (2, 8, 7, 1, 0)$. Then as we found in Section 3.3.4, by conjugacy, the posterior distribution is given by

$$\lambda \mid \boldsymbol{y} \sim \text{Gamma}(20, 7),$$

and the posterior mean for λ is given by $\hat{\lambda} = \frac{20}{7} \approx 2.8571$ to 4dp.

Since this is only a one-dimensional case, then distribution of the momentum vector, ϕ, is just the univariate normal distribution with mean 0 and variance σ^2. To initiate the Hamiltonian Monte Carlo algorithm, we need to choose values for ε, L and σ^2. As noted in Section 3.4.6, in the Stan modelling language, these parameters are automatically optimised and the number of leapfrog steps L is adapted at each iteration by the use of the NUTS algorithm. However, for this implementation, a fixed L and ε is chosen for simplicity.

Recall for HMC, the momentum and the values of the parameters are updated using the leapfrog algorithm, which requires the calculation of the derivative of the log posterior density and the inverse of the covariance matrix Σ. In this one-dimensional case, the inverse is elementary and is just $\frac{1}{\sigma^2}$. For the derivative of the log posterior density, recall the posterior density is given by

$$\pi(\lambda \mid \boldsymbol{y}) \propto \lambda^{19} e^{-7\lambda}.$$

Then by taking logarithms and differentiating with respect to λ, we find that

$$\frac{d\pi(\lambda \mid \boldsymbol{y})}{d\lambda} = \frac{19}{\lambda} - 7\lambda.$$

Hence the leapfrog steps to update the momentum and variable λ in this example are

$$\begin{aligned}
&\phi \leftarrow \phi + \frac{\varepsilon}{2}\Big(\frac{19}{\lambda} - 7\lambda\Big),\\
&\theta \leftarrow \theta + \frac{\varepsilon}{\sigma^2}\phi,\\
&\phi \leftarrow \phi + \frac{\varepsilon}{2}\Big(\frac{19}{\lambda} - 7\lambda\Big).
\end{aligned}$$

Next in the HMC algorithm, we need to calculate the acceptance ratio, which in this case is given by

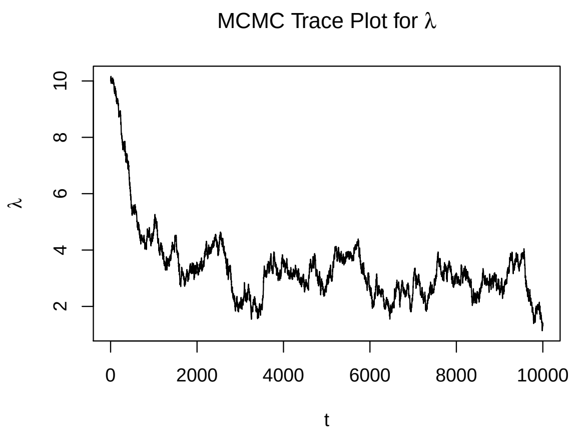
$$A = \frac{{\lambda^*}^{19} e^{-7\lambda^*} q(\phi^*)}{\lambda_{t-1}^{19} e^{-7\lambda_{t-1}} q(\phi_{t-1})},$$

where q is the density for a $N(0, \sigma^2)$ distribution. In implementing the algorithm, we use the log densities to calculate the acceptance ratios. Therefore, when deciding whether or not keep the new proposed value of λ and making a single draw from a Uniform(0,1) distribution to get u, we compare $\log(u)$ with $\log(A)$, which is given by

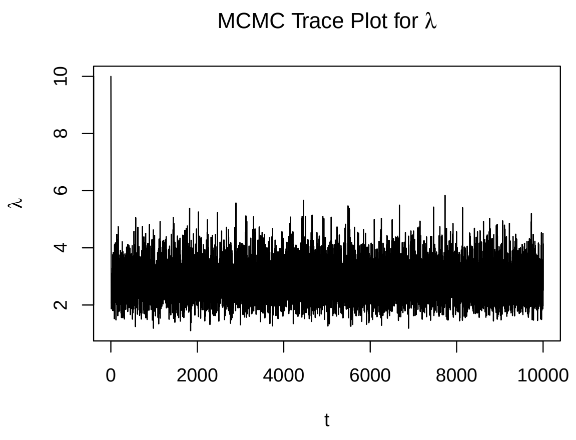
$$\log(A) = 19\log(\lambda^*) - 7\lambda^* + \log(q(\phi^*)) - (19\log(\lambda_{t-1}) - 7\lambda_{t-1} + \log(q(\phi_{t-1}))) \qquad (8)$$

Figure 6 shows the trace plots for λ when using the HMC algorithm to obtain a posterior sample for λ if we initially start from $\lambda = 10$. Here, we set $\sigma^2 = 100$ and $\varepsilon = 0.5$ and choose different values for L. If $L = 1$, meaning there is only one leapfrog step at each iteration, then we can see from 6a that the chain converges slower to the target distribution, since there were not enough leapfrog steps for λ to move through the parameter space. However, if L is increased to 100, as in Figure 6b, then λ reaches the target distribution much quicker and explores the posterior distribution well.

Since this is only a one-dimensional example to illustrate how the Metropolis-Hastings algorithm (see 3.3.4) and HMC algorithm can be implemented, we cannot clearly see the advantages of HMC over the Metropolis-Hastings algorithm here. However, in higher dimensions, HMC allows for the parameters of interest, $\boldsymbol{\theta}$, to move much quicker throughout the parameter space. While in some MCMC methods such as the Gibbs sampler (not discussed here) and the Metropolis-Hastings algorithm, where the simulations can take a long time to reach the target distribution, HMC suppresses the random walk behaviour in order to allow it to move quicker to the target distribution by using ideas from physics and adding a momentum variable. The random walk behaviour in Figure 6a is due to the small number of leapfrog steps and because the example is only in one dimension, which limits the effect of introducing the momentum variable.



(a) Number of leapfrog steps, $L = 1$



(b) Number of leapfrog steps, $L = 100$

Figure 6: HMC trace plots for λ with $\varepsilon = 0.5$, $\sigma^2 = 100$ and different number of leapfrog steps

4 Modelling football scores

The statistical modelling of sports has become increasingly popular and more data is now being collected in sports. Companies such as Opta and Prozone deliver data specifically designed to help with the coaching and scouting of players (Anderson & Sally, 2014). Maher (1982) used an independent Poisson distribution to model goals and estimated averages for each team to predict football scores on a game by game basis. He used maximum likelihood estimation to find estimates for the attack and defence parameters for the home and away team, which would contribute to the mean of the Poisson distribution. Maher also noted the benefits of a bivariate Poisson model, which he showed had a better fit to goals data for the English Division 1 between 1971 and 1974. Several researchers such as Lee (1997) and Karlis & Ntzoufras (2000) have shown relatively low correlation between the number of goals scored by the two opponents. However, Karlis & Ntzoufras (2003) note the use of bivariate distributions has been ignored in most modelling approaches, as it demands more sophisticated techniques and they then proposed to use a bivariate Poisson distribution to model goals scored. Further, Dixon and Coles (1997) also decided to use the Poisson distribution, although they did not assume independence of goals scored by the home and away team. Additionally, their model did not assume that the team's performance is constant over time as with Maher's model (1982) by introducing a time-decay function and constructed a 'pseudolikelihood' so that historical information had less value than more recent information. This helped to address the dynamic nature of each team's abilities and incorporated some information about the form of a team.

Most of the above work mentioned follow a frequentist framework, where they treat unknown quantities as fixed that are to be estimated. However, Karlis & Ntzoufras (2007) worked on this problem further by using the Skellam's distribution (the difference of two independent Poisson random variables with different means) and used Bayesian and Markov Chain Monte Carlo (MCMC) methods to obtain predictive distributions of goal differences. Baio & Blangiardo (2010) proposed a Bayesian hierarchical model for the number of goals scored by two teams in a match and used the Poisson distribution to model the goals scored for each team. This model will be looked more closely in Section 5 and is implemented in the Stan modelling language to assess the model and to have a short analysis of the 2016/17 season. Further, Suzuki *et al.* (2010) used a Bayesian approach for predicting match outcomes of 2006 World Cup games.

All of the models discussed briefly above rely on the assumption that the marginal distributions for goals for the home and away teams are Poisson distributed and they attempt to estimate the expected number of goals scored by the two opposing teams. For instance, models developed by Maher (1982), Dixon and Coles (1997), Karlis & Ntzoufras (2003) and Baio & Blangiardo (2010) all have used the Poisson distribution to model the goals scored by both teams. However, the Poisson distribution might be inadequate to model goals well and improvements can be made, and in Section 4.1, we investigate this assumption further.

It is important to note, while some researchers have decided to use a bivariate model, as Karlis & Ntzoufras (2003) did, others have used an independent model. However, it seems plausible that a bivariate model is more appropriate to account for the dependency between the two teams scoring output, since when one team is winning, we can see in many situations, the opposing team will attack more in attempt to get back in the game. As briefly noted in Section 2.6, since we will be using a Bayesian hierarchical model, there is no need to use a bivariate distribution, since there will be *conditional independence* between the variables for the number of goals scored by the home and away team. We will see in Sections 5 and 6, the hierarchical structure of the model developed Baio & Blangiardo (2010) and the Negative Binomial model that we develop imply a form of correlation between the goals scored by the home and away team.

All code to assess distributional fits can be found in the Appendix (Section D and E).

4.1 Poisson or not Poisson?

A histogram of goals scored by the home and away team in a game for the last 10 full seasons of the Premier League (from 2007/08 to 2016/17) is shown in Figure 7. The blue bars show the Poisson distribution probability values for each goal, where the parameter λ is the mean number of home or away goals per game (the maximum likelihood estimate for λ). In this section, we use maximum likelihood estimation for parameters, since we are just using this as an exploratory analysis to determine a suitable distribution to model the number of goals scored by the home and away teams.

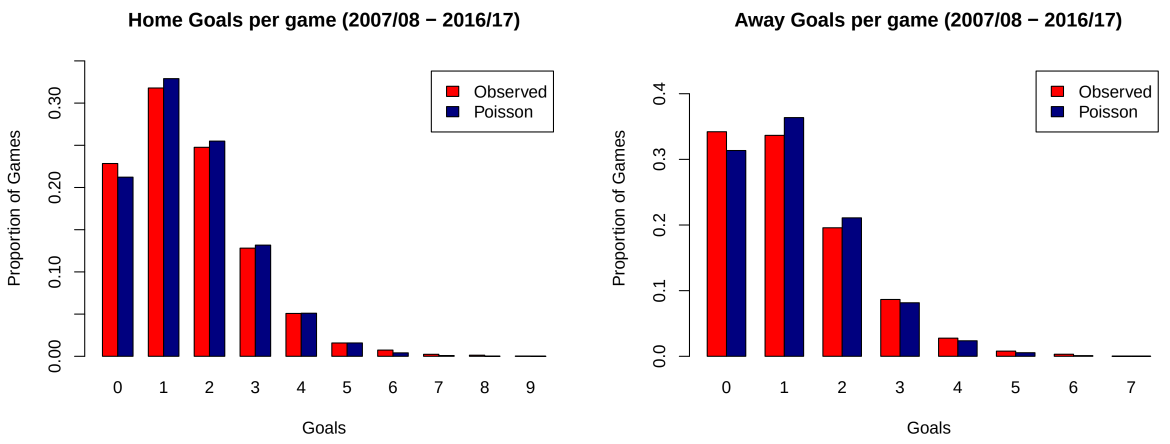


Figure 7: Histogram of number of home (left) and away (right) goals per game

To assess the distributional fit, in frequentist statistics, hypothesis tests are generally used and here we will use this approach since it is a standard assessment process and they can be useful as a tool for exploratory analysis. Although we will not be using them to accept or reject a null hypothesis, we can use the test statistics and conclusions from the tests for comparison between different distributions. One such test is the Chi-squared ($\mathcal{X}^2$) goodness of fit test. The $\mathcal{X}^2$ test statistic is calculated using the following formula

$$\mathcal{X}_t^2 = \sum_{k=1}^{n} \frac{(O_k - E_k)^2}{E_k},$$

where where O_k are the observed values from the data, E_k are the expected values from the proposed distribution (in this case, the Poisson distribution) and n is the number of observations. In such tests, a null hypothesis is proposed. For the distribution of goals scored by the home team, consider the null hypothesis:

$\boldsymbol{H_0}$: Dist. of Home Goals (07/08 - 16/17) follows a Poisson($\lambda = 1.55$) distribution,
$\boldsymbol{H_1}$: Dist. of Home Goals (07/08 - 16/17) does not follow a Poisson($\lambda = 1.55$) distribution.

In this test, assuming the null hypothesis true, the $\mathcal{X}^2$ test statistic follows a $\mathcal{X}^2$ distribution, with d degrees of freedom, where d = number of groups - number of estimated parameters. We then obtain a p-value, which is defined as the probability under the null hypothesis of obtaining a result equal to or more extreme than what was actually observed (Dahiru, 2008, p.22). However, it is important to note that these goodness of fit tests are designed to test if the proposed distribution is true and therefore almost always fail with enough data, as George Box notes, "all models are wrong" (1976, p.792). Further, Lin *et al.* (2013, p.906) states "a key issue with applying small-sample statistical inference [such as the $\mathcal{X}^2$ test] to large samples is that even minuscule effects can become statistical significant" - this is known as the *p-value problem*, according to Lin *et al.* (2013). To illustrate, consider the above hypothesis, then since there are 10 groups (0, 1,..., 9), then the degrees of freedom is $d = 10 - 1 = 9$ and the p-value obtained was 1.049×10^{-9}.

Hence, due to the extremely low p-value, we reject the null hypothesis that the distribution of home goals follows a Poisson($\lambda = 1.55$) distribution. The reason for the large $\mathcal{X}_t^2$ test statistic obtained and resulting low p-value is that we have a large number of games. Table 1 shows the expected number of home goals under a Poisson distribution where the parameter λ is estimated by the mean number of home goals, which is 1.55 (to 3sf), and the observed number of home goals. From Table 1, we can see these are quite similar. Additionally, the expected probabilities from the Poisson distribution fall within 3 standard errors about the observed probabilities. However, we can see that a Poisson distribution underestimates the number of games where the home team does not score a goal.

Note that we estimated the standard errors (SE) using the following formula,

$$\text{SE}(p) \approx \sqrt{\frac{1}{n-1}(p - p^2)},$$

where p is the observed proportion of home goals per game. To find this equation, we use the Monte Carlo estimate for the variance, shown in equation 6. In this case, we set $f(x) = I(x)$, where $I(x)$ is the indicator function for the events of interest, A say, i.e. the function for goals in a game. The *indicator function* is defined as

$$I(x) = \begin{cases} 1, & \text{if } x \in A \\ 0, & \text{if } x \notin A \end{cases}$$

Then note that $E[I(x)^2] = E[I(x)] = \Pr(A)$, and if we denote the observed proportion using p, then $\text{Var}(p) \approx \frac{1}{n-1}(p - p^2)$.

Home Goals	Exp. Prob	Obs. Prob	Exp. Freq	Obs. Freq	Obs. + 3SE	Obs. - 3SE
0	0.211	0.228	803.8	868	0.249	0.208
1	0.329	0.318	1248.6	1208	0.341	0.295
2	0.255	0.248	969.8	941	0.269	0.227
3	0.132	0.128	502.2	487	0.144	0.112
4	0.051	0.051	195.0	193	0.061	0.040
5	0.016	0.016	60.6	60	0.022	0.010
6	0.004	0.007	15.7	28	0.012	0.003
7	0.001	0.002	3.5	9	0.005	0.000
8	0.0002	0.0013	0.7	5	0.0031	-0.0005
9	0.00003	0.00026	0.1	1	0.00105	-0.00052

Table 1: Table of the expected and observed probabilities and frequencies of home goals scored (2007/08 - 2016/17) under a Poisson($\lambda = 1.55$) distribution

However, if we repeat this test with just one season to evaluate the fit of a Poisson distribution to model the goals scored by the home team, the goodness of fit test suggests that the Poisson distribution is a good fit to the data. Figure 8 shows histograms of home and away goals scored for the 2016/17 Premier League season, where the blue bars show the Poisson probability values for each goal, where the parameter λ is the mean number of home or away goals per game.

For the marginal home goals for the 2016/17 season, the mean number of home goals were 1.60 (to 3sf), and so consider a goodness of fit test with the null hypothesis:

$\boldsymbol{H_0}$: Dist. of Home Goals (16/17) follows a Poisson($\lambda = 1.60$) distribution,
$\boldsymbol{H_1}$: Dist. of Home Goals (16/17) does not follow a Poisson($\lambda = 1.60$) distribution.

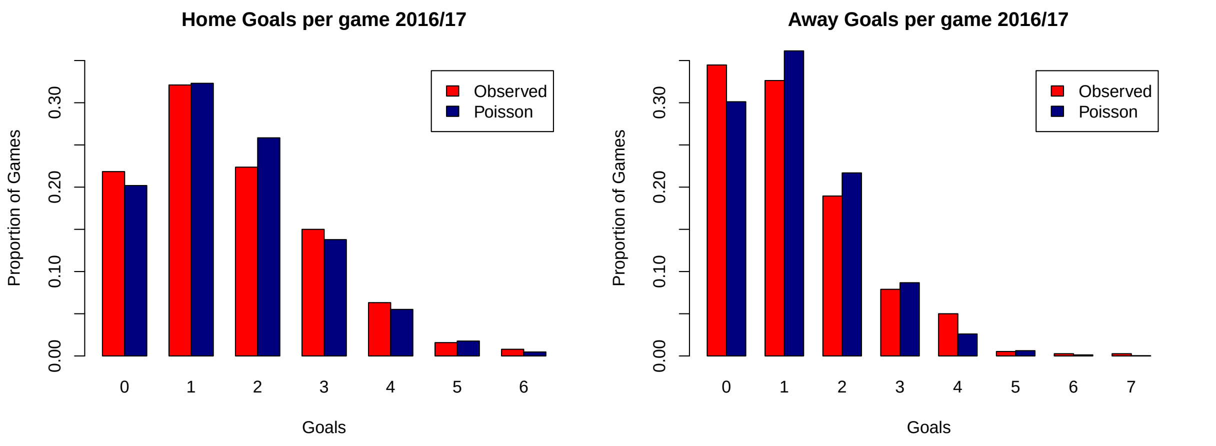


Figure 8: Histogram of number of home (left) and away (right) goals per game for 2016/17 season

Table 2 shows the expected number of goals under a Poisson($\lambda = 1.60$) distributon and the observed home goals scored in the 2016/17 season. Then the $\mathcal{X}^2$ test statistic calculated was $\mathcal{X}_t^2 = 4.0528$, with $d = 7-1 = 6$ degrees of freedom and so the p-value obtained was 0.6695. Hence, as $0.6695 > 0.10$, we do not have enough evidence to reject the null hypothesis at the 10% level. Hence, when only one season was used to evaluate the fit of a Poisson distribution to the data, the null hypothesis was not rejected, whereas it was rejected when more data were used. As stated above, $\mathcal{X}^2$ tests will fail with enough data, since a model will never be perfect and therefore, we must be careful to not use the goodness of fit test results solely to determine which distribution we wish to use. However, for exploratory analysis, we can still use these tests to give us some indication of how well the distribution fits to the data before we start to build a model. Using these with the help of visual representation of the distributional fits, such as the ones in Figures 7 and 8, can help us decide which distribution to use to model the number of goals scored by the home and away team.

The summary of the results of the $\mathcal{X}^2$ goodness of fit test of a Poisson distribution to home goals scored in the 2016/17 season and the last 10 full seasons are summarised in Table 3.

Now we focus our attention on the distribution of away goals. The mean number of goals scored by the away team per game for the last 10 seasons between 2007/08 and 2016/17 was 1.16 (to 3sf) and so consider the null hypothesis:

$\boldsymbol{H_0}$: Dist. of Away Goals (07/08 - 16/17) follows a Poisson($\lambda = 1.16$) distribution,
$\boldsymbol{H_1}$: Dist. of Away Goals (07/08 - 16/17) does not follow a Poisson($\lambda = 1.16$) distribution.

Home Goals	Exp. Prob	Obs. Prob	Exp. Freq	Obs. Freq	Obs. + 3SE	Obs. - 3SE
0	0.203	0.218	77.0	83	0.282	0.155
1	0.324	0.321	123.0	122	0.393	0.249
2	0.259	0.224	98.3	85	0.288	0.159
3	0.138	0.150	52.3	57	0.205	0.095
4	0.055	0.063	20.9	24	0.101	0.026
5	0.018	0.016	6.7	6	0.035	-0.003
6	0.005	0.008	1.8	3	0.022	-0.006

Table 2: Table of the expected and observed probabilities and frequencies of home goals under a Poisson($\lambda = 1.60$) distribution for the 2016/17 Premier League season

Data	χ^2-Statistic	df	p-value	Outcome
2007/08 - 2016/17 Seasons	60.552	9	1.049×10^{-9}	Reject at the 0.01% level
2016/17 Season	4.0528	6	0.6695	Do not reject at the 10% level

Table 3: Values of the χ^2 test statistic for the home goals scored for the Poisson(λ) model

Table 4 shows the expected number of home goals under a Poisson distribution($\lambda = 1.16$) and the observed number of away goals scored in the last 10 full seasons of the Premier League. The χ^2 test statistic is calculated as $\chi^2_t = 45.46$ with $d = 8 - 1 = 7$ degrees of freedom and the p-value obtained was 1.112×10^{-6}. Hence, we reject the null hypothesis, H_0, at the 0.01% level. As with the model for the home goals, the Poisson distribution underestimates the number of games for which the away team does scores no goals.

Now consider the marginal distribution of away goals scored in games for the 2016/17 season and consider a goodness of fit test with the following null hypothesis:

$\boldsymbol{H_0}$: Dist. of Away Goals (16/17) follows a Poisson($\lambda = 1.20$) distribution,
$\boldsymbol{H_1}$: Dist. of Away Goals (16/17) does not follow a Poisson($\lambda = 1.20$) distribution.

Then the χ^2 test statistic calculated was $\chi^2_t = 24.487$, with 7 degrees of freedom and so the p-value obtained was 0.0009. Hence, we reject this null hypothesis at the 0.01% level. Hence, although we cannot reject the hypothesis that the home goals for the 2016/17 season follows a Poisson distribution, the χ^2 test suggests that we should reject the hypothesis that the away goals scored in the 2016/17 season follow a Poisson distribution.

Further, again we can see visually from the histogram of the number of home and away goals per game shown in Figure 8 that the Poisson distribution does not have a great fit to the data, especially for the away goals, where it underestimates the number of games where the away team scores no goals and overestimates the number of games where the away team scores 1 or 2 goals. Table 6 summarises the χ^2 goodness of fit test results for testing the fit of the Poisson distribution to model the number of away goals scored.

Away Goals	Exp. Prob	Obs. Prob	Exp. Freq	Obs. Freq	Obs. + 3SD	Obs. - 3SD
0	0.314	0.342	1192.9	1300	0.365	0.319
1	0.364	0.337	1382.1	1279	0.360	0.314
2	0.211	0.196	800.7	744	0.215	0.176
3	0.081	0.087	309.3	329	0.100	0.073
4	0.024	0.028	89.6	105	0.036	0.020
5	0.005	0.008	20.8	30	0.012	0.004
6	0.0011	0.0032	4.0	12	0.0059	0.0004
7	0.00017	0.00026	0.7	1	0.00105	-0.00052

Table 4: Table of the expected and observed probabilities and frequencies of away goals scored (2007/08 - 2016/17) under a Poisson($\lambda = 1.16$) distribution

Away Goals	Exp. Prob	Obs. Prob	Exp. Freq	Obs. Freq	Obs. + 3SD	Obs. - 3SD
0	0.300	0.344	114.2	131	0.418	0.272
1	0.361	0.326	137.3	124	0.399	0.254
2	0.217	0.189	82.6	72	0.250	0.129
3	0.087	0.079	33.1	30	0.121	0.037
4	0.026	0.050	10.0	19	0.084	0.016
5	0.006	0.005	2.4	2	0.016	-0.006
6	0.001	0.002	0.5	1	0.011	-0.005
7	0.0002	0.0026	0.1	1	0.011	-0.005

Table 5: Table of the expected and observed probabilities and frequencies of home goals under a Poisson($\lambda = 1.20$) distribution for the 2016/17 Premier League season

Data	χ^2-Statistic	df	p-value	Outcome
2007/08 - 2016/17 Seasons	45.46	7	1.112×10^{-6}	Reject at the 0.01% level
2016/17 Season	24.487	7	0.0009	Reject at the 0.01% level

Table 6: Values of the χ^2 test statistic for the away goals scored for the Poisson(λ) model

Hence, from what we have seen in this section, the general assumption made by many researchers that goals follow a Poisson distribution seems inadequate to properly model the distribution of goals. It is possible that the reason for this is that in the models proposed, it is more 'natural' to think of parameters, such as the attack and defence estimates for a team contributing to the parameter λ, which is the mean, and maximum likelihood estimates are much easier to obtain for a Poisson model. In comparison, for other distributions, such as the negative binomial, which takes two parameters, the size n and probability p, it can be that these are much harder to think about and to estimate these. As Karlis & Ntzoufras (2000) note, "given the complicated nature of the negative binomial distribution and especially the difficulty in estimating parameters, it is plausible to use the simpler Poisson model".

In the following subsections, we look at other possible distributions to model home and away goals per game, such as the zero-inflated Poisson distribution, the negative binomial distribution and we also propose a Geometric-Poisson mixture distribution.

4.2 Alternative distributions for goals scored in football

4.2.1 Zero-Inflated Poisson distribution

The *Zero-Inflated Poisson* model is made from two components that correspond to two zero generating processes. Hall (2000, p.1032) defines the Zero-Inflated Poisson model as

$$Y \sim \begin{cases} 0 & \text{with probability } \omega, \\ \text{Poisson}(\lambda) & \text{with probability } (1-\omega). \end{cases}$$

Hence Y has a probability distribution function

$$\Pr(Y=y) = \begin{cases} \omega + (1-\omega)\exp(-\lambda) & \text{if } y = 0, \\ (1-\omega)\frac{\lambda^y \exp(-\lambda)}{y!} & \text{if } y > 0. \end{cases}$$

If a random variable Y follows a Zero-Inflated Poisson distribution with parameters ω and λ, then we write $Y \sim \text{ZIP}(\omega, \lambda)$.

In Section 4.1, we highlighted that a problem with using a Poisson distribution to model the goals scored by the home or away team in the Premier League was that it seemed to underestimate the number of games that resulted in the home or away team scoring no goals. Hence, we test the use of the Zero-Inflated Poisson distribution to see if by inflating the probability of the event of no goals will result in a better fit to the data. Here, ω is the probability of extra zeros.

Mean and Variance

The mean and variance for the Zero-Inflated Poisson distribution are given by

$$\begin{aligned} \text{E}[Y] &= (1-\omega)\lambda, \\ \text{Var}[Y] &= \lambda(1-\omega)(1+\omega\lambda). \end{aligned}$$

Maximum Likelihood Estimation for ZIP Distribution

The of the maximum likelihood estimates for ω and λ are given by

$$\hat{\omega} = \frac{r - e^{-\lambda}}{n(1 - e^{-\lambda})},$$
$$(1 - e^{-\lambda}) \sum_{i=0}^{n} y_i = \hat{\lambda}(n - r)$$

The derivation of these results can be found in the Appendix in Section F. Note here that the maximum likelihood estimate for λ requires numerical methods to solve.

To assess the fit of a zero-inflated Poisson distribution to home goals and away goals scored in the Premier League, we use the `Rfast` package (Papadakis et *al.*, 2017) in R, where it uses a Newton-Raphson algorithm to maximise the log-likelihood. We use the `Rfast` package to obtain maximum likelihood estimates in the next subsection to assess the fit of a Zero-Inflated Poisson distribution to home and away goals for the 2016/17 season to compare with the Poisson distribution.

Using the Zero-Inflated Poisson distribution to model goals

Now we assess the fit of a Zero-Inflated Poisson distribution to home and away goals in the Premier League. Figure 9 shows the histogram of home and away goals scored in the 2016/17 Premier League season, where the red bars show the observed proportion of goals scored and the blue bars show the proportions given by the Zero-Inflated Poisson distribution, where its parameters are found by using maximum likelihood estimation and numerical optimisation by the `Rfast` package in R.

Looking at the 2016/17 season, we see that the Zero-Inflated Poisson distribution offers a better fit than the regular Poisson distribution, especially for the distribution of away goals scored in each game. For the Poisson distribution, the conclusion from the $\mathcal{X}^2$ test was that it suggested that a Poisson distribution was not a good fit to the observed distribution of away goals, since the test rejected that the distribution of the away goals followed a Poisson distribution at the 0.01% level. However, here for the Zero-Inflated model, we do not have enough evidence to reject the null hypothesis and the $\mathcal{X}^2$ statistic was calculated to be 11.21 with the Zero-Inflated Poisson distribution and so we do not reject the null hypothesis, that the distribution of away goals does follow a Zero-Inflated Poisson distribution, at the 10% level. However, there is still room for improvement.

If we compare this to Figure 8, then we can see visually that the Zero-Inflated Poisson has a better fit to the data, especially for goals 0 and 1. However, the Zero-Inflated Poisson model still does not have a completely great fit to the data, as we can see in the results of the goodness of fit tests (where the null hypothesis is that the data follows a ZIP(ω, λ) distribution), which are summarised in Table 7.

By looking at the results a goodness of fit test for the 2016/17 season, we see that the ZIP distribution offers a much better fit than the Poisson model does. We found in Section 4.1 that the Poisson model seems to underestimate the number of games where the home and away teams score no goals and so the Zero-Inflated Poisson model adjusts this and makes for a better distribution fit to the data compared to the regular Poisson distribution.

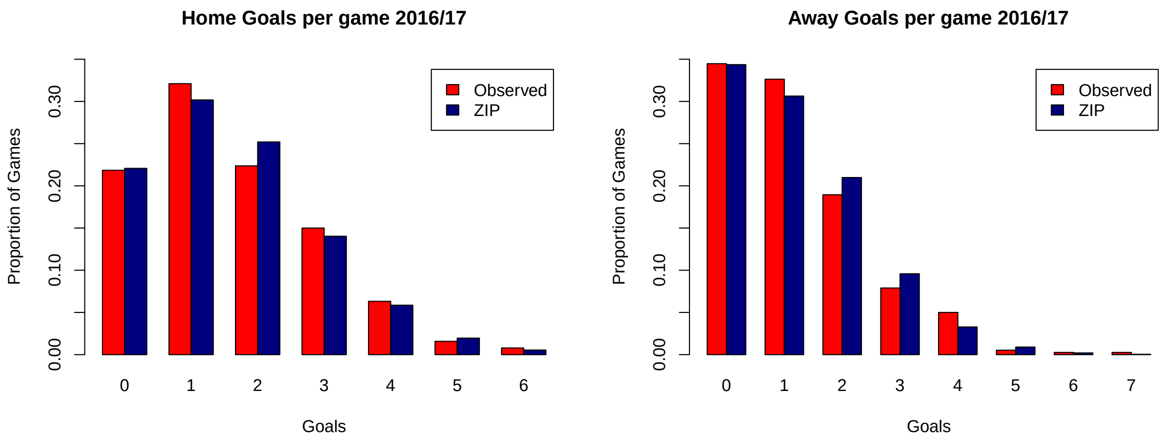


Figure 9: Histogram of number of home (left) and away (right) goals per game for 2016/17 season and comparison to a Zero-Inflated Poisson Distribution fit

Home or Away	Dist.	χ^2-Statistic	df	p-value	Outcome
Home	$ZIP(0.033, 1.65)$	2.7981	6	0.8337	Do not reject at the 10% level
Away	$ZIP(0.121, 1.37)$	11.21	7	0.1297	Do not reject at the 10% level

Table 7: Values of the χ^2 test statistic for the goals scored for the ZIP(ω, λ) model

4.2.2 Geometric-Poisson Mixture distribution

McLachlan and Peel (2004, p.6) note that given probability functions $\pi_1, ..., \pi_n$ and weights $\omega_1, ..., \omega_n$ with $\omega_i \geq 0$ for all i and $\sum_i \omega_1 = 1$, then the random variable Y has a mixture distribution with a density of the form:

$$f(y) = \sum_{i=1}^{n} \omega_i \pi_i(y).$$

The quantities $\omega_1, ..., \omega_n$ are called the *mixing proportions*. Using this, we now propose a Geometric-Poisson mixture distribution, which is defined as

$$\begin{aligned} z &\sim \text{Bernouilli}(\omega) \\ Y|\omega, z = 0 &\sim \text{Geometric}(p) \\ Y|\lambda, z = 1 &\sim \text{Poisson}(\lambda) \end{aligned}$$

and we write

$$Y \mid \lambda, p, \omega \sim \text{Geom-Poi}(\lambda, p, \omega).$$

So the probability distribution function for Y is

$$\Pr(Y = y) = \omega p(1-p)^y + (1-\omega)\frac{\lambda^y e^{-\lambda}}{y!}$$

Mean and Variance

For the Geometric-Poisson mixture distribution, the mean and variance are given by

$$\begin{aligned}
\text{E}[Y] &= \frac{\omega(1-p)}{p} + \lambda(1-\omega) \\
\text{Var}[Y] &= \omega(1-p)\Big(\frac{2}{p^2} - \frac{1}{p}\Big) - \frac{\omega^2(1-p^2)^2}{p^2} + \lambda(1-\omega)(1+\omega\lambda) - \frac{2\omega\lambda(1-p)(1-\omega)}{p}.
\end{aligned}$$

The derivation of these results can be found in the Appendix (Section G).

Maximum Likelihood Estimation for the Geometric-Poisson Distribution

Now consider a sample $\underline{y} = (y_1, ..., y_n)$ of independent and identically distributed random variables $Y_i \mid \lambda, p, \omega \sim \text{Poi-Geom}(\lambda, p, \omega)$. Then the likelihood function $l(\lambda, p, \omega; y)$ of the sample is given by

$$\begin{aligned}
l(\lambda, p, \omega; y) &= \prod_{i=1}^{n} Pr(Y = y_i) \\
&= \prod_{i=0}^{n} \Big(\omega p(1-p)^{y_i} + (1-\omega)\frac{\lambda^{y_i}\text{exp}(-y_i)}{y_i!}\Big)
\end{aligned}$$

And then the log-likelihood, $L(\lambda, p, \omega; y) = \log\, l(\lambda, p, \omega; y)$, is given by

$$L(\lambda, p, \omega; y) = n\text{log}\Big[\sum_{i=0}^{n} \omega p(1-p)^{y_i} + (1-\omega)\frac{\lambda^{y_i}\text{exp}(-y_i)}{y_i!}\Big]$$

Then this cannot be solved analytically and requires numerical optimisation. However, we can get estimates by approaching this in a Bayesian way and use MCMC methods and RStan to obtain posterior means for each variable.

Using the Geometric-Poisson distribution to model goals

We try a mixture distribution of the Geometric distribution and the Poisson distribution to model goals for the same reason as we try the Zero-Inflated Poisson. The Poisson model seems to be a relatively good fit, besides the fact that it underestimates the number of low scoring games. Hence, as the probability mass function for the Geometric distribution is strictly decreasing and has highest probability at 0, then we try a mixture distribution and test the fit of this distribution to the data for the 2016/17 season in this section.

As stated in the previous subsection, the likelihood function requires numerical optimisation to maximise and hence find the maximum likelihood estimates. Alternatively, using a Bayesian approach to solve this problem, we will propose prior distributions for each of the parameters, ω and λ and then use Stan to carry out an MCMC computation and use the posterior means as point estimates for the parameters. The prior distributions proposed for p and ω were Uniform(0,1) distributions to encode ignorance and a Gamma(3.80, 2.52) distribution was proposed for λ and was chosen using the MATCH Uncertainty Elicitation Tool (Morris *et al.*, 2014), which allows you fit distributions to your beliefs about a parameter. The prior distributions chosen for each parameter were as follows:

$$\begin{aligned}\omega &\sim \text{Uniform(0,1)}\\ p &\sim \text{Uniform(0,1)}\\ \lambda &\sim \text{gamma(3.80, 2.52)}\end{aligned}$$

The Stan and R code to find posterior means as point estimates for the parameters can be found in the Appendix (Section E).

Figures 10 shows the histogram of home and away goals scored in the Premier League for the 2016/17 season. Visually, we can first note that the Geometric-Poisson mixture distribution does not seem to be a good fit to the data, despite the fact that it passes some of the χ^2 goodness of fit tests. Table 8 summarises the results of the χ^2 test for the home and away goals for the 2016/17 season, under the null hypothesis that the distribution of the goals follow a Geometric-Poisson distribution against the alternative hypothesis that it follows a different distribution. In comparison to the Zero-Inflated Poisson distribution, we can see that this is a worse fit and so we will not be using this distribution further in our model. We chose to test this distribution, since we hoped that the combination of the Poisson distribution to the Geometric would increase the probability of zero and one goals scored, however, we can see from Figure 10 that the Geometric-Poisson distribution still underestimates the number of low scoring games.

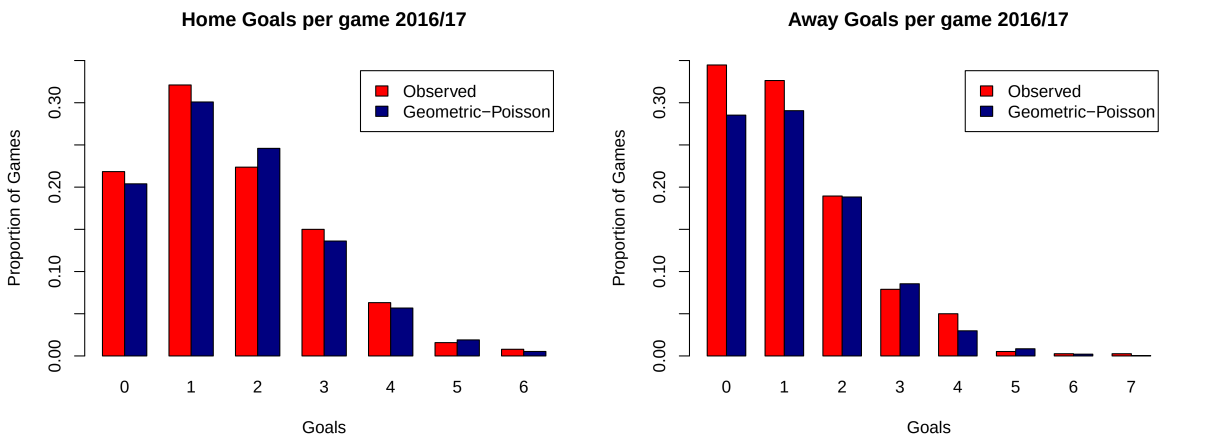


Figure 10: Histogram of number of home (left) and away (right) goals per game for 2016/17 season and comparison to a Geometric-Poisson Distribution fit

Home or Away	Dist.	χ^2-Statistic	df	p-value	Outcome
Home	G-P(0.08, 0.63, 1.67)	2.645	6	0.8519	Do not reject at the 10% level
Away	G-P(0.28, 0.61, 1.38)	9.617	7	0.2113	Do not reject at the 10% level

Table 8: Values of the χ^2 test statistic for the goals scored for the Geom-Poi(λ, p, ω) model

4.2.3 Negative Binomial distribution

In R (The R Core Team, 2017, p.1457), the negative binomial distribution with size n and probability p density is defined as

$$p(x) = \frac{\Gamma(x+n)}{\Gamma(n)x!}p^n(1-p)^x = \frac{(x+n-1)!}{(n-1)!\,x!}p^n(1-p)^x$$

for $x = 0, 1, 2, ..., n$, where $n > 0$, $0 < p \leq 1$ and $\Gamma(x)$ is the Gamma function, defined by $\Gamma(x) = (x-1)!$, where x is an integer.

If a random variable Y follows a negative binomial distribution with size n and probability p, then we write $Y \sim NB(n, p)$. The use of the negative binomial distribution to model goals data has been proposed by several authors, such as Maher (1982) and Karlis & Ntzoufras (2003). However, the use of it in models to predict goals has been largely ignored in relevant literature, perhaps due to its more complicated nature in comparison to the Poisson distribution. Now we assess the fit of the negative binomial distribution to the data.

The mean of the negative binomial distribution is $E[Y] = \mu = \frac{n(1-p)}{p}$ and the variance is given by $\text{Var}[Y] = \frac{n(1-p)}{p^2} = \frac{\mu}{p}$ (The R Core Team, 2017).

Using the Negative Binomial distribution to model goals

Figure 11 shows the histogram of home and away goals scored in the Premier League 2016/17 season, where the blue bars are the probability of each goal under the negative binomial distribution and the red bars are the observed proportions of games for each number of goals scored. The parameters n and p were estimated using maximum likelihood estimation for simplicity.

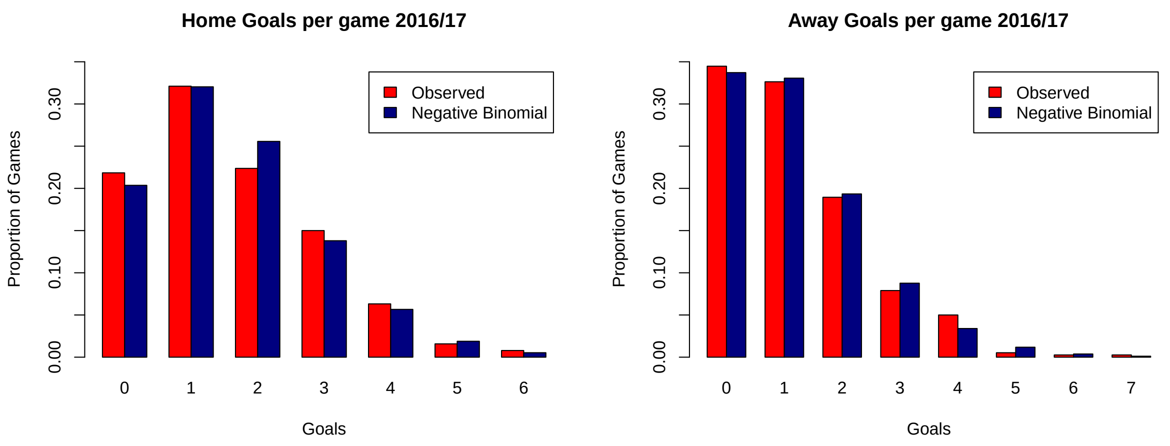


Figure 11: Histogram of number of home (left) and away (right) goals per game for 2016/17 season and comparison to a Negative Binomial Distribution fit

From visually looking at the fit, we can see that the negative binomial distribution seems to offer the closest fit compared to the other distributions that have been tried and the away goals seem to modelled very well using this distribution - much better than the fits offered by the Poisson and Zero-Inflated Poisson distributions. Table 9 summarise the $\mathcal{X}^2$ goodness of fit tests for the home and away goals, under the null hypothesis that the distribution of goals follow a negative binomial distribution, and we see that the results from these tests suggests that the negative binomial is a good fit to the data.

Home or Away	Dist.	$\mathcal{X}^2$-Statistic	df	p-value	Outcome
Home	$\text{NB}(1.60, 22.5)$	2.4614	6	0.8728	Do not reject at the 10% level
Away	$\text{NB}(1.20, 4.66)$	5.3356	7	0.6191	Do not reject at the 10% level

Table 9: Values of the $\mathcal{X}^2$ test statistic for the goals scored for the $\text{NB}(\mu, n)$ model

Since it seems that the negative binomial distribution offers the closest fit to goals data, we will proceed to build a Bayesian hierarchical model using the negative binomial distribution in Section 6.

5 An analysis of the 2016/17 Premier League Season

In this section, we discuss a model given by Baio & Blangiardo (2010) and implement this model in Stan and use the results to have a short analysis of the 2016/17 Premier League season. Baio & Blangiardo (2010) propose a Bayesian hierarchical model for the number of goals scored by the two teams in a match. Recall from Section 2.6, in hierarchical models, observable outcomes are modelled conditionally on other parameters known as *hyperparameters.*

5.1 Baio & Blangiardo's (2010) Bayesian hierarchical model

In this paper, y_{g1} and y_{g2} are used to denote the number of goals scored by the home and the away team in the g-th game of the season. In order to allow for direct comparison to Karlis & Ntzoufras's work (2003), Baio & Blangiardo considered the Italian Serie A 1991/1992 season. We will implement this model and consider the 2016/2017 Premier League season. In this model , the vector of observed counts, $\boldsymbol{y}$=(y_{g1}, y_{g2}) is modelled as independent Poisson,

$$y_{gj} \mid \theta_{gj} \sim \text{Poisson}(\theta_{gj}),$$

where $\boldsymbol{\theta}$=$(\theta_{g1}, \theta_{g2})$ represent the scoring intensity for the g-th game for the team playing at home ($j = 1$) and away ($j = 2$). Baio & Blangiardo assume a log-linear random effect model:

$$\begin{aligned}\log\,\theta_{g1} &= home + att_{h(g)} + def_{a(g)}\\ \log\,\theta_{g2} &= att_{a(g)} + def_{h(g)}\end{aligned}$$

The parameter *home* represents the advantage for the home team and it is assumed to be constant for all teams. To illustrate the need for a parameter to represent the advantage for playing at home, in the last 10 full seasons of the Premier League, 46.13% of games were won by the home team, 25.45% resulted in a draw and 28.12% ended with an away win. Since there is a much larger proportion of games resulting in a home win, this suggests that there is definitely an advantage for playing at home.

The parameters *att* and *def* represent the attacking and defensive abilities for each team and are indexed by $h(g)$ and $a(g)$, which identify the team that is playing home or away in the g-th game of the season. Theses indexes are uniquely associated with one of the teams in the league. For instance, ordering the teams in the Premier League alphabetically, then Arsenal are the first team and the index 1 will always be associated with Arsenal. Table 10 shows how the games in 2016/17 season will be indexed using these parameters.

g	Home Team	Away Team	$h(g)$	$a(g)$	y_{g1}	y_{g2}
1	Burnley	Swansea	3	16	0	1
2	Crystal Palace	West Brom	5	19	0	1
3	Everton	Spurs	6	17	1	1
...	...	...	...	...	...	
379	Swansea	West Brom	16	19	2	1
380	Watford	Man City	18	10	0	5

Table 10: The data for the Premier League 2016/17 season

Now using a Bayesian approach, prior distributions for all the random parameters in the model need to be specified. In this model, approximately flat prior distributions are used to encode ignorance. For the home advantage parameter,

$$home \sim \text{N}(0, 10000)$$

The priors for the attack and defence parameters for each team, $t = 1, ..., T$, where T is the number of teams, are

$$\begin{aligned} att_t \mid \mu_{att}, \tau_{att} &\sim \text{N}\Big(\mu_{att}, \frac{1}{\tau_{att}}\Big), \\ def_t \mid \mu_{def}, \tau_{def} &\sim \text{N}\Big(\mu_{def}, \frac{1}{\tau_{def}}\Big). \end{aligned}$$

Then to impose identifiability constraints on the team-specific parameters, Baio & Blangiardo use a sum-to-zero constraint:

$$\sum_{t=1}^{T} att_t = 0, \ \sum_{t=1}^{T} def_t = 0.$$

Finally, approximately flat prior distributions are used to model hyper-priors for the attack and defence effects,

$$\begin{aligned} \mu_{att} &\sim \text{N}(0, 10000) \\ \mu_{def} &\sim \text{N}(0, 10000) \\ \tau_{att} &\sim \text{Gamma}(0.1, 0.1) \\ \tau_{def} &\sim \text{Gamma}(0.1, 0.1) \end{aligned}$$

To summarise, the model for the number of goals scored by two teams, y_{g1} and y_{g2} for the home and away team in the g-th game, respectively, is

$$\begin{aligned} y_{g1} \mid home, att_{h(g)}, def_{a(g)} &\sim \text{Poisson}(\exp(home + att_{h(g)} + def_{a(g)})) \\ y_{g2} \mid att_{a(g)}, def_{h(g)} &\sim \text{Poisson}(\exp(att_{a(g)} + def_{h(g)})) \end{aligned}$$

A graphical representation of this model is shown in Figure 12.

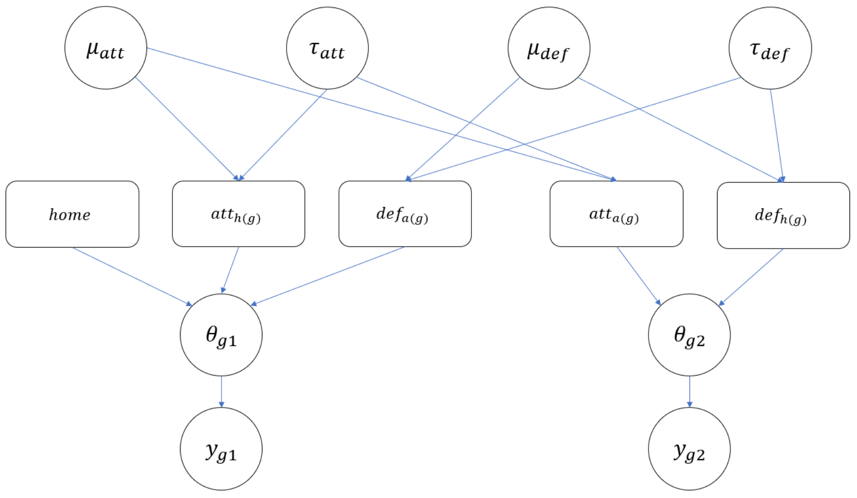


Figure 12: The DAG representation of Baio & Blangiardo's model (2010)

From this graphical representation of the model, the inherent hierarchical structure implies a form of correlation between the observable variables y_{g1} and y_{g2} by means of the unobservable hyper-parameters $\boldsymbol{\eta} = (\mu_{att}, \mu_{def}, \tau_{att}, \tau_{def})$ (Baio & Blangiardo, 2010). Therefore, the components of $\boldsymbol{\eta}$ represent a structure that the model assumes to be common for all games played in a season that contributes to the average scoring rate for the teams playing.

Therefore, as stated in Section 2.6, by using a Bayesian hierarchical model, there is no need to use a bivariate distribution to model the goals since the hierarchical structure of the model shown by the DAG in Figure 12, implies the goals scored by home and away team are conditionally independent and so correlation is taken into account.

5.2 Implementation of Baio & Blangiardo's model (2010)

The code to implement this model in Stan can be found in the Appendix (Section H) and the posterior summary statistics are presented in a table in Section H.4. As there are so many posterior parameters in this model, we will illustrate the MCMC procedure in Stan and its results by looking at just the *home* effect parameter. Figures 22a, 22b, 22c and 22d (which can all be found in the Appendix in Section H.3) shows the histogram, kernel density plot, trace plot and autocorrelation (ACF) plot for the *home* parameter, respectively. These plots are found by using the `shinystan` package (Gabry *et al.*, 2017) in R. The histogram and kernel density plots show how the *home* parameter was explored in the MCMC chains and we can see that the mean turned out to be 0.3826, which lies very near to the maximum of the kernel density. This also shows that there is definitely an advantage for teams playing games at home grounds, as the 95% credible interval is

[0.298, 0.465] to 3sf. By a simple hypothesis test, we found that the posterior probability of the home parameter to be greater than 0 was 100%, since there were no samples that were less than 0. The trace plot in Figure 22c shows the proposed value of the parameters at each iteration of the algorithm; the plot looks like random noise hence the sample is exploring the posterior density effectively, as required. The ACF plot in Figure 22d cuts off at lag 1, which is ideal for MCMC methods.

From this model, we interpret a higher att_t parameter to mean the team has a better ability to score but interpret a higher def_t parameter to mean that the team is worse at defending, as a positive def_t parameter means that the team contributes to the opposition scoring more goals. Hence by looking at the posterior summary results shown in the Appendix (Section H.4) and the posterior parameter plots shown in 13, then we can see that Tottenham Hotspurs seem to have had the best attack and defence in the league in the 2016/17 season, despite not winning the league. Chelsea were the Premier League winners in 2016/17 and from this model had the second best attack in the league and the third best defence, behind Spurs and Manchester United. On the other end of the spectrum, Middlesbrough and Sunderland had noticeably poor posterior attacking and defence parameters and so it is no surprise that they eventually got relegated to the Championship at the end of the season.

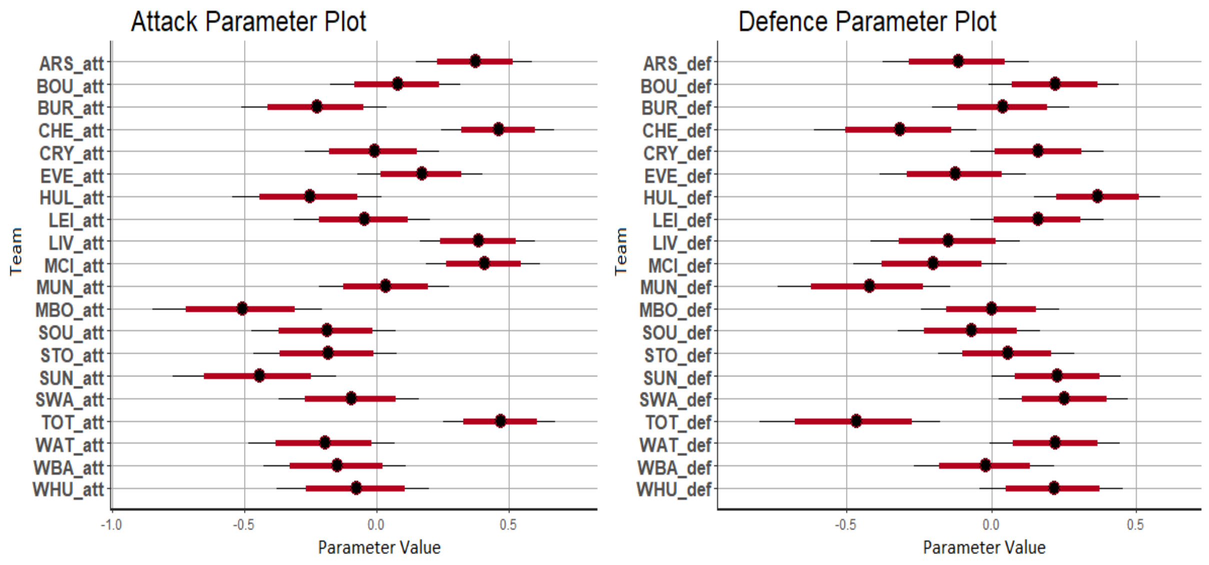


Figure 13: Posterior parameter plots for the attack and defence parameters

An alternative visualisation of these posterior summary results is shown in Figure 14. Here, we plot the posterior means for the attack parameter against the defence parameter for each team. Note that this plot does ignore information about the variance of each parameter since it only plots the posterior means, but since the variances for these parameters are quite small, then this is fine to use.

These kinds of plots can be useful in sports analytics, since we can start to look at the *characteristics* for each team. Football teams may use these plots to look for potential areas for improvement and to assess the strengths and weaknesses of other teams. For example, from the plot, we can see that Manchester United (MUN) had the second best defence in the league according to this model, but their attack parameter was low. Hence, an area of improvement for Manchester United could be to improve its attacking ability.

However, to take this idea further and make this more useful, one would need to look at *why* Manchester United's attack parameter is so low and could include other factors and data into the model, for instance, attempts at goal, number of passes or shots on target. Including these could provide some more information on what areas teams should improve on and also make the model more extensive to include what parts of football teams are good at and are bad at. However, for the model that we develop in Section 6, we will keep the model more simple since the aim of the model is for predicting the outcome of a football match, not for extensive use in football analytics.

After using the Stan programming language to implement HMC to obtain a sample from the posterior density, we can use this to obtain other values of interest, such as the predictive distribution to predict the number of goals scored by a team, or to perform some Bayesian hypothesis testing. Recall from Section 2 that in the Bayesian approach to hypothesis testing, we just need to calculate the posterior probabilities. Suppose that we have two hypotheses, $H_0 : \theta \in R$ (the *null hypothesis*), and $H_1 : \theta \notin R$, (the *alternative hypothesis*), then we just calculate the posterior probabilities,

$$\pi_0 = \pi(\theta \in R \mid x) \text{ and } \pi_1 = \pi(\theta \notin R \mid x),$$

and then decide between H_0 and H_1 accordingly.

Using this idea of hypothesis testing, we can start to compare teams and ask questions such as "is Manchester United's defence better than Tottenham's?". For these types of questions, we can obtain an estimate from this model by extracting the sample of draws in R using the `rstan` package (Stan Development Team, 2018) and count the number of times where Manchester United's defence parameter was lower than Tottenham's defence parameter (recall a lower defence parameter implies a better defensive ability in this model). From this implementation of the model, we found that the probability of Manchester United having a better defence than Tottenham's in the 2016/17 season was about 37%.

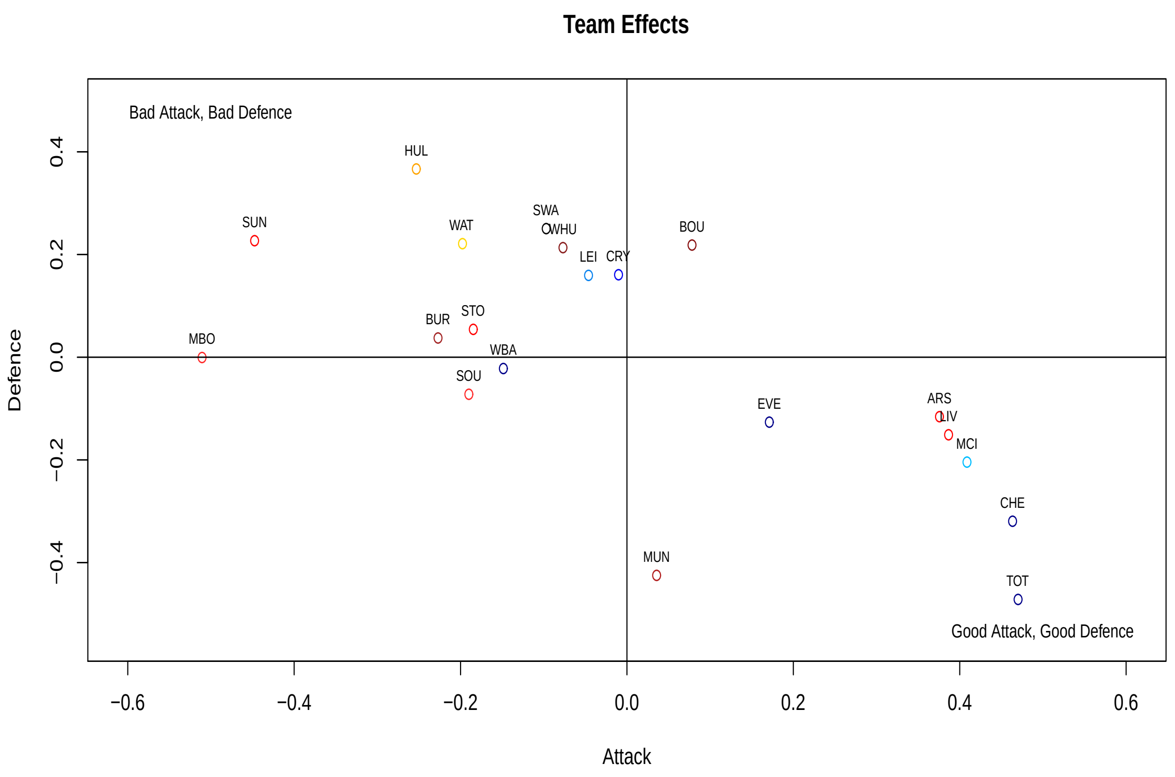


Figure 14: A plot of the posterior means of the attack and defence parameters for each team

We now show a small example of how we would use this model to start predicting football matches and apply this model to try and predict the outcome of a match between Manchester United and Crystal Palace on the last day of the 2016/17 season. Once we have a sample from the posterior distribution, we can draw from predictive distribution of unobserved data or future data, y^*, which are the goals scored by either team in this case. For each draw of θ, we draw a sample y^* from the predictive distribution $\pi(y^* \mid \theta)$.

In the case of one football match, we take our samples for the home, attack and defence parameters for both teams and then simulate from a Poisson distribution for each sample using:

$$\log\,\theta_1 = home + att_{MUN} + def_{CRY}$$
$$\log\,\theta_2 = att_{CRY} + def_{MUN}$$

Next, we obtain draws from our likelihood, $\pi(y_j^* \mid \theta_j)$, for $j = 1, 2$, (from a Poisson distribution) and then we estimate the probabilities as

$$\begin{aligned}
\Pr(\text{Manchester United Win}) &= \frac{\text{Number of times } y_1^* > y_2^*}{\text{Number of samples}}, \\
\Pr(\text{Draw}) &= \frac{\text{Number of times } y_1^* = y_2^*}{\text{Number of samples}}, \\
\Pr(\text{Crystal Palace Win}) &= \frac{\text{Number of times } y_1^* < y_2^*}{\text{Number of samples}}.
\end{aligned}$$

From using the data from the 2016/17 season (with the Manchester United vs. Crystal Palace game removed), the estimate of the probabilities were

$$\begin{aligned}
\Pr(\text{Manchester United Win}) &= 0.637, \\
\Pr(\text{Draw}) &= 0.223, \\
\Pr(\text{Crystal Palace Win}) &= 0.140.
\end{aligned}$$

5.3 Model criticisms

A criticism of this model is that the prior distributions that were proposed were too flat and the variances could have been reduced significantly. Baio & Blangiardo (2010) used approximately flat prior distributions to encode ignorance. For the μ_{att}, μ_{def} and *home* parameters, $\mathrm{N}(0, 10000)$ priors were used; with a standard deviation of 100, these parameters can vary so much that the prior distributions have no discernible impact on the analyses. By simulating values from the priors, we are able to obtain a pre-posterior distribution of goals scored by two teams in a match. Recall from Section 2, the *preposterior distribution* encodes our uncertainty about what data we will observed before any data is available yet. The R code to simulate these values can be found in the Appendix (Section H.5). From simulating 10000 games from these distributions, we find that in 9481 games, the home team returned `NA` values and in 9488 games, the away team returned `NA` values. The reason for this is that the number of goals given by the model are too large. Additionally, the prior distributions came up with games where the away team would score 446985258 goals. From this we can see that the prior distributions are too flat, since they imply that almost any score can happen. Although we see that the model seems to produce sensible results, the original priors are not sensible and the variance of the prior distributions can be reduced significantly. If more sensible prior distributions were used with smaller variance, we would be more certain about the distribution of goals apriori.

Baio & Blangiardo (2010) also proposed alternative priors to reduce the effect of *overshrinkage*, which means that extreme occurrences are pulled towards the grand mean. However, the prior distributions in this model are equally as flat with some changes by putting truncated normal prior distributions on different teams. We will not be looking into this model and in the next section, we start to develop a different Bayesian hierarchical model.

6 Negative Binomial Model

In Section 4, we found that to model the goals scored in the Premier League, alternative distributions to the Poisson distribution offered a closer fit to the data, such as the Zero-Inflated Poisson or the negative binomial distribution. In this section, we propose a different Bayesian hierarchical model for the number of goals scored by two teams in a match but the main difference is that rather than using a Poisson distribution to model goals, this model will use the negative binomial distribution.

An alternative parametrization is by the mean μ and size n, which is used in the Stan documentation (2017); the probability function is given below. In this case, $p = \frac{n}{n+\mu}$ and the variance is given by $\mu + \frac{\mu^2}{n}$ and the probability mass function is given by

$$p(x) = \frac{(x+n-1)!}{(n-1)!\,x!}\Big(\frac{n}{n+\mu}\Big)^n\Big(\frac{\mu}{\mu+n}\Big)^x.$$

If a random variable Y follows a negative binomial distribution with mean μ and size n, then we write $Y \sim \mathrm{NB}(\mu, n)$.

In order to start putting prior distributions on random variables, we will use the parametrisation of the negative binomial in terms of the mean μ and size n, since this is an alternative parametrisation used in Stan and it also seems more natural to propose priors for the mean when modelling football goals, whereas to try and propose prior distributions for the probability, p, of a team scoring is less intuitive.

6.1 Negative Binomial Model

In this model, we use y_{g1} and y_{g2} to denote the number of goals scored by the home and away team in the g-th game of the season, respectively. Here, the vector of observed goals, $\boldsymbol{y} = (y_{g1}, y_{g2})$ are modelled using a independent negative binomial distribution,

$$y_{gj} \mid \mu_{gj}, n_j \sim \mathrm{NB}(\mu_{gj}, n_j),$$

where $\boldsymbol{\mu_{gj}} = (\mu_{g1}, \mu_{g2})$ represents the mean number of goals expected to be scored by the home team ($j = 1$) and the away team ($j = 2$) in the g-th game of the season. Similar to Baio & Blangiardo (2010), we assume a log-linear random effect model, as it allows for the condition that the mean number of goals must be positive:

$$\log \mu_{g1} = \mathit{home\_att}_{h(g)} + \mathit{away\_def}_{a(g)}$$
$$\log \mu_{g2} = \mathit{away\_att}_{a(g)} + \mathit{home\_def}_{h(g)}$$

A common feature of models for football scores is that the home-effect parameter is constant. In the model proposed by Baio & Blangiardo (2010), the *home* parameter represents the advantage for the team playing at home and is assumed to be constant for all teams in the season. However, this seems too generalised, as some football teams are especially good at home, whereas for some teams the effect of playing at home does not seem to be as strong. For example, consider Sunderland in the 2016/17 season; they won only 3 times at home and scored 16 goals, whereas they only scored 13 away from home and won 3 times. Hence we can see that the effect of playing at home did not have much significance for Sunderland, as they did not win at home after the new year. However, for Chelsea, they did seem to perform better at home than they did away, as they scored 55 goals at home and 30 in away games for the 2016/17 Premier League season. Although simply looking at goal scored by a team does not represent how well a team plays and more complicated metrics are needed to determine whether a constant home-effect parameter is suitable, it seems that using a constant home-effect parameter does not seem to be adequate to describe the varying home-effects for different teams. Hence in this model, we will encode the home effect for each team by separating the attack and defence parameters for each team into two parameters representing whether or not they are playing at home.

In this way of conditioning the mean number of goals μ_{gj}, we are able to encode more information, as we have now have information on how well a team performs in attack and defence and also how these vary depending on whether they are playing home or away. In contrast to Baio & Blangiardo's model (2010), the attack and defence parameters are deemed to be equal whether they are playing at home or away and the home-effect is measured to just add towards the number of goals scored by the home team. This is because only θ_{g1} is conditioned on the home parameter, whereas θ_{g2} does not. Hence, the *home* parameter does not encode any information about much better a team would defend playing at home. But now this model and its choice of parameters allows for varying performances in attack and defence for each team and which depends on whether the team is playing home or away.

These parameters will be indexed in the same way as they were when implementing Baio & Blangiardo's model (2010) in Section 5.1. In this model, the prior distributions for the home and away parameters for the attacking and defensive strengths of each team, $t = 1, ..., T$, where T is the number of teams, are

$$\begin{aligned}
home\_att_t &\sim \mathrm{N}(\mu_{h\_att}, \sigma^2_{att}),\\
away\_att_t &\sim \mathrm{N}(\mu_{a\_att}, \sigma^2_{att}),\\
home\_def_t &\sim \mathrm{N}(\mu_{h\_def}, \sigma^2_{def}),\\
away\_def_t &\sim \mathrm{N}(\mu_{a\_def}, \sigma^2_{def}).
\end{aligned}$$

To impose identifiability constraints on these parameters, we use a sum-to-zero constraint, that is,

$$\sum_{t=1}^{T} home\_att_t = 0 \ , \ \sum_{t=1}^{T} away\_att_t = 0 \ , \ \sum_{t=1}^{T} home\_def_t = 0 \ , \ \sum_{t=1}^{T} away\_def_t = 0.$$

Then the prior distributions for the hyperparameters are given as follows:

$$\begin{aligned} \mu_{h\_att} &\sim \text{N}(0.2, 1), \\ \mu_{a\_att} &\sim \text{N}(0, 1), \\ \mu_{h\_def} &\sim \text{N}(-0.2, 1), \\ \mu_{a\_def} &\sim \text{N}(0, 1). \end{aligned}$$

where the slight difference in the mean for the home parameters are used to encode a belief that teams tend to play better at home. Note that these are vastly different compared to the prior distributions placed on μ_{att} and μ_{def} in Baio & Blangiardo's model (2010) in Section 5. As noted in 5.3, the prior distributions were approximately flat and since the variance for these parameters were so large, the prior distributions had no discernible impact on the analyses and the pre-posterior distribution of goals scored given by their prior distribution were not sensible. However, with these chosen priors, the pre-posterior distribution of goals scored by home and away team was more sensible with no `NA` values returned. The code to simulate from the prior distributions to obtain a pre-posterior distribution can be found in the Appendix in Section I.3.

The prior distributions for the variance of the attack and defence parameters are:

$$\begin{aligned} \sigma^2_{att} &\sim \text{Gamma}(10, 10), \\ \sigma^2_{def} &\sim \text{Gamma}(10, 10). \end{aligned}$$

Lastly, the prior distributions for the size n in the model are,

$$\begin{aligned} n_{home} &\sim \text{Gamma}(2.5, 0.05), \\ n_{away} &\sim \text{Gamma}(2.5, 0.05). \end{aligned}$$

The prior distributions for n_{home} and n_{away} were chosen to have a large variance, since we were uncertain about these parameters apriori.

A graphical representation of this model is shown in Figure 15.

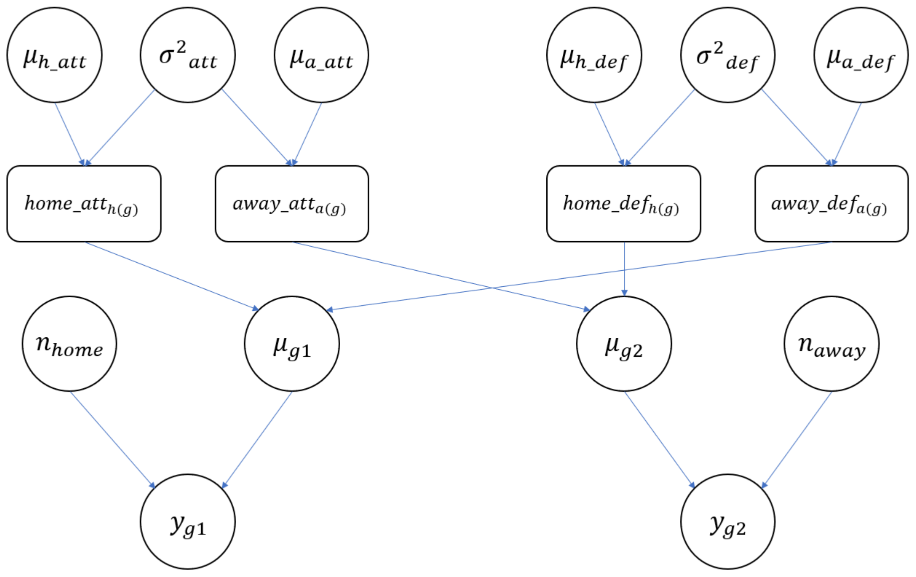


Figure 15: The DAG representation of the Negative-Binomial Model

6.2 Implementation of the Negative Binomial model in Stan

The code to implement this model in Stan and R can be found in the Appendix (Section I) and the posterior parameter plots for the attack and defence parameters can be found in Figures 16 and 17, respectively. Comparing these with the posterior parameter plots for the model given by Baio & Blangiardo (2010), shown in Figure 13, we can see that by this way of splitting up the parameters, we are able to see more information about whether or not a team does seem to play better at home. For instance, before we used Sunderland as an example of team who does not seem to play better at home and these plots support this claim since Sunderland had a better attacking parameter for playing away rather than home. The posterior mean for the home attack parameter was -0.557 and the posterior mean for the away attack parameter for Sunderland was -0.371 (higher is better for attack). By comparing the posterior samples of the attack parameters for home and away for Sunderland, from this model, we believe aposteriori that Sunderland has a better attack at home than away with 27% certainty. However, in defence, the model suggests that the probability that Sunderland defend better at home is 70%. This highlights an advantage of splitting the attack and defence parameters for each team, since we can encode more information about how the teams play at home or away.

Since Figure 16 and 17 captures most of the posterior summary statistics for the attack and defence parameters in this model illustrating the mean and spread of the data, the table of the posterior summary statistics can be found in the Appendix instead, in Section I.4.

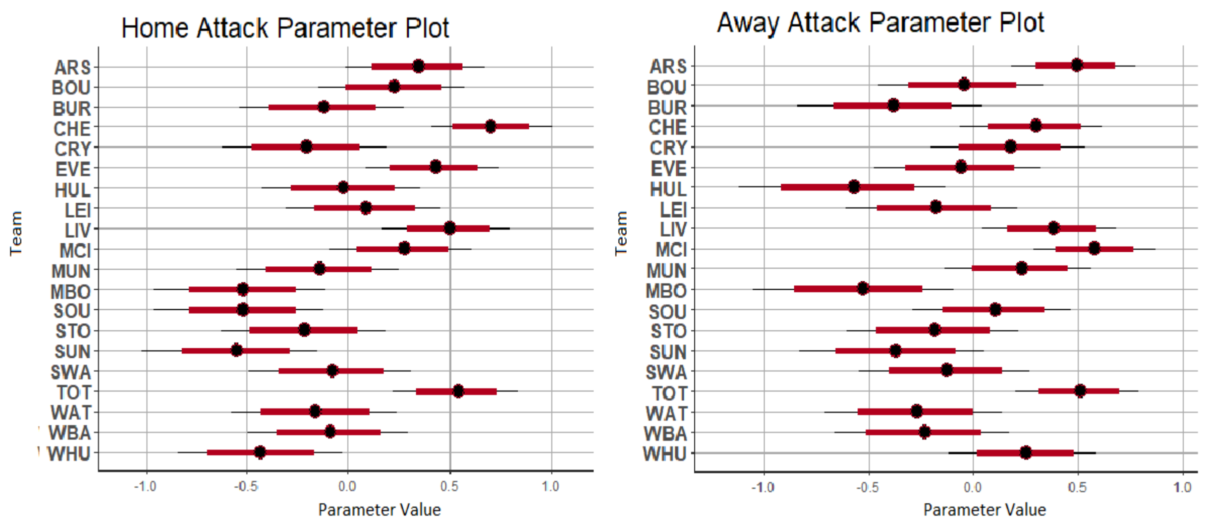


Figure 16: Posterior parameter plots for the attack parameters for home and away

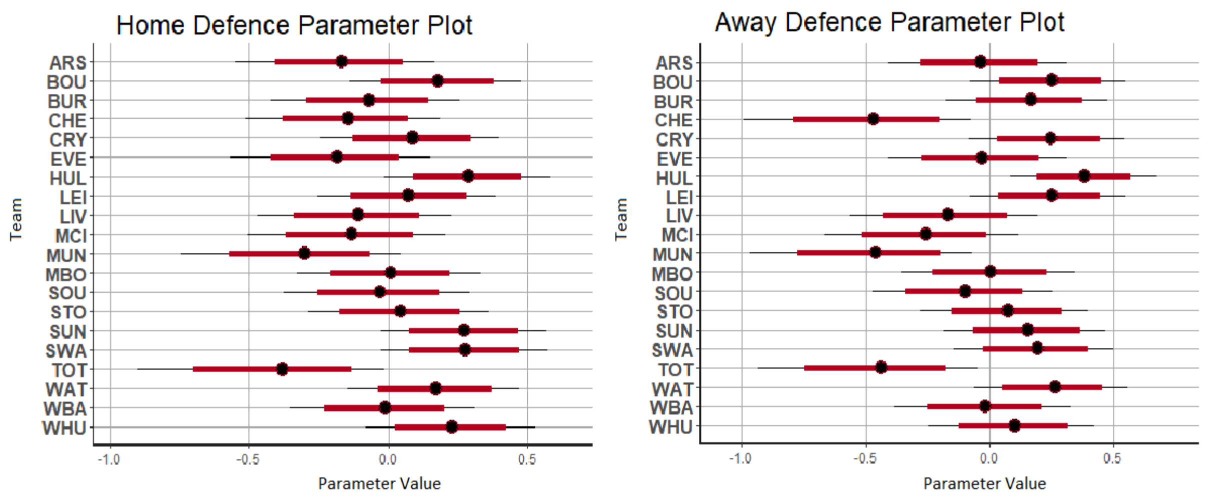


Figure 17: Posterior parameter plots for the defence parameters for home and away

Figures 18 and 19 plots the posterior mean of the attack parameter against the posterior mean of the defence parameter for each team playing home and away, respectively. Since it is better to have a lower defence and higher attack parameter, it is best to be in the bottom right of the plot, whereas being in the top left of the plot suggests that the team has a bad attack and a bad defence. Comparing these to the plot given in Figure 14, we can see again that we can encode more information about team performances by splitting the attack and defence parameters for home and away. For example, Figure 18 and 19 suggest that Chelsea have the highest attacking parameter while at home but have the fifth highest attacking parameter when playing away from home. In contrast, from Figure 14, this plot only indicates that Chelsea has the second best attacking parameter, with no information about how they attack when they are home or away. In Baio & Blangiardo's model (2010), only the *home* parameter accounts for home advantage, which is assumed to be constant for all teams.

In addition, by looking at the plots in Figures 18 and 19, we can see that Chelsea have a much lower defence parameter when playing away. Baio & Blangiardo's model (2010) does not allow for varying defensive performances by the team, since only θ_{g1} is conditioned on the home parameter, whereas θ_{g2} does not. Therefore, Chelsea is assumed to have the same defence parameter whether they are playing home or away, but from this model, we infer that Chelsea generally has a better defence parameter when playing away from home.

One could also take the mean of the home and away effects of the attack and defence parameters to obtain an 'overall effect' estimate for each team; see Figure 23 in the Appendix (Section I.4).

Note that while these plots are useful as a summary of the parameters for each team, only the posterior means are plotted and they ignore the spread of the samples obtained for each parameter. Therefore, strictly speaking, above we are only saying that Chelsea has the highest posterior mean value for the home attacking parameter, not that they definitely have be best home attacking parameter. The spread of the samples obtained for the attack and defence parameters are displayed in Figure 16 and 17 and the table of posterior summary statistics can be found in the Appendix in Section I.4.

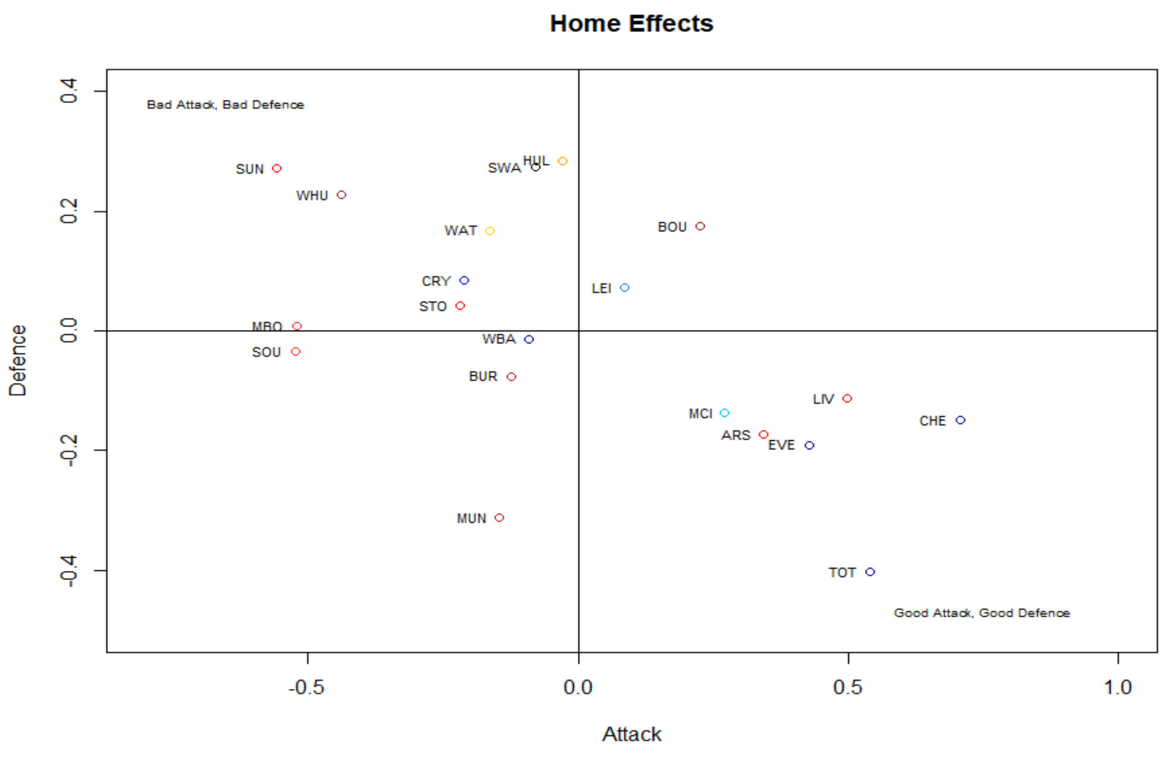


Figure 18: A plot of the posterior means of the attack and defence parameters for each team

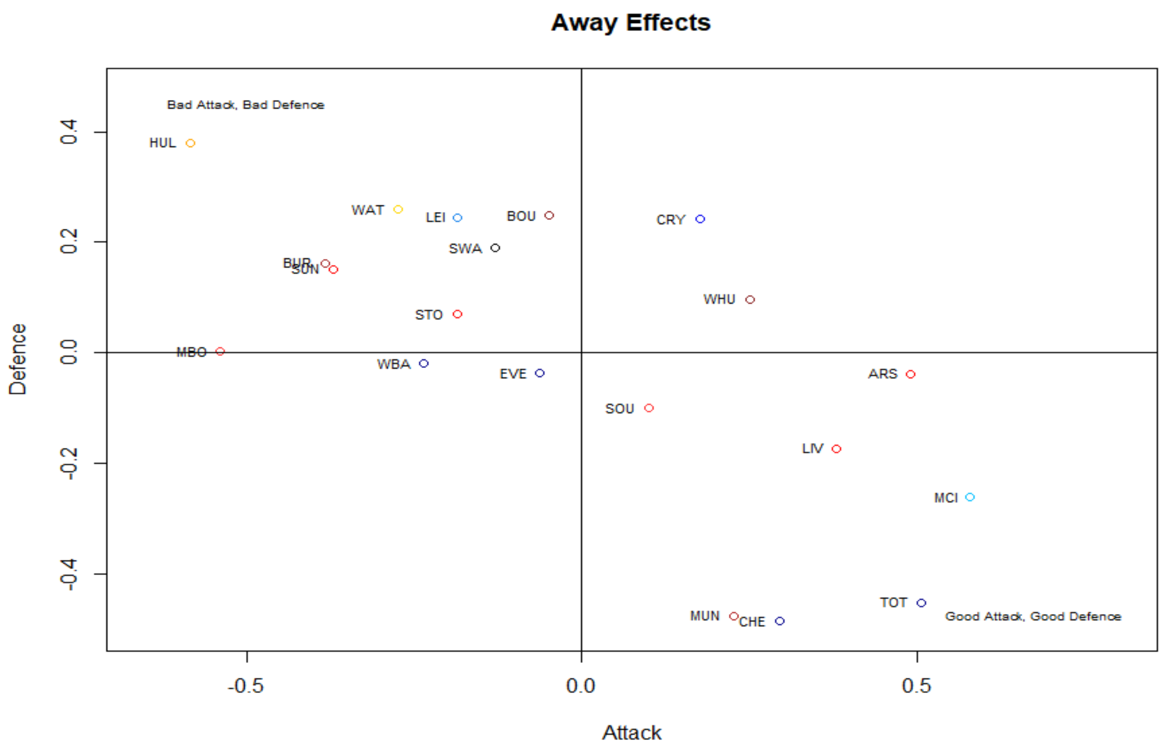


Figure 19: A plot of the posterior means of the attack and defence parameters for each team

7 Model assessments and comparisons

In Section 5, we presented a model developed by Baio & Blangiardo (2010) to model the number of goals scored in a football match by each team and in Section 6, we proposed a different Bayesian hierarchical model which used a Negative Binomial distribution to model goals scored. We let 'BB' denote Baio & Blangiardo's model and 'NB' to denote the Negative Binomial model presented in Section 6. The aim of these models is to predict future fixtures or a league table and so in this section, we use various methods to assess the predictive power of the models, such as *cross validation*, the *brier score* and *rank probability score*, which will all be explained in Sections 7.1 - 7.3 and we will discuss the strengths and weaknesses of each probability score. In Sections 7.4 - 7.5, we look at two more ways to assess the model by attempting to predict a league table from the models and also by seeing how the model performs when using it as a basis of a betting model.

7.1 Cross-Validation

One method to assess the predictive power of a model is to fit the model to the whole dataset and then to evaluate its performance on the same dataset. For the problem of predicting football matches, this method would use the entire dataset of games to obtain posterior samples for the parameters (i.e. the attack and defence parameters) and then use these samples to simulate and predict the same games. However, as Arlot & Celisse (2010, p.52) note, "training an algorithm and evaluating its statistical performance on the same data yields an overoptimistic result". In general, it is not good practice to use the same data to obtain predictions and to test the model. Cross-Validation (CV) can be used to help eliminate this problem. If there is enough data, it is better to split the dataset into two sets - a *training set* and a *test set.* Next, we use the training set to build the model and to obtain posterior samples of the parameters of interest and then use the test set to assess our model and to validate predictions. By 'validating a prediction' we mean that we assess how accurate the model was for the test set. For a predictive football model, we might validate either how close it was to predict the number of goals scored by each team or if it correctly predicted the outcome of the match - we will only assess the model performance for outcome predictions (i.e. if the match ended in a home win, draw or an away win).

From our predictive distribution, we can obtain probabilities for the outcomes of a match - a home win, a draw or an away win. For the purposes of assessing our model, we use the simple decision rule that we choose the outcome of the game with the highest estimated probability. Of course, more complicated decision rules can be developed. For instance, bettors may want to look for bets with good value, where the bookmaker's odds are higher than ours. So a bettor may compare the model probabilities and a bookmaker's *implied probabilities* (probabilities that are calculated using the bookmaker's odds using the formula $\tilde{p} = (1/\text{decimal odds}))$ and having a decision rule to choose an outcome from that, but for simplicity, we decide to choose the outcome with highest estimated probability.

Suppose there are N number of samples in the dataset, then there are several variants of cross-validation (Arlot & Celisse, 2010), such as:

1. Leave-one-out Cross-Validation (LOOCV)
   In LOOCV, each data point in the dataset is successively 'left out' from the sample and is then used for validation.

2. Leave-p-out Cross-Validation (LPOCV)
   In LPOCV, where $1 \leq p \leq N$, every possible subset of p data is successively 'left out' from the data the sample and is then used for validation.
   Note that LPOCV with $p = 1$ is just LOOCV.

3. K-fold Cross-Validation
   We first split the dataset into subsamples of size approximately equal to $\frac{K}{N}$ and then each subsample successively plays the role of the validation sample (test set) and the remaining data is the training set.

4. Monte Carlo Cross-Validation (MCCV)
   In LOOCV and LPOCV, all of the data is used and so can become computationally expensive if it requires a lot of computational power to build one model and the number of samples in the dataset is large. A less computationally expensive CV method is Monte Carlo Cross-Validation (MCCV) (Xu & Liang, 2000).
   In MCCV, the dataset is randomly split into two parts, one is used as a training set and the other as the test set to validate. This is then repeated m times, say, and we take the average of the CV scores obtained.

Xu & Liang (2000, p.4) state by means of the MCCV method, the amount of computation can be reduced substantially. For example, if we used this for a season that consists of 380 games in the Premier League, for LOOCV, we build the model 380 times, successively taking a game out and predicting that game. However, with MCCV, we can choose to do this a number of times and split the data randomly with 80% of the data being used as the training set and 20% of the data being used as the test set, fitting the model with the training set and then predicting the outcomes or score of the matches represented in this testing set. The accuracy will vary each time due to the randomness in the train/test dataset splitting so it is repeated several times. Then, we average the resulting accuracy estimates to obtain a estimate of how good each model is.

The function to cross-validate a particular test set and training set for a model can be found in the Appendix (Section J.1), named *WLD_validate* (WLD standing for Win-Loss-Draw). This uses a different function called *predict_game*, which predicts a game using one of the two models described in Sections 5 and 6. In *WLD_validate*, we predict a game in the test set and then choose the outcome with the highest probability. Next, the function checks the actual outcome and records a `TRUE` value if the model correctly guessed the outcome and a `FALSE` value otherwise. The function then returns a vector of `TRUE` and `FALSE` values and we can calculate the percentage of `TRUE` values to find the percentage of correct predictions.

A possible problem occurs when using MCCV to test these models, since our data includes temporal information, as it includes the date of each game. This is important because a team at a given time may be having a winning streak or losing streak, which can affect the outcome of the game. Moreover, during a season other factors such as a managerial change or an injury to a star player can have effects on a team's performance and so by assessing the predictive performance of a model using MCCV can cause bias. This problem is sometimes referred to as *look-ahead bias.* Investopedia (2018) state that *look-ahead bias* occurs by using information or data in a study or simulation which would not have been known or available during the period being analysed. They note that this could lead to inaccurate results. To make sure we avoid look-ahead bias when assessing the performance of the model, we must only use information that would have been available at that time. This also mimics how the model would have been used in a real world setting because if we were to use such models for football predictions, then we could only ever use past game data. Therefore, we will use a variant of cross-validation where we will progressively extend the amount of data being fed into the model for the training set and will try to predict the games in the season as if we were predicting the results of games on a week-by-week basis. We call this variant of cross-validation as *Sequential Cross-Validation* (SCV), since we are progressively extending the amount of data in the training set sequentially through the dataset. With this type of cross validation, predictions should get better as the rounds of games progress, since more data is available in the model. How this is implemented is explained in Section 7.1.2.

7.1.1 MCCV Results

In MCCV, the dataset is randomly split into training and test sets to assess the model and is repeated several times. This is done by using the *MCCV* function that can be found in the Appendix (Section J.1). This function requires a test percentage, k say, to be passed and then the dataset is split into a test and training set randomly, where $k\%$ of the data is the test set and $(100 - k)\%$ of the data is the training set. The function then uses *WLD_validate* to find a cross validation score for this test set. We iterate this process several times and then return the a set of MCCV scores. From this we can find the mean of these scores. Additionally, we can find the standard deviation and standard error of this estimate for the cross-validation score of the model.

For our tests for the models, we ran 50 iterations to obtain 50 MCCV scores where we split the data randomly into an 80% training set and a 20% test set. Here the data that we used is the 2016/17 Premier League season so the test set would always contain $(0.2 \times 380) = 76$ games and there would be $(0.8 \times 380) = 304$ games in the training set. The Monte Carlo cross-validation scores for each of the four models are summarised below in Table 11.

Model	MCCV Score (Mean average of scores)	Standard Deviation	Standard Error of Accuracy
BB	0.603	0.0478	0.007
NB	0.581	0.0604	0.009

Table 11: Monte Carlo Cross Validation (MCCV) Results

From the MCCV results, these suggest that the model presented by Baio & Blangiardo (2010) is more accurate since it correctly guessed, on average, around 60% of games when the data was randomly split into test sets (20% of the data) and training sets (80% of the data), whereas the Negative Binomial correctly predicted the outcome of a match about 58% of the time on average. However, as noted before, we will also calculate a *Sequential Cross-Validation score* to progressively extend the amount of data in the training set to try predict future games to avoid look-ahead bias in the scoring rule.

7.1.2 SCV Results

To obtain a fairer assessment of the model, we limit ourselves to only use data that would have been available at the day of each game. One possible way to do this is to use LOOCV, but as stated before this is computationally expensive. Alternatively, we can progressively extend the amount of data in the training set. The R code to calculate the SCV score can be found in the Appendix (Section J.1), in the function called *seqCV*. The trick used to avoid LOOCV is that in the Premier League and other leagues, the games are generally played in rounds. The only problem is that sometimes teams do not play in round and get games rescheduled due to other competition commitments. Hence, in this function, it requires a minimum number of games, n, that will be in the training set from the beginning. The data passed will then be split into a test and training set, where the training set is the first n games of the dataset. If the total number of games is N, the remaining $N - n$ games are the games to be predicted. To obtain a test set, we loop through these remaining games and for each game we add the teams playing in a list. We stop this loop if only one or no teams are added to the list, since there is a repeat of a team and we then label these games as the test set. Next, the function fits a model with the training set and see how the model predicts these games. The function then adds the games into the training set and starts to predict the next round in the same way. This loop carries on until all games in the season are in the training set, meaning that there are no more games to be predicted. Put simply, the function progressively adds to the training set to predict the next round of games. The sequential cross validation scores for the three models of interest are summarised below in Table 12.

Model	Sequential CV Score
BB	0.638
NB	0.651

Table 12: Sequential Cross Validation (SCV) Results

The data that is tested is always the last 370 games of the season, since we use the first 10 games as a training set to begin. Therefore, there is no need to repeat this process and average scores as we did with MCCV. Therefore, from the results shown in Table 12, then we can see that the Negative Binomial correctly predicted more outcomes than Baio & Blangiardo's model (2010). However, note that since there are only 370 games in the overall test set, then the Negative Binomial distribution only got 5 more correct games than Baio & Blangiardo's model (2010).

7.2 Brier Score

Cross-validation provides a useful tool to assess how well a model predicts results, however, it generally ignores the probabilities at which we make the predictions. Especially in the case where our decision rule is to choose the outcome with the highest probability. To illustrate why this is a problem, consider a Premier League match between Watford and Swansea, then using the Negative Binomial model with the training set as the whole 2016/17 season, the predicted probabilities for a future game between these two teams is:

$$\begin{aligned}\Pr(\text{Watford Win}) &= 0.351\\ \Pr(\text{Draw}) &= 0.295,\\ \Pr(\text{Swansea Win}) &= 0.354.\end{aligned}$$

In this case, we could predict a Swansea away win, but note that we are only 35.4% certain that this is outcome occurs, which is pretty low - lower probability than guessing head or tails when flipping a fair coin. In contrast, consider a game between two unevenly matched opponents, where the model estimates a home win with 90% probability. The problem using cross-validation above is that it does not take into account the probabilities that we are predicting the outcome. Now suppose that we used cross-validation to assess a game between Watford and Swansea and the result was a Watford win, we would just get 0%. But this is the same score as getting the game wrong where we were 90% certain of a home win, which is a worse since, the model was so convinced that the home team would win.

Another problem is how the model distributes the probabilities of the unobserved events. To illustrate this point, consider that we have two models, model A and model B. Then Table 13 shows some predictions for hypothetical matches. From match 1, although both models predict a home win with the same probability and is the most probable event for both models, model A is preferred over model B because with model A, the distribution for draw and away win is quite even, whereas model B places much higher probability of an away win than a draw at 0.45, which is quite close to 0.5, the model's estimate for a home win. In game 2, since model A predicted a home win with a much higher probability than model B, then model A is clearly better, this is similar to the situation described above. In game 3, although both models had the same probability for a draw, which was the observed outcome, model A is preferred since it had similar probability for a home or away win, whereas model B predicted a home win at a much higher probability than a draw.

Match	Model	Pr(Home win)	Pr(Draw)	Pr(Away win)	Result	Better model
1	A	0.5	0.3	0.2	Home win	A
	B	0.5	0.05	0.45		
2	A	0.8	0.15	0.05	Home win	A
	B	0.4	0.3	0.2		
3	A	0.375	0.3	0.325	Draw	A
	B	0.65	0.3	0.05		

Table 13: Hypothetical probability estimates from two different models, A and B

An alternative verification score that accounts for the probability at which the model predicts all events is the *Brier score.* The *Brier score* was proposed by Brier (1950) as a means for verifying weather forecasts, but we will alter this to assess the predictive capabilities of each football model that has been discussed so far.

Suppose that we have n observations where only one of r possible events can occur. For each observation i, $i = 1, ..., n$, we have forecast probabilities (estimated probability for a specific outcome) $\hat{\pi}_{i1}, ..., \hat{\pi}_{ir}$ that the event r occurs, respectively. The r possible events are chosen to be mutually exclusive and exhaustive so that

$$\sum_{j=1}^{r} \hat{\pi}_{ij} = 1.$$

The *Brier score* for an individual forecast, j, for observation i is defined as

$$BS_{ij} = (O_{ij} - \hat{\pi}_{ij})^2,$$

where O_{ij} takes the value 1 if the forecast event occurred and 0 otherwise and $\hat{\pi}_{ij}$ is the probability of the forecast. Put simply, the Brier score for an individual forecast is defined as

$$\text{Brier score for individual forecast} = (\text{Actual result - Forecast Probability})^2.$$

For the entire set of observations, the Brier score is defined as the sum over all the individual Brier scores for each forecast and for each observation divided by the total number of observations:

$$BS = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{r}(O_{ij} - \hat{\pi}_{ij})^2,$$

Note that a higher Brier score indicates a worse probability prediction scores, whereas a lower Brier scores suggests a better predictive score.

Example: Brier score for predicting football match outcomes with 3 events

Consider the forecast probabilities for outcomes of football matches shown in Table 13, then Table 14 shows how the Brier score can be calculated for each game for each model. From this example, we can see that for each game, the Brier score is lower for model A. Hence, the Brier score can recognise the problems that were discussed above since it considers the probability of forecasts for unobserved events and so we can obtain a more accurate score for model assessment.

Match	Model	Result	Brier Score calculation for game	Brier Score
1	A	Home win	$(1-0.5)^2 + (0-0.3)^2 + (0-0.2)^2$	0.38
	B		$(1-0.5)^2 + (0-0.05)^2 + (0-0.45)^2$	0.455
2	A	Home win	$(1-0.8)^2 + (0-0.15)^2 + (0-0.05)^2$	0.065
	B		$(1-0.4)^2 + (0-0.3)^2 + (0-0.2)^2$	0.49
3	A	Draw	$(0-0.375)^2 + (1-0.3)^2 + (0-0.325)^2$	0.73625
	B		$(0-0.65)^2 + (1-0.3)^2 + (0-0.05)^2$	0.915

Table 14: Example calculations of the Brier score for hypothetical probabilities given in Table 13

By using the simple decision rule to pick the highest probable event and cross-validation to assess the model, the models would be indistinguishable, since both models correctly predicted the first two games and got the third wrong, so would have obtained a cross-validation score of 66.7%. In contrast, the Brier score for model A is given by $BS_A = \frac{1}{3}(1.18125) = 0.39375$ and the Brier score for model B is $BS_B = \frac{1}{3}(1.86) = 0.62$, and so model A is preferred under this scoring rule.

Further note that if a model just guessed the outcome of a match by assigning equal probability to all outcomes, then the Brier score for this prediction, regardless of outcome, is 0.667 (to 3dp), since if the outcome was a home win,

$$BS = \left(1 - \frac{1}{3}\right)^2 + \left(0 - \frac{1}{3}\right)^2 + \left(0 - \frac{1}{3}\right)^2 = \frac{2}{3}.$$

If the outcome was a draw or an away win, the first term in the sum is just swapped with the second or third term. Therefore for a prediction that is better than purely guessing, a Brier score of less than $\frac{2}{3}$ is preferred.

To obtain a Brier score for each model, we calculate a *Sequential Brier Score* (SBS), which again is the same as SCV, but we calculate the Brier score rather than the cross-validation score where we validate the highest probable event prediction. Here, we will not calculate a *Monte Carlo Brier Score* (MCBS), which is the same as MCCV, but instead we calculate the Brier score for the test set in order to avoid look-ahead bias in our score. However, the code to find a Monte Carlo Brier Score was written and can be found in the Appendix (Section J.2) in the function called *MCBrier*.

7.2.1 Brier Score Results

The R code to obtain the Brier score for a match and a SBS score can be found in the Appendix (Section J.2) in a function called *seqBrier*. This uses a similar algorithm to *seqCV* described in Section 7.1.2 and progressively increases the training set data and keeps predicting the next round of games, only using the data that would have been available at the time of the game. This function calls another function named *calculate_BS*, which calculates the Brier score for a set of games in a test set given a training set to build the model. The sequential Brier scores for each of the models are summarised below in Table 15.

Model	Sequential Brier Score
BB	0.5226
NB	0.5232

Table 15: Sequential Brier Results

Therefore, from just looking at the Brier scores, there does not seem to be much difference between the two models, as they are equal up to 2 decimal places. Therefore, alternative methods might need to be explored to compare these models.

7.3 Rank Probability Score

Although the Brier score offers a better scoring rule than cross-validation in the case of assessing probabilistic football forecast models, it is not perfect. Constantinou & Fenton (2012) discuss more cases where the Brier score does not return the 'better' model. For example, consider the hypothetical scenarios shown in Table 16 and note that the Brier score does not take into account the ordering when the probability of unobserved outcomes are equal. In game number 4 in Table 16, both model A and B assign the same probability to the correct outcome. They also assign the same probability values for unobserved outcomes, but in a different order. But model A is more preferred since a draw is closer to a home win than an away win is. However, a Brier score calculation for this game would return the same value. Additionally, in match 5, although model B predicts the correct outcome at a higher probability, model A is more indicative of a home win, since there is only 5% of an away win, whereas model B suggests there is a 35% chance of an away win. However, again a calculation of the Brier score for this game would suggest model B is the preferred model. Constantinou & Fenton (2012) add to this and discuss other scoring rules available that have been used in other literature around the topic of assessing probabilistic football model, such as the *rank probability score*, which we will use here.

Match	Model	Pr(Home win)	Pr(Draw)	Pr(Away win)	Result	Brier Score	Better model
4	A	0.5	0.45	0.05	Home win	0.455	A
	B	0.5	0.05	0.45		0.455	
5	A	0.5	0.45	0.05	Home win	0.455	A
	B	0.55	0.1	0.35		0.335	

Table 16: Hypothetical probability estimates from two different models, A and B

Constantinou & Fenton (2012) state for for a single problem instance (i.e. a single match in our case), then the *rank probability score* (RPS) is defined as

$$RPS = \frac{1}{r-1}\sum_{i=1}^{r}\Big(\sum_{j=1}^{i}\hat{\pi}_j - \sum_{j=1}^{i}O_j\Big)^2$$

where r is the number of potential outcomes, $\hat{\pi}_j$ is the probability of the forecast and O_j is the observed outcome for the j-th possible event. O_j is defined as before where it takes the value 1 if event j occurred and 0 otherwise. The RPS represents the difference between the cumulative distributions of forecasts and the observations. By this definition, RPS increases if the cumulative distribution of the forecast probabilities differs more from the actual observed outcome and so a lower RPS indicates a better model. By using the RPS score, we are able to 'correctly' identify model A as the preferred model for prediction in each of the five matches given in Table 13 and Table 16.

Example

To illustrate how to calculate the rank probability score for the games shown in Table 16. In each game there are $r = 3$ possible outcomes: home win, draw and away win. We present the cumulative distributions of the forecasts and the observed outcomes as a vector in the columns below.

Match	Model	$\sum_{j=1}^{i=1,2,3} \hat{\pi}_j$	$\sum_{j=1}^{i=1,2,3} O_j$	RPS calculation	RPS
4	A	(0.5, 0.95, 1)	(1,1,1)	$\frac{1}{2}[(0.5-1)^2 + (0.95-1)^2 + (1-1)^2]$	0.12625
	B	(0.5, 0.55, 1)	(1,1,1)	$\frac{1}{2}[(0.5-1)^2 + (0.55-1)^2 + (1-1)^2]$	0.22625
5	A	(0.5, 0.95, 1)	(1,1,1)	$\frac{1}{2}[(0.5-1)^2 + (0.95-1)^2 + (1-1)^2]$	0.12625
	B	(0.55, 0.65, 1)	(1,1,1)	$\frac{1}{2}[(0.55-1)^2 + (0.65-1)^2 + (1-1)^2]$	0.1625

Table 17: Example calculations of RPS for hypothetical probabilities given in Table 16

Note that since $\sum_{j=1}^{3} \hat{\pi}_j$ and $\sum_{j=1}^{3} O_j$ both equal 1, the RPS formula can also be written as

$$RPS = \frac{1}{r-1} \sum_{i=1}^{r-1} \Big(\sum_{j=1}^{i} \hat{\pi}_j - \sum_{j=1}^{i} O_j \Big)^2$$

Since a lower RPS is better, then we can see that RPS now chooses model A as the superior model for prediction.

7.3.1 RPS Results

The function *calculate_RPS* is written in R and can be found in the Appendix in Section J.3. This function returns the total RPS for a set of games (the sum of the RPS scores for each game). To find the mean RPS per game, we need to divide by the number of games in the test set. To obtain a RPS score for each of the models we wish to assess, we use the *seqRPS* function in the Appendix (Section J.3), which again uses the same algorithm as *seqCV* and *seqBrier* to progressively increase the training set data to predict the next round of games. The rank probability scores for each of the models for the last 370 games of the season (first 10 games are used as the initial training set) are summarised below in Table 18.

Model	Total Sequential RPS Score	Average Sequential RPS Score per game
BB	65.88	0.178
NB	64.05	0.173

Table 18: Sequential Rank Probability Score (RPS) Results

Again, similar to the Brier score results found in Section 7.2.1, the rank probability scores for the two models are very close to each other, with the Negative Binomial model having a slightly better score.

The difficulty of choosing the 'best' model using scoring rules is evident here since the Brier score preferred Baio & Blangiardo's model (2010), while the RPS preferred the Negative Binomial. It is important to note that we cannot definitively conclude which model is superior and as Constantinou & Fenton (2012) note, in the absence of an agreed and appropriate type of scoring rule for football models, it will always be difficult to reach a consensus on whether a particular model is accurate or which of two or more competing models is the 'best'. However, the concern is that an inappropriate assessment of forecast accuracy may lead to inconsistencies, whereby one rule may prefer one model and another rule prefers an alternative model, as we have here.

Gneiting and Raftery (2007) note that in determining a good scoring rule for a particular model, one should recognise that the underlying measurement scale type of the outcomes for a specific problem should drive the type of scoring rule used. Constantinou & Fenton (2012) use the example of assessing a model to predict a winning lottery number. They note that although the possible outcomes appear to be an ordered set $\{1, 2, ..., 49\}$, the relevant scale type is only *nominal*, meaning that if the winning number is 10, then a prediction of 9 is no 'closer' than a prediction of 49, as they are both equally wrong and any scoring rule should capture this. In contrast, in football, the set of outcomes {H, D, A} must be considered on an *ordinal* scale, where H represents a home win, D a draw and A an away win, since the outcome D is closer to a H than A is to H. Therefore, if the result of the match is H, then the scoring rule should penalise the probability assigned to A more than the probability assigned to D. The scenarios shown in Table 16 illustrate these cases and the RPS satisfies all these scenarios by choosing the preferred model. In contrast, the Brier score fails to recognise that football outcomes should be on an ordinal scale and so its use to assess the forecast accuracy of football outcomes is inadequate, as the Brier score fails to determine the more accurate forecasting model in the scenarios presented in Table 16.

Therefore, it seems that the RPS is the more suitable scoring rule to assess models for the prediction of football games compared to the Brier score. However, we are not suggesting that the RPS is the only valid scoring rule for assessment, but we have shown several scenarios where the RPS correctly chooses the more accurate model. It is also important to note that such scoring rules only measure the forecast accuracy but does not measure how useful the model is for other applications. Therefore, we look at two more methods to assess the model by looking more closely at the applications of the models to specific problems: using the model as a basis of a betting model and to predict a league table.

7.4 Using the models for a betting tool

One of the main applications of building a predictive model for football is to use it for a betting model. We assess the usefulness of each model for betting by using the simple decision rule to place a *single* on the most favoured outcome. In betting, a *single* is just a bet that wins or loses depending on the outcome of one event. In contrast, an *accumulator* is a bet that combines several selections and wins only if all the parts of the bet win. Here we only use a very simple decision rule of betting on the outcome that has the highest probability according to the model. The Football-Data website (2018) also gives the odds offered by several bookmakers for each match, but here we will just use the odds offered by Bet365. We again avoid using data that would not be available at the time of placing the bet and use the 2016/17 Premier League season and try to predict the last 370 games of this season.

Note that this is a very simple decision rule and in more complicated betting models, the amount of money placed on each bet may vary depending on how certain the event will happen. Further note that this report should not be used for betting purposes and this part of this report is only to assess model performances of past data to give an indication of how well the model will perform in the future, not to guarantee the same results.

7.4.1 Betting Assessment Results

The function *calculate_PL*, which is written in R and can be found in the Appendix (Section J.4), takes a training set and test set to build a model in Stan and for each game in the test set, it will see if the model correctly guesses the outcome of the game. If a match is correctly guessed, then the profit from that match is equal to (wager $\times$ betting odds) $-$ wager, and if the model is incorrect then the profit from that match is just $-$wager, since the bet is lost. Here, *wager* is just the amount of money placed on the bet, and the function *calculate_PL* requires a wager to be passed. This is repeated for all matches in the test set and the profit for each game is returned in a vector. The function *seqBetting* (which can be found in the Appendix, Section J.4) allows us to find the overall profit from this decision rule for a given model. The overall profit return from this simple betting rule for each model (if the wager for each game is £10) can be found below in Table 19.

Model	Profit (in £s)
BB	909.80
NB	1031.60

Table 19: Profit from using the models for betting

We can see here that the Negative Binomial model outperforms Baio & Blangiardo's model (2010) significantly. Recall from 7.1.2, the Negative Binomial distribution correctly predicted 5 more games, hence this will contribute to the higher profit for the Negative Binomial model. Also note that the two models do not always agree on the most probable outcome. The breakdown of the profit/loss returned by the games for each model is shown in a frequency table in Table 20 and so we can see that Baio & Blangiardo's model would have correctly predicted more games that returned a profit greater than £50, whereas the Negative Binomial model did not get predict the same outcome for some of these games. However, the Negative Binomial model was more consistent in correctly predicting games that returned less profit (between £0 to £10) and more in between £20 and £30.

Profit/Loss (PL)	Frequency
-10.00 (lost bet)	134
$0 \leq PL < 10$	153
$10 \leq PL < 20$	66
$20 \leq PL < 30$	9
$30 \leq PL < 40$	2
$40 \leq PL < 50$	3
$PL \geq 50$	3

(a) Baio & Blangiardo's model (2010)

Profit/Loss	Frequency
-10.00 (lost bet)	129
$0 \leq PL < 10$	159
$10 \leq PL < 20$	57
$20 \leq PL < 30$	17
$30 \leq PL < 40$	2
$40 \leq PL < 50$	5
$PL \geq 50$	1

(b) Negative Binomial model

Table 20: Frequency of each profit/loss for each model in £s

Further, Figure 20 shows the profit for each model over the course of the season.

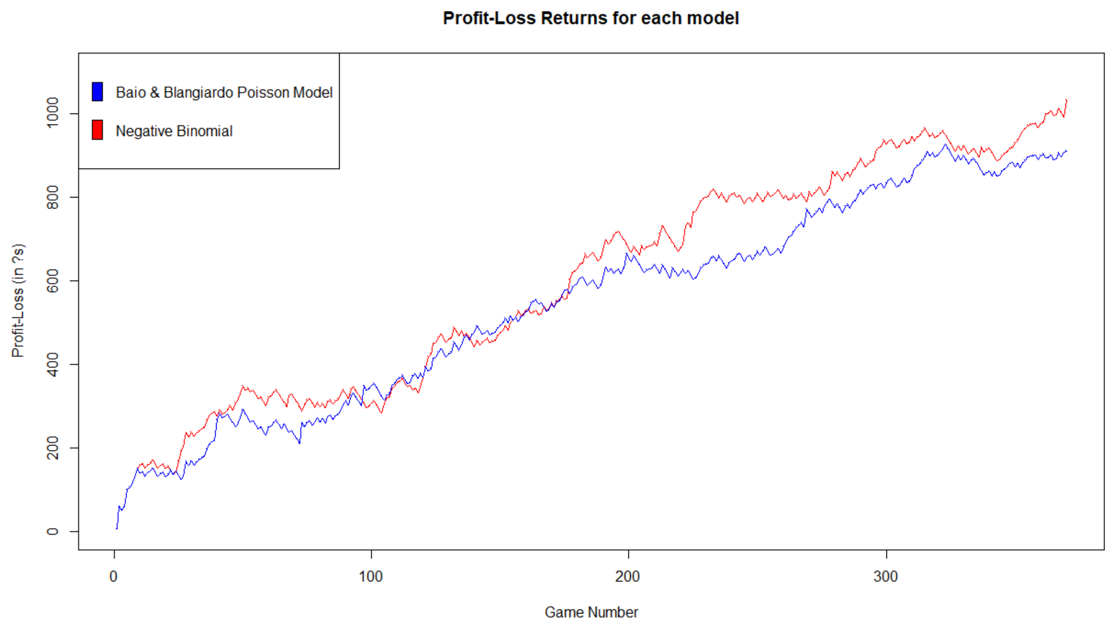


Figure 20: Overall profit returns from each model in £s

Therefore, from this analysis it seems that the Negative Binomial may be more useful as a basis of a betting model. Although we cannot guarantee that the model will perform like this in the future, by back testing the model on past data, this gives us a good indication of how they will perform for future games.

7.5 Using the models to predict a league table

Baio & Blangiardo (2010) used their model to predict the Serie A 1992-93 league table as a means to assess their model. To do this, they used the model to obtain a posterior sample for their model parameters and then used these to predict the same games in that season to create a league table. Hence their model assessment method is very biased, since they tested the model with the same dataset that was used to train the model.

Instead of using this method, we will use the same process as we did in calculating the sequential probability scores and so we only predict games using data that was available at the time. So far, we have only predicted the outcomes of games, but to predict a league table, we will need to predict the scores for the games. To do this, the posterior mode (or *MAP estimate*) of the number of goals scored is used to predict the number of goals scored by each team.

The function *get_table* can be found in the Appendix (Section J.5), which returns a league table given a set of results. Recall that in the Premier League, the points system is as follows: 3 points are awarded to the match winner, 1 point is awarded to both teams in the event of a draw and 0 points are awarded to team if they lose. The observed table from the 2016/17 Premier League season can be found in Section K in the Appendix. In addition, a function called *track_points_progress* can be found in Section J.5 in the Appendix, which returns a matrix showing the cumulative points gained for each team given a set of results.

7.5.1 Predicting League Table Results

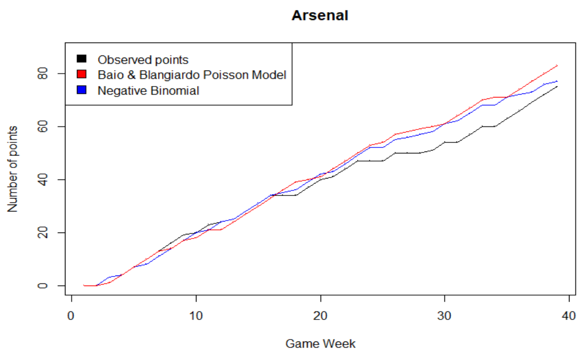
To assess model performance in predicting a league table, the function *predict_table* returns both a predicted league table and a matrix showing the predicted cumulative points from a model by using *get_table* and *track_points_progress* after predicting all the seasons games. This function progressively predicts each game in the league after a minimum number of games, which is passed in this function. Rather than choosing the highest probable event, since a league table also records the goal difference for each team, the posterior mode of goals scored by each team is used to obtain a predicted score line. After obtaining a set of results, the function calls *get_table* and *track_points_progress* to obtain a predicted league table for the model and a find each teams points progression through out the season.

The predicted league tables from each of these models can be found in the Appendix (Section K). In Figure 21, we plot the observed cumulative points through the season for each team in black, the predicted results from Baio & Blangiardo's model (2010) in red and the blue lines represent the predictions from the Negative Binomial model.

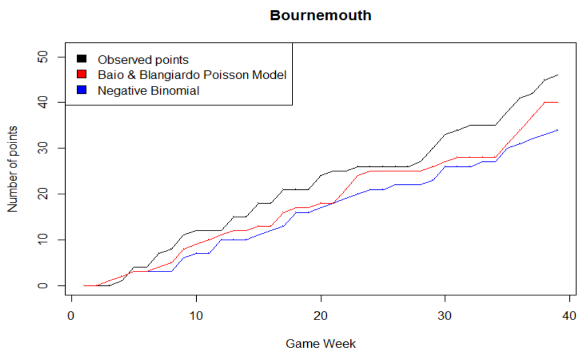
Therefore, rather than using the whole dataset to obtain posterior samples for our model parameters and then use these to predict games, this method to assess model performance only uses data that would have been available at the time for each game. As a result, this give a fairer assessment of the models than Baio & Blangiardo's method (2010) does, where data is used twice to fit and test the model.

For some teams (Hull City, Leicester, Manchester City, Stoke, Watford, West Brom), we can see that the Negative Binomial distribution seems to perform better than Baio & Blangiardo's model (2010), since the blue line offers a closer fit to the black line. However, there are some notable teams where the Negative Binomial performed particularly poor in, for example, Middlesbrough, Swansea and West Ham. Despite these particular teams, the Negative Binomial generally performed well to predict a teams cumulative points over the season and therefore to predict a league table. Further, by looking at the predicted league tables in Section K in the Appendix, we can see that the Negative Binomial model correctly predicted 8 team final league positions (the top 7 positions and 15th place), whereas Baio & Blangiardo's model (2010) only got 3 final league positions correct. Hence, by using the posterior mode to predict scores, the Negative Binomial generally outperformed the Poisson model by Baio & Blangiardo (2010).

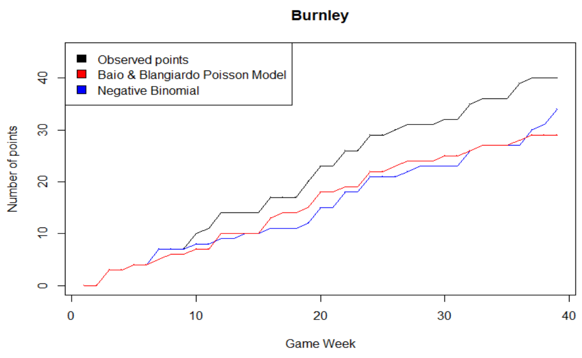
Note that we are predicting each game week by week until the end. In practice, we may predict a league table from these models half way through the season to simulate the rest of the remaining games. In this case, we keep adding our predicted scores to the training set progressively, rather than the observed games, since they would be unknown at that time. We also do this so that the team parameters do not remain static for the remainder of the season but instead they rise or fall based on the simulated matches that are played. However, this could be computationally heavy, since it would be necessary to repeat these simulations to obtain an average. In doing this method, we would obtain several predicted league tables, which we could then estimate the probabilities that a team would finish above a certain position by counting the number of tables where they finished above this position. In practice, one may want to do this to find the probability of survival in the league (not relegating) or to finish in top 4 to obtain a Champions League spot. We could also estimate probabilities of finishing in a specific position in the league by counting the number of times the team finished there in the simulated league tables. This could be a variation on this method to assess model performance, but here we only focus on predicting the scores of the games progressively through the season.



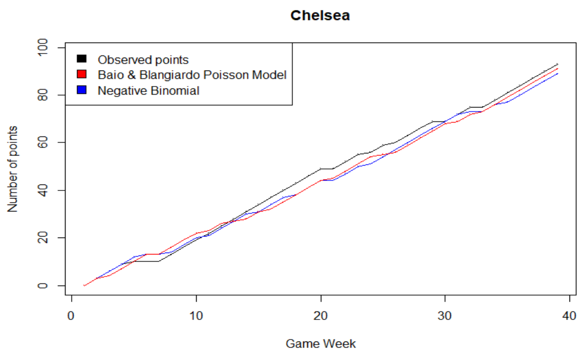
(a) Arsenal



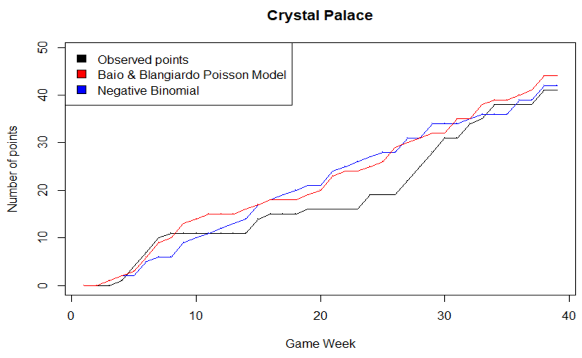
(b) Bournemouth



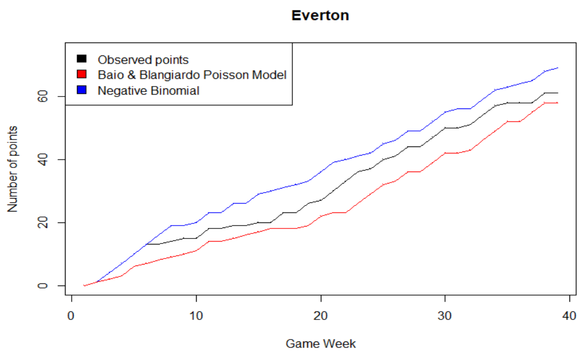
(c) Burnley



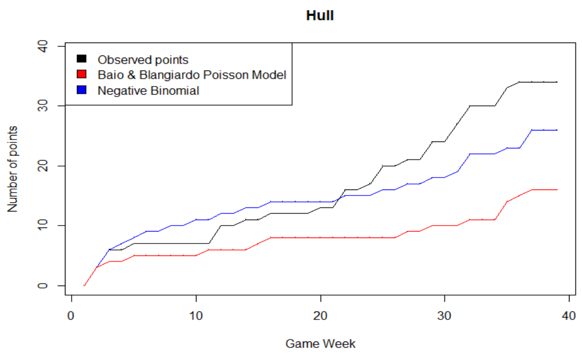
(d) Chelsea



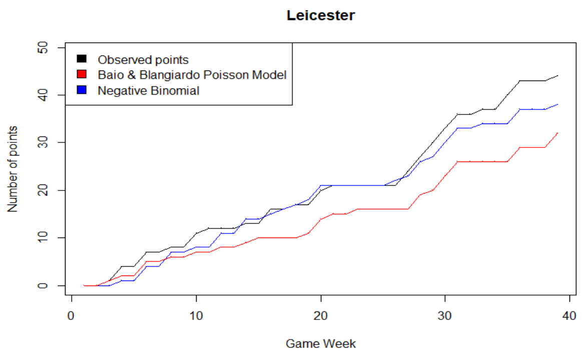
(e) Crystal Palace



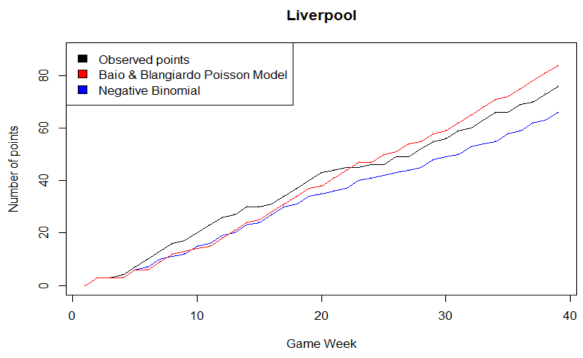
(f) Everton



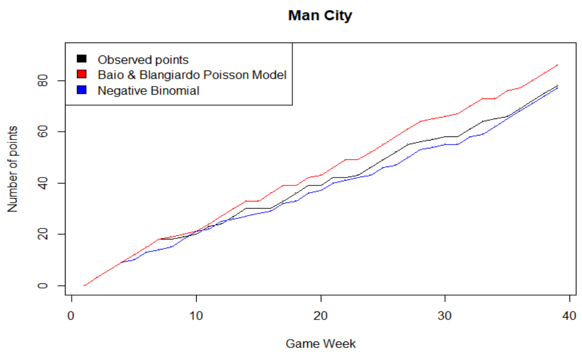
(g) Hull City



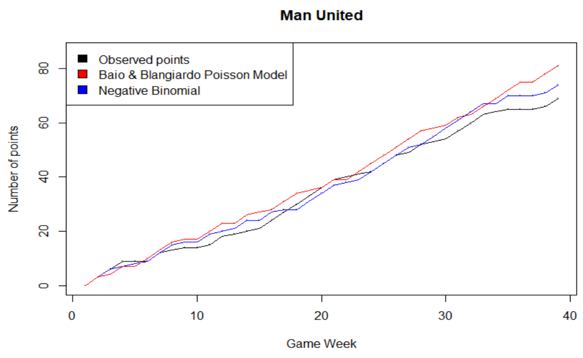
(h) Leicester



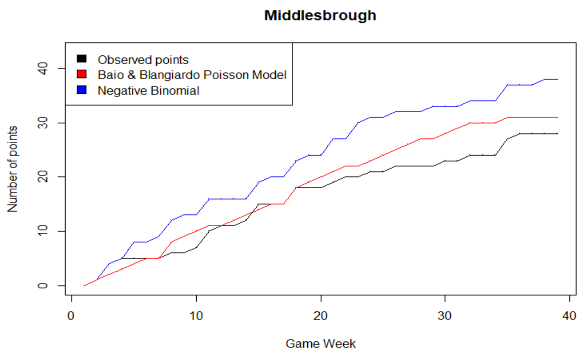
(i) Liverpool



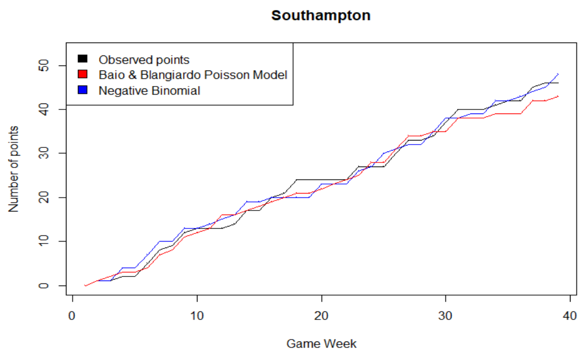
(j) Manchester City



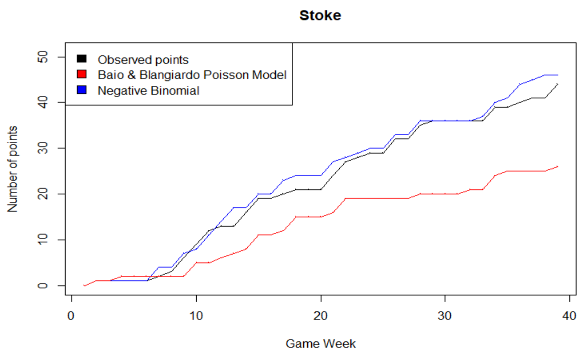
(k) Manchester United



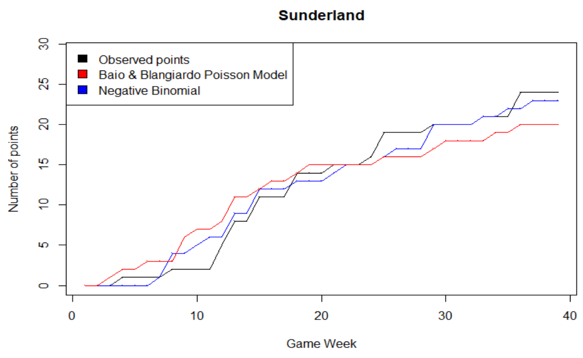
(l) Middlesbrough



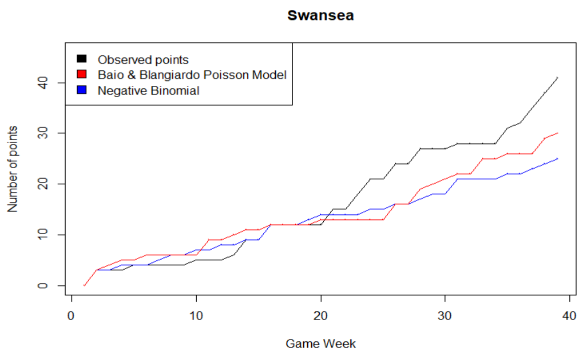
(m) Southampton



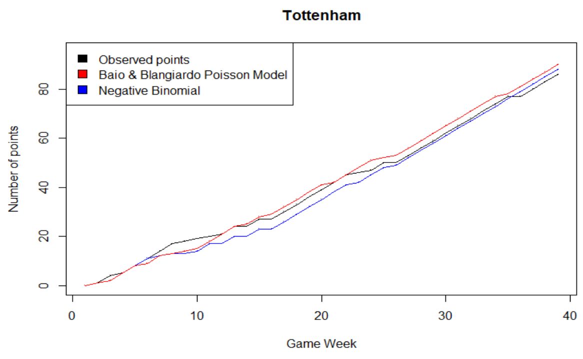
(n) Stoke City



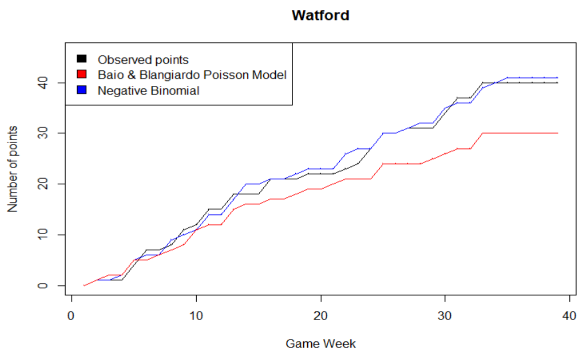
(o) Sunderland



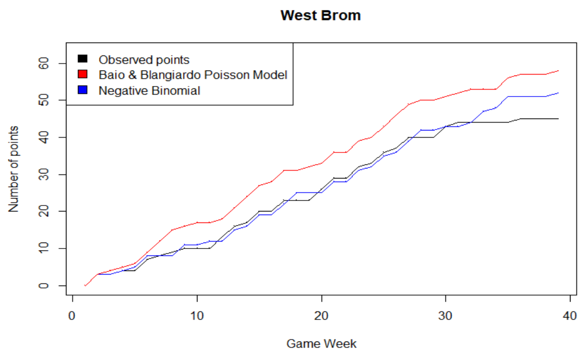
(p) Swansea City



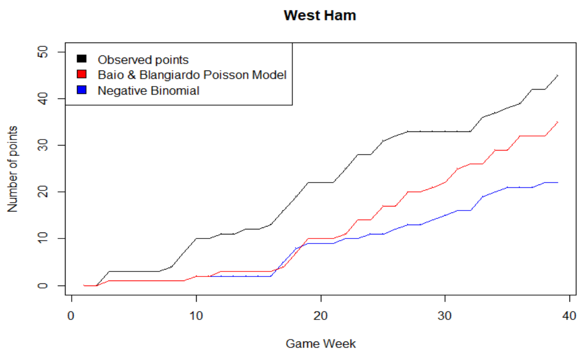
(q) Tottenham Hotspurs



(r) Watford



(s) West Bromwich Albion



(t) West Ham United

Figure 21: Predictive validation of the Negative Binomial model in comparison with Baio & Blangiardo's model (2010)

8 Conclusion

In this report, a Bayesian hierarchical model has been developed and analysed to determine if the model predicts future football matches well. We have implemented a model discussed by Baio & Blangiardo (2010) and developed an alternative model with different parameters and one that uses the Negative Binomial distribution to model the number of goals scored by each side. In relevant literature, the use of the Negative Binomial distribution to model goals has been largely ignored and a Poisson distribution is generally assumed. While we found that the Poisson distribution was a reasonable fit to the data, improvements could be made. In Section 7, we discussed several different methods to analyse predictive performance such as cross validation, the Brier score, rank probability score (RPS) and also in predicting a league table and assessing its use in a betting model. While the assessment methods for predicting a league table and its use in a betting are specific to football models, cross-validation, Brier score and RPS can all be applied in any probabilistic model. The Brier score is more useful in cases where we are interested in the probability at which we make our predictions, and RPS takes this further and is important in situations where there is a ranked outcome, as there is with football. From our results in Sections 7.1 - 7.3, the scores for the two models discussed were very close, but when applying the models as a betting tool and to predict a league table, the Negative Binomial model outperformed Baio & Blangiardo's model (2010).

Therefore, to conclude, the Negative Binomial Bayesian hierarchical model developed throughout this report performs well in predicting football matches and can also be used to obtain estimates to attack and defence parameters that can be used to understand how each team plays. In this report, we focused on using this model to predict English Premier League games, so to further assess this model, one can try to apply this model to other countries and leagues to see if a Negative Binomial distribution can provide a good fit to goals data in competitions outside the Premier League.

Further, note that the model only uses goals to obtain estimates for parameters for each team and goals may not be the best indicator for how well a team may perform. For instance, a team may have a run of games where they should have scored more but were unlucky and so would have a low attack parameter from those games, but in reality, they played very well. Therefore, since goals may not necessarily be the best indicator of team performance, one may look for more data to help add information on underlying team strengths and to better predict the future performances for each team. For example, football data experts Opta calculate *expected goals* (or xG), which is a metric which assess every chance a team has and is a way of assigning a quality value to every attempt at goal (Opta, 2017).

There are many different expected goals models, but Opta have analysed over 300,000 shots to calculate the chance of an attempt being scored in a specific position on the picture during a particular phase of play (Stanton, 2017). It is based on several variables such as the assist type, shot angle, distance from goal and several others. Summing up the total xG for a team for a game gives the expected goals they should have scored in that game. For very good teams, they will usually score more than the average team will and for poorer teams, they may score lower than their expected goals. We can interpret this statistic as how many goals an average team would have scored given the chances created. It can also help us determine the quality of chances a team produces to give us a better indication of how well a team plays in general. If a team is consistently creating a lot of very good chances but not scoring, we may be missing the fact that they are playing very well but just not scoring them. Therefore, a possible extension of this model is to incorporate more data that can help obtain more accurate estimates to team performance levels.

Moreover, the model currently ignores possible factors that can have an effect on a team's performance or chances of winning, such as injuries (or resting) of star players during the season, the effect of fatigue, managerial changes, distance travelled by away team and the form of the teams. Therefore, another possible extension of this model is to add these factors to the model. For instance, Joseph *et al.* (2006) constructed a Bayesian Network to predict the outcome of football matches only including Tottenham Hotspurs, which included features such as the presence or absence of particular key players. Hence, although the Negative Binomial model performed well in our assessments, there is a lot of room for improvement, which may possibly be achieved by developing the model further to account for psychological factors and particularly to factors concerning the form, the fitness of teams and the presence or absence of particular players. It is important to note when adding these to the model, we must also determine whether they have a significant effect on team performances and their chances of winning a football match.

Therefore, the next steps for this project would be to try add more factors and parameters to the model with the aim to better represent the dynamics that contribute to teams winning football matches. However, the Negative Binomial model proposed here offers a good model performance with simple data that is readily accessible.

References

Anderson, C. & Sally, D. 2014. *The Numbers Game.* London: Penguin Group.

Arlot, S. & Celisse, A. 2010. A survey of cross-validation procedures for model selection. *Statistics Surveys.* **4**, pp. 40-79.

Baio, G. & Blangiardo, M. 2010. Bayesian hierarchical model for the prediction of football results. *Journal of Applied Statistics.* **37**(2), pp. 253-264.

Barp, A., Briol, F.X., Kennedy, A.D. & Girolami, M. 2017. [Pre-print]. Geometry & Dynamics for Markov Chain Monte Carlo. *arXiv.* [Online]. [Accessed 11 April 2018]. Available from:
`https://arxiv.org/abs/1705.02891`

Ben-Naim, E., Vazquez, F. & Redner, S. 2005. [Pre-print]. What is the most competitive sport?. *arXiv.* [Online]. [Accessed 25 January 2018]. Available from:
`https://arxiv.org/abs/physics/0512143`

Betancourt, M.J. 2017. [Pre-print]. A Conceptual Introduction to Hamiltonian Monte Carlo. *arXiv.* [Online]. [Accessed 2 April 2018]. Available from:
`https://arxiv.org/abs/1701.02434`

Betancourt, M.J., Byrne, S., Livingstone, S. & Girolami, M. 2014. [Pre-print]. The Geometric Foundations of Hamiltonian Monte Carlo. *arXiv.* [Online]. [Accessed 2 April 2018]. Available from:
`https://arxiv.org/abs/1410.5110`

Betancourt, M.J. & Girolami, M. 2013. [Pre-print]. Hamiltonian Monte Carlo for Hierarchical Models. *arXiv.* [Online]. [Accessed 17 February 2018]. Available from:
`https://arxiv.org/abs/1312.0906`

Betancourt, M.J. & Stein, L.C. 2011. [Pre-print]. The Geometry of Hamiltonian Monte Carlo. *arXiv.* [Online]. [Accessed 22 February 2018]. Available from:
`https://arxiv.org/abs/1112.4118`

Bolstad, W.B. 2007. *Introduction to Bayesian Statistics.* 2nd ed. New Jersey: John Wiley & Sons.

Box, G. E. P. 1976. Science and Statistics. *Journal of the American Statistical Association.* **71**(356), pp. 791-799.

Box, G. E. P. & Tiao, G. C. 1973. *Bayesian Inference in Statistical Analysis.* Boston: Addison-Wesley Pub. Co.

Brier, G.W. 1950. Verification of forecasts expressed in terms of probability. *Monthly Weather Review.* **78**(1), pp. 1-3.

Chatfield, C. & Collins, A.J. 1980. *Introduction to Multivariate Analysis.* 1st ed. Florida: Chapman & Hall.

Constantinou, A. & Fenton, N.E. 2012. Solving the problem of inadequate scoring rules for assessing probabilistic football forecasting models. *Journal of the Royal Statistics Society.* Series C: Applied Statistics.

Cook, I. & Upton, G. 2014. *A Dictionary of Statistics.* 3rd ed. Oxford: Oxford University Press.

Dahiru, T.P. 2008. *p*-value, a true test of statistical significance? A cautionary note. *Annals of Ibadan Postgraduate Medicine.* **6**(1), pp. 21-26.

Davis, P.J. & Rabinowitz, P. 2007. *Methods of Numerical Integration.* New York: Dover Publications.

Deloitte. 2017. *Annual Review of Football Finance.* [Online]. [Accessed 22 November 2017]. Available from:
`https://www2.deloitte.com/uk/en/pages/sports-business-group/articles/annual-review-of-football-finance.html`

Diestel, R., 2006. *Graph Theory.* 3rd ed. Berlin: Springer Science & Business Media.

Dixon, M. & Coles, S. 1997. Modelling Association Football Scores and Inefficiencies in the Football Betting Market. *Journal of the Royal Statistical Society.* **46**(2), pp. 265-280.

Economist. 2011. *Ranking sports' popularity.* [Online]. [Accessed 24 March 2018]. Available from:
`https://www.economist.com/blogs/gametheory/2011/09/ranking-sports%E2%80%99-popularity`

Fishman, G.S. 1996. *Monte Carlo: Concepts, Algorithms and Applications.* New York: Springer-Verlag.

Football Data. 2017. Football Data Website. [Online]. [Accessed 04 September 2018]. Available from:
`http://www.football-data.co.uk/`

Gabry, J., Stan Development Team, Andreae, M., Betancourt, M., Carpenter, B., Gao, Y., Gelman, A., Goodrich, B., Lee, D., Song, D. & Trangucci, R. 2017. Shinystan: Interactive Visual and Numerical Diagnostics and Posterior Analysis for Bayesian Models. [Online]. [Accessed 19 November 2017]. Available from:
`https://cran.r-project.org/web/packages/shinystan/index.html`

Gelman, A., Carlin, J.B., Hal, S.S., David, B.D., Aki, V., and Donald, B.R. 2014. *Bayesian Data Analysis.* 3rd ed. Florida: Taylor & Francis Group.

Gelman, A & Hoffman, M.D. 2011. [Pre-print]. The No-U-Turn-Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research.* [Online]. [Accessed 22 February 2018]. Available from:
`https://arxiv.org/abs/1111.4246`

Geyer, C.J. 1992. Practical Markov Chain Monte Carlo. *Statistical Science.* **7**(4), pp. 473-483.

Geyer, C.J. 2011. Introduction to Markov Chain Monte Carlo. In: Brooks, S., Gelman, A., Jones, G.L., Meng, X.L. ed(s). *Handbook of Markov Chain Monte Carlo.* Florida: Taylor & Francis Group, pp. 3-47.

Gill, J. 2014. *Bayesian Methods: A social and Behavioural Science Approach.* 3rd ed. Florida: Taylor & Francis Group.

Gneiting, T. & Raftery, A. 2007. Strictly Proper Scoring Rules, Prediction, and Estimation. *Journal of the American Statistical Association.* **102**(477), pp. 359-378.

Goodman, S.N. 1999. Toward evidence-based medical statistics. 2: The Bayes Factor. *Annals of Internal Medicine.* **130**(12), pp. 1005-1013.

Hall, D.B. 2000. Zero-Inflated Poisson and Binomial Regression with Random Effects: A Case Study. *Biometrics.* **56**, pp. 1030-1039.

Hartigan, J.A. 1983. Asymptotic Normality of Posterior Distributions. In: *Bayes Theory.* New York: Springer.

Hastings, W.K. 1970. Monte Carlo Sampling Methods Using Markov Chains and Their Applications. *Biometrika.* **57**(1), pp. 97-109.

Howie, D. 2002. *Interpreting probability. Controversies and development in the early twentieth century.* Cambridge: Cambridge University Press.

Hughes, G. & Topp, C.F.E. 2015. Probabilistic Forecasts: Scoring Rules and Their Decomposition and Diagrammatic Representation via Bregman Divergences. *Entropy.* **17**, pp. 5450-5471.

Investopedia. 2018. *Look-Ahead Bias.* [Online]. [Accessed 29th March 2018]. Available from:
`https://www.investopedia.com/terms/l/lookaheadbias.asp`

Joseph, A., Fenton, N.E. & Neil, M. 2006. Predicting football results using Bayesian nets and other machine learning techniques. *Knowledge-Based Systems.* **19**(7), pp. 544-553.

Karlis, D. & Ntzoufras, I. 2000. On modelling soccer data. *Student.* **3**(4), pp. 229-244.

Karlis, D. & Ntzoufras, I. 2003. Analysis of sports data using bivariate Poisson models. *Journal of the Royal Statistical Society.* **52**(3), pp. 381-393.

Lee, A. 1997. Modelling Scores in the Premier League: Is Manchester United Really the Best?. *Chance.* **10**(1), pp. 15-19.

Lee, P.M. 2004. *Bayesian Statistics: An Introduction.* 3rd ed. London: John Wiley & Sons.

Lewis, M. 2003. *Moneyball: The Art of Winning an Unfair Game.* New York: W. W. Norton & Company.

Lin, M., Lucas Jr., H.C. & Shmueli, G. 2013. Too Big to Fail: Large Samples and the p-value Problem. *Informations Systems Research.* **24**(4), pp. 906-917.

Maher, M. 1982. Modelling association football scores. *Statistica Neerlandica.* **36**. pp. 109-118.

McCullagh, P. 2002. What is a Statistical Model?. *The Annals of Statistics.* **30**(5), pp. 1225-1267.

McLachlan, G. & Peel, D. 2004. *Finite Mixture Models.* New York: John Wiley & Sons.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. 1953. Equation of state calculation by fast computing machines. *J. Chem Phys.* **21**(1087), pp. 1087-1092.

Miller, T. 2015. *Sports Analytics and Data Science.* New Jersey: Pearson Education, Inc.

Morris, D.E., Oakley, J.E. & Crowe, J.A. 2014. [Online]. A web-based tool for eliciting probability distributions from experts. *Environmental Modelling & Software.* **52**, pp. 1-4. Available from:
`http://dx.doi.org/10.1016/j.envsoft.2013.10.010.`

Murray, I. 2007. *Advances in Markov chain Monte Carlo methods.* Ph.D. thesis, University College London.

Neal, R. 2011. MCMC using Hamiltonian dynamics. In: Brooks, S., Gelman, A., Jones, G.L., Meng, X.L. ed(s). *Handbook of Markov Chain Monte Carlo.* Florida: Taylor & Francis Group, pp. 113-160.

Opta. 2017. *Opta's event definitions.* [Online]. [Accessed 23 April 2018]. Available from: `https://www.optasports.com/news/optas-event-definitions/`

Papadakis, M., Tsagris, M., Dimitriadis, M., Fafalios, S., Tsamardinos, I., Fasiolo, M., Borboudakis, G., Burkardt, J., Zou, C. & Lakiotaki, K. Rfast Package Documentation. [Online]. [Accessed 6 October 2017]. Available from:

https://cran.r-project.org/web/packages/Rfast/Rfast.pdf

Pearl, J. 2009. *Causality: models, reasoning, and inference.* 2nd ed. New York: Cambridge University Press.

Pearl, J. 1985. Bayesian networks: A model of self-activated memory for evidential reasoning. *In Proceedings, Cognitive Science Society.* University of California, Irvine, pp. 329-334.

Pearl, J., Geiger, D. & Verma, T. 1989. Conditional independence and its representations. *Kybernetika.* **25**(7), pp. 33-44.

Perkins, J. & Wang, D. 2004. A Comparison Of Bayesian And Frequentist Statistics As Applied In A Simple Repeated Measures Example. *Journal of Modern Applied Statistical Methods.* **3**(1), pp. 227-233.

Reep, C. & Benjamin, B. 1968. Skill and Chance in Association Football. *Journal of the Royal Statistical Society A.* **131**(4), pp. 623-629.

The R Core Team. 2017. A Language and Environment for Statistical Computing. [Online]. [Accessed 2 October 2017]. Available from:
https://cran.r-project.org/doc/manuals/r-release/fullrefman.pdf

Rasmussen, C.E. & Ghahramani, Z. 2002. Bayesian Monte Carlo. *Proceedings of the 15th International Conference on Neural Information Processing Systems*, pp. 505-512.

Ravenzwaaij, D.V., Cassey, P. & Brown, S.D. 2018. A simple introduction to Markov Chain Monte-Carlo sampling. *Psychonomic Bulletin & Review.* **25**(1), pp. 143-154.

Rice, J.R. 1995. *Mathematical Statistics and Data Analysis.* 2nd ed. California: Wadsworth Publishing Company.

Robert, C.P. 2010. [Pre-print]. Bayesian computational methods. *arXiv.* [Online]. [Accessed 22 February 2018]. Available from:
https://arxiv.org/abs/1002.2702

Robert, C.P. & Casella, G. 2004. *Monte Carlo Statistical Methods.* 2nd ed. New York: Springer.

Roberts, G.O. 1996. Markov chain concepts related to sampling algorithms. In: Gilks, W.R. Gilks, Richardson, S. & D.J. Spiegelhalter. ed(s). *Markov chain Monte Carlo in practice.* London: Chapman & Hall, pp. 45-54.

Stan Development Team. 2018. RStan: the R interface to Stan. [Online]. [Accessed 04 January 2018]. Available from:
https://cran.r-project.org/web/packages/rstan/index.html

Stan Development Team. 2017. User's Guide and Reference Manual. [Online]. [Accessed 15 October 2017]. Available from:
`http://mc-stan.org/users/documentation/`

Stanton, J. 2017. *Premier League: 'Expected goals' tells us whether a player really should have scored.* [Online]. [Accessed 23 April 2018]. Available from:
`https://www.bbc.co.uk/sport/football/40699431`

Voss, J. 2014. *An Introduction to Statistical Computing - A Simulation-based Approach.* 1st ed. West Sussex: John Wiley & Sons.

Walker, A.M. 1969. On the Asymptotic Behaviour of Posterior Distributions. *Journal of the Royal Statistics Society. Series B.* **31**(1), pp. 80-88.

Weinzierl, S. 2000. Introduction to Monte Carlo Methods. *arXiv.* [Online]. [Accessed 7 March 2018]. Available from:
`https://arxiv.org/abs/hep-ph/0006269`

Xu, Q.S. & Liang, Y.Z. 2000. Monte Carlo cross validation. *Chemometrics and Intelligent Laboratory Systems.* **56**(1), pp. 1-11.

A Example of Rejection Sampling

```
### Plotting densities
curve(dbeta(x, 4, 2), 0, 1, col = "red", ylab=expression(paste(pi(theta)
    )), xlab=expression(theta), main=expression(paste('Plot of Beta(4,2)
    density and Uniform(0,1) density')), lwd=2.5)
curve(dunif(x, 0, 1), 0, 1, add=T, col = "blue", lwd=2.5)
legend("topleft",legend=c('Beta(4,2)', 'Uniform(0,1)'), lty=c(1,1), lwd=
    c(2.5,2.5),col=c('blue','red'))

### Rejection Sampling
i=1
N=10000
iterations=0
theta=NULL
while (i<N) {
  iterations=iterations+1
  # sample theta from a uniform distribution
  sample = runif(1,0,1)
  # evaluate the probabiltiy of beta(4,2) at sampled value
  target = dbeta(sample, 4, 2)
  # seeing if we want to keep this value or not
  if (runif(1,0,1) < target/3) {
    theta[i] = sample
    i=i+1
  }
}
paste('Acceptance rate: ', N/iterations)
hist(theta, freq = F, col = "grey", breaks = 100, ylab=expression(paste(
    pi(theta))), xlab=expression(theta), main=expression(paste('Histogram
     of accepted values of ', theta, ' and Beta(4,2) density')))
curve(dbeta(x, 4, 2), 0, 1, add =T, col = "red")
```

B Example of Metropolis-Hastings Algorithm

```
### Using the Gamma(4,2) Proposal
N = 10000;
lpost = NULL
lpost[1] = 10
for (i in 2:N){
  # we use a Gamma(4,2) proposal distribution
  lstar = rgamma(1,4,2)
  # we evaluate the log posterior density for the proposed value
  ratiotop = 19*log(lstar) - 7*lstar
  # we evaluate the log posterior density for the previous value
  ratiobot = 19*log(lpost[i-1]) - 7*lpost[i-1]
  # we evaluate the log proposal density for the previous value
  qtop = log(dgamma(lpost[i-1],4,2))
```

```r
  # we evaluate the log proposal density for the proposed value
  qbot = log(dgamma(lstar,4,2))

  # Here, we used the log densities to avoid computational overflows or
    underflows to avoid exponents
  # If we were to use the regular densities, the variables set would
    have been:
  #ratiotop = (lstar^(19))*exp(-7*(lstar))
  #ratiobot = (lpost[i-1]^(19))*exp(-7*(lpost[i-1]))
  #qtop = dgamma(lpost[i-1],4,2)
  #qbot = dgamma(lstar,4,2)

  # now, we decide if we keep it or not
  U = runif(1,0,1)
  if (log(U)<((ratiotop + qtop)-(ratiobot + qbot))){
    lpost[i] = lstar
  } else { lpost[i] = lpost[i-1] }
}
plot(lpost, type='l', xlab="t", ylab=expression(lambda), main=expression
    (paste('MCMC Trace Plot for ', lambda)))
mean(lpost[1000:N])
median(lpost[1000:N])

### Using the Folded Normal Proposal
N = 10000
lpost = NULL
lpost[1] = 10
sigma2 = 10
for (i in 2:N){
  # we use a folded normal distribution for the proposal distribution
  lstar = abs(rnorm(1,lpost[i-1],sqrt(sigma2)))
  # we evaluate the log posterior density for the proposed value
  ratiotop = 19*log(lstar) - 7*lstar
  # we evaluate the log posterior density for the previous value
  ratiobot = 19*log(lpost[i-1]) - 7*lpost[i-1]
  # we evaluate the log proposal density for the previous value
  qtop = log(abs(dnorm(lpost[i-1],lpost[i-1],sqrt(sigma2))))
  # we evaluate the log proposal density for the proposed value
  qbot = log(abs(dnorm(lstar,lpost[i-1],sqrt(sigma2))))

  # now, we decide if we keep it or not
  U = runif(1,0,1)
  if (log(U)<((ratiotop + qtop)-(ratiobot + qbot))){
    lpost[i] = lstar
  } else { lpost[i] = lpost[i-1] }
}
plot(lpost, type='l', xlab="t", ylab=expression(lambda), main=expression
    (paste('MCMC Trace Plot for ', lambda)))
```

```
mean(lpost[1000:N])
median(lpost[1000:N])
```

C Example of Hamiltonian Monte Carlo Algorithm

```
# Example for 10000 samples and different number of leapfrog steps (Just
    change the value of L)

N = 10000;
lpost = NULL
lpost[1] = 10
for (i in 2:N){
  # first we sample a new momentum variable, which has normal(0, sigma2)
    distribution
  sigma2 = 100
  phi = rnorm(1, 0, sqrt(sigma2))
  # we need to keep a variable that is phi before to use in the
    acceptance ratio
  phi_before = phi
  # we need to have a variable lambda to update in the leapfrog steps
  lambda = lpost[i-1]
  # in order to update the values, we need to set the size epsilon and
    number of leapfrog steps L
  epsilon = 0.5
  L = 100
  for (j in 1:L) {
    phi = phi + 0.5*epsilon*((19/lambda) - 7)
    lambda = lambda + (epsilon/sigma2)*phi
    phi = phi + 0.5*epsilon*((19/lambda) - 7)
  }
  # now lambda = lambda* and phi = phi*
  # we evaluate the log posterior density for the proposed value for
    lambda
  ratiotop = 19*log(lambda) - 7*lambda
  # we evaluate the log posterior density for the previous value for
    lambda
  ratiobot = 19*log(lpost[i-1]) - 7*lpost[i-1]
  # we evaluate the log proposal density for the previous value for phi
  phitop = log(dnorm(phi, 0, sqrt(sigma2)))
  # we evaluate the log proposal density for the proposed value for phi
  phibot = log(dnorm(phi_before, 0, sqrt(sigma2)))

  # now, we decide if we keep it or not
  U = runif(1,0,1)
  if (log(U)<((ratiotop + phitop)-(ratiobot + phibot))){
    lpost[i] = lambda
  } else { lpost[i] = lpost[i-1] }
}
```

```
plot(lpost, type='l', xlab="t", ylab=expression(lambda), main=expression
    (paste('MCMC Trace Plot for ', lambda)))
mean(lpost[1000:N])
median(lpost[1000:N])

### Example for one update - no need to do any loops

lpost = NULL
lpost[1] = 10
sigma2 = 100
phi = rnorm(1, 0, sqrt(sigma2))
# we need to keep a variable that is phi before to use in the acceptance
     ratio
phi_before = phi
# we need to have a variable lambda to update in the leapfrog steps
lambda = lpost[i-1]
# in order to update the values, we need to set the size epsilon and
    number of leapfrog steps L
epsilon = 0.5
# performing one leapfrog step
phi = phi + 0.5*epsilon*((19/lambda) - 7)
p = c(phi_before, phi) # keeping track of phi
lambda = lambda + (epsilon/sigma2)*phi
phi = phi + 0.5*epsilon*((19/lambda) - 7)
p = c(p, phi)# keeping track of phi
l=c(lpost[i-1], lambda) # keeping track of lambda
# now lambda = lambda* and phi = phi*
# we evaluate the log posterior density for the proposed value for
    lambda
ratiotop = 19*log(lambda) - 7*lambda
# we evaluate the log posterior density for the previous value for
    lambda
ratiobot = 19*log(lpost[i-1]) - 7*lpost[i-1]
# we evaluate the log proposal density for the previous value for phi
phitop = log(dnorm(phi, 0, sqrt(sigma2)))
# we evaluate the log proposal density for the proposed value for phi
phibot = log(dnorm(phi_before, 0, sqrt(sigma2)))
# now, we decide if we keep it or not
U = runif(1,0,1)
if (log(U)<((ratiotop + phitop)-(ratiobot + phibot))){
  lpost[i] = lambda
} else { lpost[i] = lpost[i-1] }

# plotting the proposed values for lambda an phi from the leapfrog steps
plot(x=c(0,1), l, xlab="t", ylab=expression(lambda), main=expression(
    paste('MCMC Trace Plot for ', lambda)), pch=16)
lines(x=c(0,1), l)
plot(x=c(0,0.5,1), p, xlab="t", ylab=expression(phi), main=expression(
    paste('MCMC Trace Plot for ', phi)), pch=16)
lines(x=c(0,0.5,1), p)

# printing log(A) and log(U)
print(((ratiotop + phitop)-(ratiobot + phibot)))
print(log(U))
```

D Assessing distribution fits to goals data (Poisson, ZIP, NB)

```
options("scipen"=100)
require(dplyr)
E0_2017 = read.csv("E0_2017.csv")
E0_2016 = read.csv("E0_2016.csv")
E0_2015 = read.csv("E0_2015.csv")
E0_2014 = read.csv("E0_2014.csv")
E0_2013 = read.csv("E0_2013.csv")
E0_2012 = read.csv("E0_2012.csv")
E0_2011 = read.csv("E0_2011.csv")
E0_2010 = read.csv("E0_2010.csv")
E0_2009 = read.csv("E0_2009.csv")
E0_2008 = read.csv("E0_2008.csv")
E0_2007 = read.csv("E0_2007.csv")
E0_2006 = read.csv("E0_2006.csv")

E0_2016 = E0_2016[rev(rownames(E0_2016)),][2:6]
E0_2015 = E0_2015[rev(rownames(E0_2015)),][2:6]
E0_2014 = E0_2014[rev(rownames(E0_2014)),][2:6]
E0_2013 = E0_2013[rev(rownames(E0_2013)),][2:6]
E0_2012 = E0_2012[rev(rownames(E0_2012)),][2:6]
E0_2011 = E0_2011[rev(rownames(E0_2011)),][2:6]
E0_2010 = E0_2010[rev(rownames(E0_2010)),][2:6]
E0_2009 = E0_2009[rev(rownames(E0_2009)),][2:6]
E0_2008 = E0_2008[rev(rownames(E0_2008)),][2:6]
E0_2007 = E0_2007[rev(rownames(E0_2007)),][2:6]
E0_2006 = E0_2006[rev(rownames(E0_2006)),][2:6]
pl.data = rbind(E0_2016, E0_2015, E0_2014, E0_2013, E0_2012, E0_2011, E0
    _2010, E0_2009, E0_2008, E0_2007)

H=0; D=0; A=0
for (game in 1:nrow(pl.data)){
  if (pl.data[game,]$FTHG > pl.data[game,]$FTAG) {
    H = H + 1
  } else if (pl.data[game,]$FTHG == pl.data[game,]$FTAG) {
    D = D + 1
  } else if (pl.data[game,]$FTHG < pl.data[game,]$FTAG) {
    A = A + 1
  }
}
x = c(H,D,A); labels = c('Home Win', 'Draw', 'Away Win')
percent = round(100*x/sum(x), 2)
pie(x, labels = percent, main='Aggregate Outcome Percentages', col = c('
    dodgerblue', 'darkolivegreen3', 'firebrick'))
legend('topright', labels, cex = 0.8, fill = c('dodgerblue', '
    darkolivegreen3', 'firebrick'))

assess_fit = function(expected, observed, data) {
  expected = expected/sum(expected) #making the probabilities sum to 1
  variance = c()
  for (i in 1:length(observed)) {
    variance[i] = ((observed[i]/nrow(data))-(observed[i]/nrow(data))^(2)
    )/(nrow(data)-1)
  }
  st.error = sqrt(variance)

  between = c()
  for (i in 1:length(expected)) {
    between[i] = (expected[i]<(observed[i]/nrow(data))+3*st.error[i]) &&
    (expected[i]>(observed[i]/nrow(data))-3*st.error[i])
  }
  m=cbind(expected, observed=observed/nrow(data), expected.freq=expected
    *nrow(data), observed.freq=observed, 'obs+3se'=(observed/nrow(data))
    +3*st.error, 'obs-3se'=(observed/nrow(data))-3*st.error, variance, st
    .error, between)
  print(m)
  ### Chi-squared test
  chisq.test(x=observed, p=expected)
}

### Poisson?

### 10 seasons worth of data

### Home Goals
t = table(pl.data$FTHG)
t = t/nrow(pl.data) #dividing to get goals per game
bar = t(cbind(as.matrix(t), dpois(0:max(pl.data$FTHG), lambda=mean(pl.
    data$FTHG))))
barplot(bar, ylim=c(0, 0.35), col=c('red', 'navy'), beside=T, legend=
    rownames(bar), legend.text=c('Observed', 'Poisson'), main='Home Goals
     per game (2007/08 - 2016/17)', xlab='Goals', ylab='Proportion of
    Games', border='black')

### Chi-Squared test and finding if they lie in 3sigma either side
assess_fit(expected = dpois(0:max(pl.data$FTHG), lambda=mean(pl.data$
    FTHG)), observed = table(pl.data$FTHG), data = pl.data)

### Away Goals
t = table(pl.data$FTAG)
t = t/nrow(pl.data) #dividing to get goals per game
bar = t(cbind(as.matrix(t), dpois(0:max(pl.data$FTAG), lambda=mean(pl.
    data$FTAG))))
barplot(bar, ylim=c(0, 0.45), col=c('red', 'navy'), beside=T, legend=
    rownames(bar), legend.text=c('Observed', 'Poisson'), main='Away Goals
     per game (2007/08 - 2016/17)', xlab='Goals', ylab='Proportion of
    Games', border='black')

# Assessing fit
assess_fit(expected = dpois(0:max(pl.data$FTAG), lambda=mean(pl.data$
    FTAG)), observed = table(pl.data$FTAG), data = pl.data)


### 2016/17 Season

### Home Goals

# Creating Histogram
t = table(E0_2016$FTHG)
t = t/nrow(E0_2016) #dividing to get goals per game
bar = t(cbind(as.matrix(t), dpois(0:max(E0_2016$FTHG), lambda=mean(E0_
    2016$FTHG))))
barplot(bar, ylim=c(0, 0.35), col=c('red', 'navy'), beside=T, legend=
    rownames(bar), legend.text=c('Observed', 'Poisson'), main='Home Goals
     per game 2016/17', xlab='Goals', ylab='Proportion of Games', border=
    'black')

### Chi-Squared test and finding if they lie in 3sigma either side
assess_fit(expected = dpois(0:max(E0_2016$FTHG), lambda=mean(E0_2016$
    FTHG)), observed = table(E0_2016$FTHG), data = E0_2016)

### Away Goals

# Creating Histogram
t = table(E0_2016$FTAG)
t = t/nrow(E0_2016) #dividing to get goals per game
bar = t(cbind(as.matrix(t), dpois(0:max(E0_2016$FTAG), lambda=mean(E0_
    2016$FTAG))))
barplot(bar, ylim=c(0, 0.35), col=c('red', 'navy'), beside=T, legend=
    rownames(bar), legend.text=c('Observed', 'Poisson'), main='Away Goals
     per game 2016/17', xlab='Goals', ylab='Proportion of Games', border=
    'black')

# Assessing fit
```

```r
assess_fit(expected = dpois(0:max(E0_2016$FTAG), lambda=mean(E0_2016$
    FTAG)), observed = table(E0_2016$FTAG), data = E0_2016)


### Zero-Inflated Poisson?

require(ZIM) #loading package to allow use of zero inflated models
require(Rfast)

### 2016/17 Season

### Home goals
parameters = zip.mle(E0_2016$FTHG)

# Creating histogram
t = table(E0_2016$FTHG)
t = t/nrow(E0_2016) #dividing to get goals per game
bar = t(cbind(as.matrix(t), dzip(0:max(E0_2016$FTHG), lambda=parameters$
    param[1], omega=parameters$param[2])))
barplot(bar, ylim=c(0, 0.35), col=c('red', 'navy'), beside=T, legend=
    rownames(bar), legend.text=c('Observed', 'ZIP'), main='Home Goals per
     game 2016/17', xlab='Goals', ylab='Proportion of Games', border='
    black')

# Assessing fit
assess_fit(expected = dzip(0:max(E0_2016$FTHG), lambda=parameters$param
    [1], omega=parameters$param[2]), observed = table(E0_2016$FTHG), data
     = E0_2016)

### Away goals
parameters = zip.mle(E0_2016$FTAG)

# Creating histogram
t = table(E0_2016$FTAG)
t = t/nrow(E0_2016) #dividing to get goals per game
bar = t(cbind(as.matrix(t), dzip(0:max(E0_2016$FTAG), lambda=parameters$
    param[1], omega=parameters$param[2])))
barplot(bar, ylim=c(0, 0.35), col=c('red', 'navy'), beside=T, legend=
    rownames(bar), legend.text=c('Observed', 'ZIP'), main='Away Goals per
     game 2016/17', xlab='Goals', ylab='Proportion of Games', border='
    black')

# Assessing fit
assess_fit(expected = dzip(0:max(E0_2016$FTAG), lambda=parameters$param
    [1], omega=parameters$param[2]), observed = table(E0_2016$FTAG), data
     = E0_2016)


```

```r
### Negative Binomial?

require(fitdistrplus)

### 2016/17 Season

### Home goals
parameters = fitdist(E0_2016$FTHG, distr='nbinom', method='mle')

# Creating histogram
t = table(E0_2016$FTHG)
t = t/nrow(E0_2016) #dividing to get goals per game
bar = t(cbind(as.matrix(t), dnbinom(0:max(E0_2016$FTHG), size=parameters
    $estimate[1], mu=parameters$estimate[2])))
barplot(bar, ylim=c(0, 0.35), col=c('red', 'navy'), beside=T, legend=
    rownames(bar), legend.text=c('Observed', 'Negative Binomial'), main='
    Home Goals per game 2016/17', xlab='Goals', ylab='Proportion of Games
    ', border='black')

# Assessing fit
assess_fit(expected = dnbinom(0:max(E0_2016$FTHG), size=parameters$
    estimate[1], mu=parameters$estimate[2]), observed = table(E0_2016$
    FTHG), data = E0_2016)

### Away goals
parameters = fitdist(E0_2016$FTAG, distr='nbinom', method='mle')

# Creating histogram
t = table(E0_2016$FTAG)
t = t/nrow(E0_2016) #dividing to get goals per game
bar = t(cbind(as.matrix(t), dnbinom(0:max(E0_2016$FTAG), size=parameters
    $estimate[1], mu=parameters$estimate[2])))
barplot(bar, ylim=c(0, 0.35), col=c('red', 'navy'), beside=T, legend=
    rownames(bar), legend.text=c('Observed', 'Negative Binomial'), main='
    Away Goals per game 2016/17', xlab='Goals', ylab='Proportion of Games
    ', border='black')

# Assessing fit
assess_fit(expected = dnbinom(0:max(E0_2016$FTAG), size=parameters$
    estimate[1], mu=parameters$estimate[2]), observed = table(E0_2016$
    FTAG), data = E0_2016)
```

E Assessing distribution fits to goals data (Geometric-Poisson)

E.1 Stan Code

```
data {
  int ngames;
  int<lower=0> y[ngames];
}
parameters {
  real<lower=0, upper=1> omega;
  real<lower=0, upper=1> prob;
  real<lower=0> lambda;
}
model {
  omega ~ uniform(0,1);
  prob ~ uniform(0,1);
  lambda ~ gamma(3.804279, 2.52015);

  for (i in 1:ngames){
    target += log_sum_exp(log(omega*prob*(1-prob)^(y[i])), log(1-omega)
     + poisson_lpmf(y[i] | lambda));
  }
}
```

E.2 R Code

```
# Using Stan in R to do MCMC and obtain posterior estimates for
    parameters lambda, p, omega
library(rstan)
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())

### Defining a function to work out probabilities given by the Geom-Poi
    distribution
dgeom_pois = function(x, omega, p, lam) {
  probabilities = sapply(x, function(x){omega*p*dgeom(x, prob=p) + (1-
    omega)*dpois(x, lambda=lam)})
  return(probabilities)
}

assess_fit = function(expected, observed, data) {
  expected = expected/sum(expected) #making the probabilities sum to 1
  variance = c()
  for (i in 1:length(observed)) {
    variance[i] = ((observed[i]/nrow(data))-(observed[i]/nrow(data))^(2)
   )/(nrow(data)-1)
  }
  st.error = sqrt(variance)

  between = c()
  for (i in 1:length(expected)) {
    between[i] = (expected[i]<(observed[i]/nrow(data))+3*st.error[i]) &&
    (expected[i]>(observed[i]/nrow(data))-3*st.error[i])
  }
  m=cbind(expected, observed=observed/nrow(data), expected.freq=expected
   *nrow(data), observed.freq=observed, 'obs+3se'=(observed/nrow(data))
   +3*st.error, 'obs-3se'=(observed/nrow(data))-3*st.error, variance, st
   .error, between)
  print(m)
  ### Chi-squared test
  chisq.test(x=observed, p=expected)
}

### Plot of the prior for lambda = Gamma(3.804279, 2.52015)
plot(seq(0,4,0.1), dgamma(seq(0,4,0.1), 3.804279, 2.52015), type='l',
    xlab=expression(lambda), ylab=expression(paste("Prior density for ",
    lambda)), main=expression(paste("Pior distribution for ", lambda)))

### 2016/17 season

### Home goals
data = list(y = E0_2016$FTHG, ngames = nrow(E0_2016))
fitting_model = stan_model(file = 'geom_pois.stan')
fit = sampling(object = fitting_model, data = data, iter = 100000,
    chains = 4)

# Looking at the fit and finding the posterior means for the parameters
print(fit)
plot(fit)

# Creating Histogram
t = table(E0_2016$FTHG)
t = t/nrow(E0_2016) #dividing to get goals per game
bar = t(cbind(as.matrix(t), dgeom_pois(0:max(E0_2016$FTHG), omega=0.08,
    p=0.63, lam=1.67)))
barplot(bar, ylim=c(0, 0.35), col=c('red', 'navy'), beside=T, legend=
    rownames(bar), legend.text=c('Observed', 'Geometric-Poisson'), main='
    Home Goals per game 2016/17', xlab='Goals', ylab='Proportion of Games
    ', border='black')

```

```r
# Getting the distributions given by the formula
y = dgeom_pois(0:max(E0_2016$FTHG), omega=0.08, p=0.63, lam=1.67)

# Assessing the fit
assess_fit(expected = y, observed = table(E0_2016$FTHG), data = E0_2016)

### Away goals
data = list(y = E0_2016$FTAG, ngames = nrow(E0_2016))
fitting_model = stan_model(file = 'geom_pois.stan')
fit = sampling(object = fitting_model, data = data, iter = 100000,
    chains = 4, control = list(max_treedepth = 15))

# Looking at the fit and finding the posterior means for the parameters
print(fit)
plot(fit)

# Creating Histogram
t = table(E0_2016$FTAG)
t = t/nrow(E0_2016) #dividing to get goals per game
bar = t(cbind(as.matrix(t), dgeom_pois(0:max(E0_2016$FTAG), omega=0.28,
    p=0.61, lam=1.38)))
barplot(bar, ylim=c(0, 0.35), col=c('red', 'navy'), beside=T, legend=
    rownames(bar), legend.text=c('Observed', 'Geometric-Poisson'), main='
    Away Goals per game 2016/17', xlab='Goals', ylab='Proportion of Games
    ', border='black')

# Getting the distributions given by the formula
y=dgeom_pois(0:max(E0_2016$FTAG), omega=0.28, p=0.61, lam=1.38)

# Assessing the fit
assess_fit(expected = y, observed = table(E0_2016$FTAG), data = E0_2016)
```

F Derivation of results for Zero-Inflated Poisson distribution

The probability mass function for a Zero-Inflated Poisson random variable, Y, is given by

$$\Pr(Y = y) = \begin{cases} \omega + (1-\omega)\exp(-\lambda) & \text{if } y = 0, \\ (1-\omega)\frac{\lambda^y \exp(-\lambda)}{y!} & \text{if } y > 0. \end{cases}$$

The Mean

First calculating the mean, by the expectation of a variable Y, we have

$$\begin{aligned}
\mathrm{E}[Y] &= \sum_{y=1}^{\infty} y Pr(Y = y) \\
&= \sum_{y=1}^{\infty} y \frac{(1-\omega)\lambda^y \exp(-\lambda)}{y!} \\
&= (1-\omega) \sum_{y=1}^{\infty} y \frac{\lambda^y \exp(-\lambda)}{y!} \\
&= (1-\omega)\lambda
\end{aligned}$$

as $\frac{\lambda^y \exp(-\lambda)}{y!}$ is the probability function for the Poisson(λ) distribution and the mean of a Poisson(λ) distribution is λ.

The Variance

To calculate the variance, we use $\mathrm{Var}[Y] = \mathrm{E}[Y^2] - \mathrm{E}[Y]^2$. So first finding $\mathrm{E}[Y^2]$.

$$\begin{aligned}
\mathrm{E}[Y]^2 &= \sum_{y=1}^{\infty} y^2 Pr(Y = y) \\
&= \sum_{y=1}^{\infty} y^2 \frac{(1-\omega)\lambda^y \exp(-\lambda)}{y!} \\
&= (1-\omega) \sum_{y=1}^{\infty} y^2 \frac{\lambda^y \exp(-\lambda)}{y!} \\
&= (1-\omega)\lambda(\lambda+1)
\end{aligned}$$

as $\frac{\lambda^y \exp(-\lambda)}{y!}$ is the probability function for the Poisson(λ) distribution and we know that $\mathrm{E}[Y]^2 = \lambda(\lambda+1)$ if $Y \mid \lambda \sim$ Poisson(λ).

Hence, by rearrangement, the variance is

$$\begin{aligned}
\mathrm{Var}[Y] &= \mathrm{E}[Y^2] - \mathrm{E}[Y]^2 \\
&= \lambda(1-\omega)(1+\omega\lambda)
\end{aligned}$$

Hence for the Zero-Inflated Poisson model, the mean and variance are given by

$$\mathrm{E}[Y] = (1-\omega)\lambda \text{ and } \mathrm{Var}[Y] = \lambda(1-\omega)(1+\omega\lambda)$$

Maximum Likelihood Estimates

Now consider a sample $\underline{y} = (y_1, ..., y_n)$ of independent and identically distributed random variables $Y_i \mid \lambda, \omega \sim \text{ZIP}(\lambda, \omega)$. Recall the probability distribution function for a random variable $Y \mid \lambda, \omega \sim \text{ZIP}(\lambda, \omega)$ is

$$\Pr(Y = y) = \begin{cases} \omega + (1-\omega)\exp(-\lambda) & \text{if } y = 0, \\ (1-\omega)\frac{\lambda^y \exp(-\lambda)}{y_i!} & \text{if } y > 0. \end{cases}$$

Let r denote the number of zero observations in the sample $\underline{y}$. Then the likelihood function $l(\lambda, \omega; y)$ of the sample $\underline{y}$ is given by

$$\begin{aligned} l(\lambda, \omega; y) &= \prod_{i=1}^{n} Pr(Y = y_i) \\ &= \prod_{y_i=0} (\omega + (1-\omega)\exp(-\lambda)) \prod_{y_i>0} (1-\omega)\frac{\lambda^{y_i}\exp(-\lambda)}{y_i!} \\ &= (\omega + (1-\omega)\exp(-\lambda))^r (1-\omega)^{n-r} \frac{\lambda^{\sum_{i=0}^{n} y_i} \exp(-\lambda(n-r))}{y_1! \, ... y_n!} \end{aligned}$$

Note that the summation and the product of y_i's in this formula only have $n-r$ elements despite having the limit going up to n, as r of them are when $y_i = 0$.

If we take logarithms, then the log-likelihood, $L(\lambda, \omega; y) = \log\, l(\lambda, \omega; y)$, is given by

$$L(\lambda, \omega; y) = r\log(\omega + (1-\omega)\exp(-\lambda)) + (n-r)\log(1-\omega) + \sum_{i=0}^{n} y_i \log(\lambda) - \lambda(n-r) - \log(y_1! \, ... y_n!\,)$$

First partial differentiating this with respect to ω, we get

$$\frac{\partial L}{\partial \omega} = \frac{r(1-\exp(-\lambda))}{\omega + (1-\omega)\exp(-\lambda)} - \frac{n-r}{1-\omega}$$

Now setting this to zero, $\frac{\partial L}{\partial \omega} = 0$, to identify the two maxima/minima, gives

$$\frac{r(1-e^{-\lambda})}{\omega + (1-\omega)e^{-\lambda}} - \frac{n-r}{1-\omega} = 0$$

$$
\begin{aligned}
&\implies r(1-\omega)(1-e^{-\lambda}) + (r-n)(\omega + (1-\omega)e^{-\lambda}) = 0 \\
&\implies r(1-e^{-\lambda} - \omega + \omega e^{-\lambda}) + r\omega + r(1-\omega)e^{-\lambda} - n\omega - n(1-\omega)e^{-\lambda} = 0 \\
&\implies r - re^{-\lambda} - r\omega + r\omega e^{-\lambda} + r\omega + re^{-\lambda} - r\omega e^{-\lambda} - n\omega - ne^{-\lambda} + n\omega e^{-\lambda} = 0 \\
&\implies r - n\omega - ne^{-\lambda} + n\omega e^{-\lambda} = 0 \\
&\implies r - e^{-\lambda} = \omega(n - ne^{-\lambda}) \\
&\implies \omega = \frac{r - e^{-\lambda}}{n(1-e^{-\lambda})}
\end{aligned}
$$

Hence the maximum likelihood estimate for ω is

$$
\hat{\omega} = \frac{r - e^{-\lambda}}{n(1-e^{-\lambda})}.
$$

Now to find the maximum likelihood estimate for λ, we need to differentiate $L(\lambda, \omega; y)$ with respect to λ.

$$
\begin{aligned}
\frac{\partial L}{\partial \lambda} &= \frac{r(-(1-\omega)\exp(-\lambda))}{\omega + (1-\omega)\exp(-\lambda)} + \sum_{i=0}^{n} \frac{y_i}{\lambda} - (n-r) \\
&= \frac{r\exp(-\lambda)(\omega - 1)}{\omega + (1-\omega)\exp(-\lambda)} + \sum_{i=0}^{n} \frac{y_i}{\lambda} - (n-r)
\end{aligned}
$$

Now setting this to zero, $\frac{\partial L}{\partial \lambda} = 0$, and substituting in $\omega = \frac{r-e^{-\lambda}}{n(1-e^{-\lambda})}$, then we get

$$
\begin{aligned}
&\frac{re^{-\lambda}(r-n)}{r(1-e^{-\lambda})} + \sum_{i=0}^{n} \frac{y_i}{\lambda} - (n-r) = 0 \\
&\implies re^{-\lambda}(r-n) + r(1-e^{-\lambda}) \sum_{i=0}^{n} \frac{y_i}{\lambda} + (r-n)(r(1-e^{-\lambda})) = 0 \\
&\implies r^2 e^{-\lambda} - re^{-\lambda} n + r(1-e^{-\lambda}) \sum_{i=0}^{n} \frac{y_i}{\lambda} + r^2 - r^2 e^{-\lambda} - nr + re^{-\lambda} n = 0 \\
&\implies r(1-e^{-\lambda}) \sum_{i=0}^{n} \frac{y_i}{\lambda} + r^2 - nr = 0 \\
&\implies (1-e^{-\lambda}) \sum_{i=0}^{n} \frac{y_i}{\lambda} + r - n = 0 \\
&\implies (1-e^{-\lambda}) \sum_{i=0}^{n} \frac{y_i}{\lambda} + r - n = 0 \\
&\implies (1-e^{-\lambda}) \sum_{i=0}^{n} y_i = \lambda(n-r)
\end{aligned}
$$

which requires numerical methods to solve.

G Derivation of results for Geometric-Poisson distribution

The probability mass function of a Geometric-Poisson random variable, Y, is given by

$$\Pr(Y = y) = \omega p(1-p)^y + (1-\omega)\frac{\lambda^y e^{-\lambda}}{y!}$$

The Mean

By the expectation of a variable Y, we have

$$\begin{aligned}
\mathrm{E}[Y] &= \sum_{y=1}^{\infty} y Pr(Y = y) \\
&= \sum_{y=1}^{\infty} \Big[y\omega p(1-p)^y + y(1-\omega)\frac{\lambda^y \exp(-\lambda)}{y!} \Big] \\
&= \omega p(1-p) \sum_{y=1}^{\infty} y(1-p)^{y-1} + (1-\omega) \sum_{y=1}^{\infty} \frac{y\lambda^y \exp(-\lambda)}{y!} \\
&= \omega p(1-p) \Big(\frac{1}{1-(1-p)} \Big)^2 + \lambda(1-\omega) \\
&= \frac{\omega(1-p)}{p} + \lambda(1-\omega)
\end{aligned}$$

as $\frac{\lambda^y \exp(-\lambda)}{y!}$ is the probability function for the Poisson(λ) distribution and the mean of a Poisson(λ) distribution is λ and by using Geometric progression formulae.

The Variance

To calculate the variance, we first calculate $\mathrm{E}[Y^2]$,

$$\begin{aligned}
\mathrm{E}[Y^2] &= \sum_{y=1}^{\infty} y^2 Pr(Y = y) \\
&= \sum_{y=1}^{\infty} \Big[y^2\omega p(1-p)^y + y^2(1-\omega)\frac{\lambda^y \exp(-\lambda)}{y!} \Big] \\
&= \omega p(1-p) \sum_{y=1}^{\infty} y^2(1-p)^{y-1} + (1-\omega) \sum_{y=1}^{\infty} \frac{y^2\lambda^y \exp(-\lambda)}{y!}
\end{aligned}$$

Consider the first term, then this can be simplified by using the geometric progression formulae.

$$\begin{aligned}\omega p(1-p)\sum_{y=1}^{\infty} y^2(1-p)^{y-1} &= \omega p(1-p)\sum_{y=1}^{\infty} y(y+1)(1-p)^{y-1} - \omega p(1-p)\sum_{y=1}^{\infty} y(1-p)^{y-1} \\ &= \omega p(1-p)\frac{2}{(1-(1-p))^3} - \frac{\omega(1-p)}{p} \\ &= \omega(1-p)\Big(\frac{2}{p^2} - \frac{1}{p}\Big)\end{aligned}$$

Then, we have

$$\begin{aligned}\mathrm{E}[Y^2] &= \omega p(1-p)\sum_{y=1}^{\infty} y^2(1-p)^{y-1} + (1-\omega)\sum_{y=1}^{\infty}\frac{y^2\lambda^y \exp(-\lambda)}{y!} \\ &= \omega(1-p)\Big(\frac{2}{p^2} - \frac{1}{p}\Big) + (1-\omega)\lambda(\lambda+1)\end{aligned}$$

Hence by rearrangement and using the formula, $\mathrm{Var}[Y] = \mathrm{E}[Y^2] - \mathrm{E}[Y]^2$, the variance for the Poisson-Geometric mixture distribution is

$$\mathrm{Var}[Y] = \omega(1-p)\Big(\frac{2}{p^2} - \frac{1}{p}\Big) - \frac{\omega^2(1-p^2)^2}{p^2} + \lambda(1-\omega)(1+\omega\lambda) - \frac{2\omega\lambda(1-p)(1-\omega)}{p}$$

Hence for the Geometric-Poisson mixture distribution, the mean and variance are

$$\begin{aligned}\mathrm{E}[Y] &= \frac{\omega(1-p)}{p} + \lambda(1-\omega) \\ \mathrm{Var}[Y] &= \omega(1-p)\Big(\frac{2}{p^2} - \frac{1}{p}\Big) - \frac{\omega^2(1-p^2)^2}{p^2} + \lambda(1-\omega)(1+\omega\lambda) - \frac{2\omega\lambda(1-p)(1-\omega)}{p}\end{aligned}$$

H Implementing Baio & Blangiardo's model (2010)

H.1 Stan Code

```
data {
  int nteams;
  int ngames;
  int home_team[ngames];
  int away_team[ngames];
  int<lower=0> home_goals[ngames];
  int<lower=0> away_goals[ngames];
}
```

```
parameters {
  real home;
  real mu_att;
  real mu_def;
  real tau_att;
  real tau_def;

  // parameters are made to sum to zero in the transformed parameters
  vector[nteams-1] att_free;
  vector[nteams-1] def_free;
}
transformed parameters {
  vector[nteams] att;
  vector[nteams] def;
  vector[ngames] log_theta_home;
  vector[ngames] log_theta_away;

  // need to make sum(att)=sum(def)=0
  for (k in 1:(nteams-1)) {
    att[k] = att_free[k];
    def[k] = def_free[k];
  }
  att[nteams] = -sum(att_free);
  def[nteams] = -sum(def_free);

  log_theta_home = home + att[home_team] + def[away_team];
  log_theta_away = att[away_team] + def[home_team];
}
model {
  home ~ normal(0, 10000);
  mu_att ~ normal(0, 10000);
  mu_def ~ normal(0, 10000);
  tau_att ~ gamma(0.1, 0.1);
  tau_def ~ gamma(0.1, 0.1);

  att_free ~ normal(mu_att, 1/tau_att);
  def_free ~ normal(mu_def, 1/tau_def);

  home_goals ~ poisson_log(log_theta_home);
  away_goals ~ poisson_log(log_theta_away);
}
```

H.2 R Code

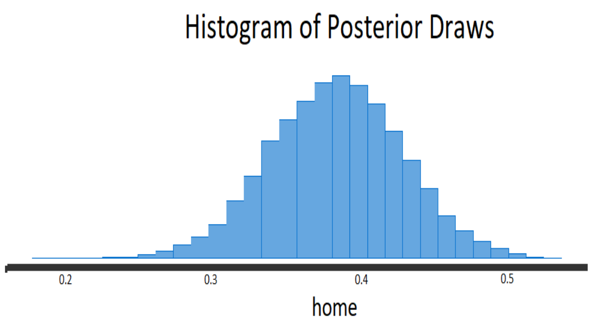
R code to implement the model, obtain parameter plots, examples of predicting football match and to create plot in Figure 14.

```r
library(rstan)
library(shinystan)
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())

# Loading in the data into R

E0_2016 = read.csv('E0_2016.csv')
E0_2016 = E0_2016[rev(rownames(E0_2016)),][3:6]
E0_2016$HomeIndex = as.numeric(factor(E0_2016$HomeTeam))
E0_2016$AwayIndex = as.numeric(factor(E0_2016$AwayTeam))

# Preparing the data in R to use in Stan

teams = as.character(sort(unique(E0_2016$HomeTeam)))
HomeTeam = E0_2016$HomeIndex
AwayTeam = E0_2016$AwayIndex
HomeGoals = E0_2016$FTHG
AwayGoals = E0_2016$FTAG
nTeams = length(teams)
nGames = nrow(E0_2016)

data = list(nteams = nTeams, ngames = nGames, home_team = HomeTeam, away
    _team = AwayTeam, home_goals = HomeGoals, away_goals = AwayGoals)

# Getting fit using the stan_model and sampling functions

fitting_model = stan_model(file = 'sum_to_zero.stan')
fit = sampling(object = fitting_model, data = data, iter = 100000,
    chains = 4, control = list(max_treedepth = 15))

# Looking at the fit
print(fit)

# Making the parameter plot figures
# Rather than using labels with indexes, we rename them to the teams
    that they represent
new_labels=rev(c("ARS", "BOU", "BUR", "CHE", "CRY", "EVE", "HUL", "LEI",
     "LIV", "MCI", "MUN", "MBO", "SOU", "STO", "SUN", "SWA", "TOT", "WAT"
    , "WBA", "WHU"))
# Getting the figure and naming it att_figure (attack parameters)
```

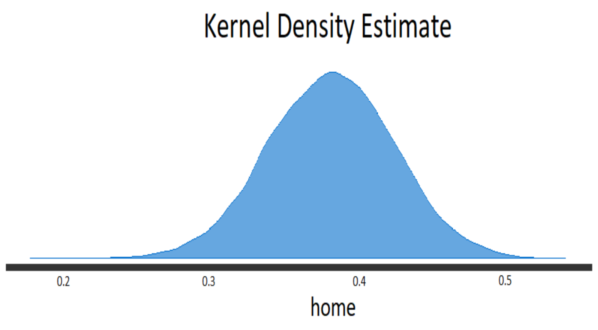
```r
att_figure = plot(fit, pars = c("att[1]", "att[2]", "att[3]", "att[4]",
    "att[5]", "att[6]", "att[7]", "att[8]", "att[9]", "att[10]", "att[11]
    ", "att[12]", "att[13]", "att[14]", "att[15]", "att[16]", "att[17]",
    "att[18]", "att[19]", "att[20]"))
att_figure + scale_y_continuous(labels=new_labels, breaks=1:20)

# Getting the figure and naming it def_figure (defence parameters)
def_figure = plot(fit, pars = c("def[1]", "def[2]", "def[3]", "def[4]",
    "def[5]", "def[6]", "def[7]", "def[8]", "def[9]", "def[10]", "def[11]
    ", "def[12]", "def[13]", "def[14]", "def[15]", "def[16]", "def[17]",
    "def[18]", "def[19]", "def[20]"))
def_figure + scale_y_continuous(labels=new_labels, breaks=1:20)

# Getting the shinystan page to make further analysis
fit_shinystan = as.shinystan(fit)
launch_shinystan(fit_shinystan)

### Example to predict a football match between Manchester United and
    Crystal Palace

list_of_draws = extract(fit)
home = list_of_draws$home
MUN_att = list_of_draws$att[,11]
MUN_def = list_of_draws$def[,11]
CRY_att = list_of_draws$att[,5]
CRY_def = list_of_draws$def[,5]

log_theta1 = home + MUN_att + CRY_def
log_theta2 = CRY_att + MUN_def

y1 = NULL; y2 = NULL
for (i in 1:length(log_theta1)) {
  y1[i] = rpois(1, exp(log_theta1[i]))
  y2[i] = rpois(1, exp(log_theta2[i]))
}

sum(round(y1,1) > round(y2,1))/length(log_theta1)
sum(round(y1,1) == round(y2,1))/length(log_theta1)
sum(round(y1,1) < round(y2,1))/length(log_theta1)

### Example of how to perform a Bayesian hypothesis test

TOT_def = list_of_draws$def[,17]
MUN_def = list_of_draws$def[,11]
sum(MUN_def < TOT_def)/length(TOT_def)

### Figure 14

```

```
att_sum = summary(fit, pars = c("att[1]", "att[2]", "att[3]", "att[4]",
    "att[5]", "att[6]", "att[7]", "att[8]", "att[9]", "att[10]", "att[11]
    ", "att[12]", "att[13]", "att[14]", "att[15]", "att[16]", "att[17]",
    "att[18]", "att[19]", "att[20]"))
att_mean = att_sum$summary[,'mean']

def_sum = summary(fit, pars = c("def[1]", "def[2]", "def[3]", "def[4]",
    "def[5]", "def[6]", "def[7]", "def[8]", "def[9]", "def[10]", "def[11]
    ", "def[12]", "def[13]", "def[14]", "def[15]", "def[16]", "def[17]",
    "def[18]", "def[19]", "def[20]"))
def_mean = def_sum$summary[,'mean']

new_labels=c("ARS", "BOU", "BUR", "CHE", "CRY", "EVE", "HUL", "LEI", "
    LIV", "MCI", "MUN", "MBO", "SOU", "STO", "SUN", "SWA", "TOT", "WAT",
    "WBA", "WHU")

cols = c('red','firebrick4','brown','darkblue','blue2','darkblue','
    orange','dodgerblue2','red','deepskyblue','firebrick','firebrick1','
    firebrick1','red','red','gray0','blue4', 'gold1', 'darkblue' ,'brown4
    ')

data = data.frame(att_mean, def_mean, new_labels)

plot(att_mean, def_mean, col= cols, xlab = 'Attack', ylab = 'Defence',
    main='Team Effects', xlim=c(-0.6,0.6), ylim=c(-0.55,0.5))
abline(v=0); abline(h=0)
text(att_mean, def_mean, labels = data$new_labels, cex=0.7, pos=3)
text(0.55, -0.535, "Good Attack, Good Defence", cex=0.8)
text(-0.55, 0.475, "Bad Attack, Bad Defence", cex=0.8)
```

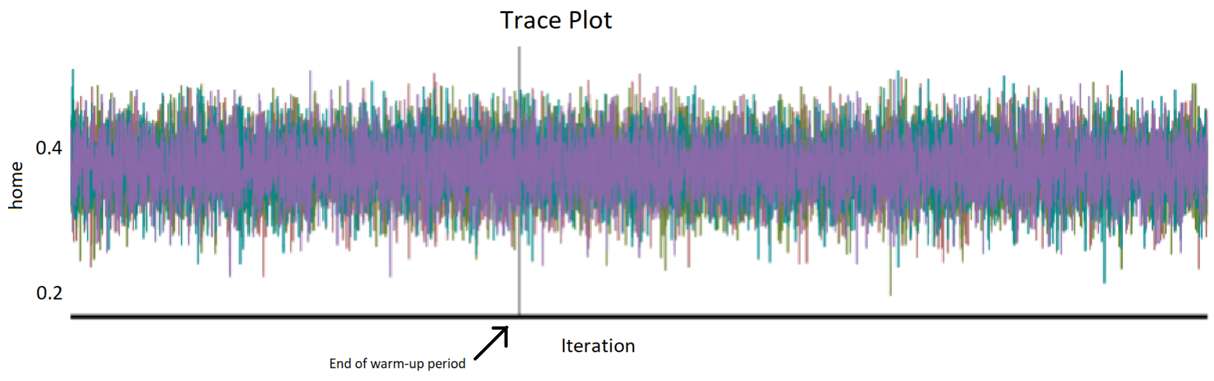
H.3 Extra Figures for Section 5



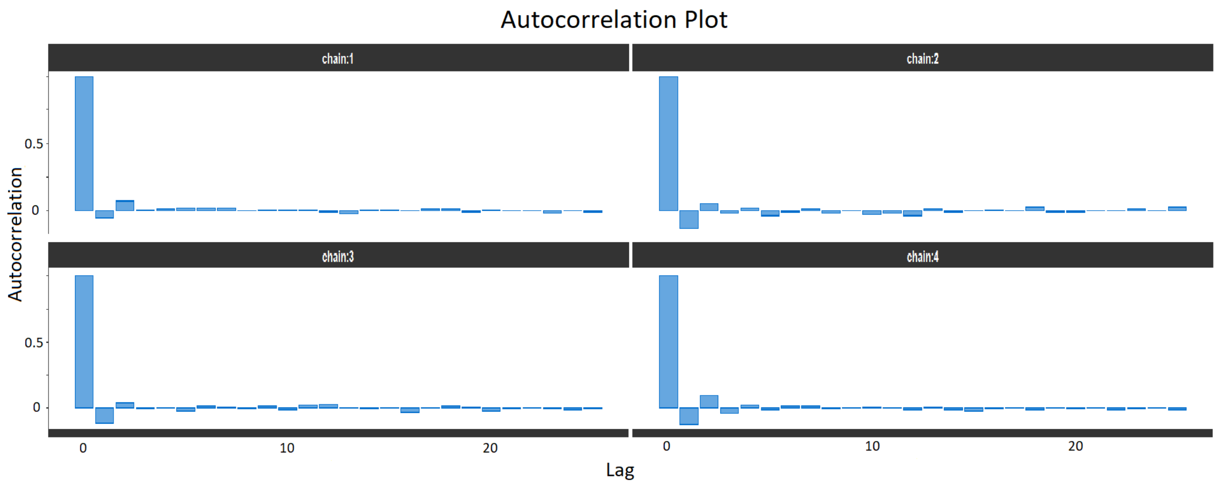
(a) Histogram for draws of *home*



(b) Kernel Density Estimate for *home*



(c) Trace plot for *home*



(d) Autocorrelation plot for *home*

Figure 22: MCMC plots for *home* parameter

H.4 Posterior Summary Table

	Attack					Defence				
Team	mean	sd	2.5%	50%	97.5%	mean	sd	2.5%	50%	97.5%
ARS	0.375	0.112	0.152	0.377	0.592	-0.116	0.130	-0.375	-0.113	0.130
BOU	0.078	0.126	-0.173	0.080	0.320	0.219	0.115	-0.006	0.219	0.440
BUR	-0.228	0.141	-0.513	-0.224	0.039	0.039	0.121	-0.204	0.040	0.270
CHE	0.462	0.109	0.247	0.462	0.673	-0.320	0.144	-0.612	-0.315	-0.048
CRY	-0.010	0.131	-0.273	-0.007	0.239	0.161	0.119	-0.074	0.162	0.393
EVE	0.170	0.120	-0.069	0.171	0.401	-0.127	0.129	-0.388	-0.124	0.114
HUL	-0.250	0.143	-0.540	-0.245	0.022	0.368	0.110	0.150	0.369	0.580
LEI	-0.045	0.131	-0.307	-0.043	0.204	0.159	0.119	-0.077	0.161	0.390
LIV	0.386	0.112	0.167	0.389	0.599	-0.152	0.131	-0.414	-0.151	0.096
MCI	0.409	0.109	0.190	0.411	0.620	-0.207	0.136	-0.479	-0.204	0.051
MUN	0.035	0.126	-0.218	0.037	0.280	-0.427	0.152	-0.745	-0.421	-0.19
MBO	-0.511	0.162	-0.842	-0.505	-0.204	0.0002	0.122	-0.240	0.001	0.235
SOU	-0.191	0.138	-0.470	-0.188	0.071	-0.072	0.126	-0.327	-0.071	0.172
STO	-0.185	0.141	-0.470	-0.181	0.081	0.055	0.121	-0.187	0.056	0.290
SUN	-0.448	0.157	-0.770	-0.443	-0.154	0.228	0.112	0.007	0.228	0.443
SWA	-0.096	0.134	-0.367	-0.094	0.159	0.252	0.113	0.030	0.252	0.472
TOT	0.470	0.108	0.252	0.472	0.680	-0.474	0.160	-0.808	-0.467	-0.177
WAT	-0.198	0.142	-0.486	-0.195	0.071	0.222	0.116	-0.004	0.222	0.445
WBA	-0.148	0.136	-0.423	-0.144	0.111	-0.022	0.123	-0.266	-0.021	0.212
WHU	-0.076	0.145	-0.367	-0.0720	0.200	0.215	0.126	-0.041	0.217	0.454
home	0.382	0.040	0.298	0.383	0.465					

Table 21: Summary of the results from the implementation of Baio & Blangiardo's model (2010)

H.5 Simulating from the priors from Baio & Blangiardo's model (2010)

```
### Simulating mu and tau parameters
### Priors given by Baio & Blangiardo
#mu.att = rnorm(10000, mean=0, sd=sqrt(1/0.0001))
#mu.def = rnorm(10000, mean=0, sd=sqrt(1/0.0001))
#tau.att = rgamma(10000, shape=0.1, scale=0.1)
#tau.def = rgamma(10000, shape=0.1, scale=0.1)

### Proposed more suitable priors
mu.att = rnorm(10000, mean=0, sd=1)
mu.def = rnorm(10000, mean=0, sd=1)
tau.att = rgamma(10000, shape=10, scale=10)
tau.def = rgamma(10000, shape=10, scale=10)

# simulating home advantage parameter

# home.adv = rnorm(10000, mean=0, sd=sqrt(1/0.0001)) # Prior given by
    Baio & Blangiardo
home.adv = rnorm(10000, mean=0, sd=1) # Proposed more suitable prior

# simulating the next 4 parameters for home/away team attack and defence
     depending on mu, tau
home.att = c()
home.def = c()
away.att = c()
away.def = c()
for (i in 1:10000){
  home.att[i]=rnorm(1, mean=mu.att[i], sd=sqrt(1/tau.att[i]))
  home.def[i]=rnorm(1, mean=mu.def[i], sd=sqrt(1/tau.def[i]))
  away.att[i]=(-1)*home.att[i]
  away.def[i]=(-1)*home.def[i]
}

# translating these into theta for home and away based on the above
    parameters
# fitting in log linear model to find theta1 (home) and theta2 (away)
theta1 = c()
theta2 = c()
for (i in 1:10000){
  theta1[i] = exp(home.adv[i] + home.att[i] + away.def[i])
  theta2[i] = exp(away.att[i] + home.def[i])
}

# getting the scores by taking a random sample from a Poisson
    distribution
y1=c()
```

```
y2=c()
for (i in 1:10000){
  y1[i]=rpois(1, lambda=theta1[i])
  y2[i]=rpois(1, lambda=theta2[i])
}

scores=data.frame(y1, y2)
```

I Negative Binomial Bayesian hierarchical model

I.1 Stan Code

```
data {
  int nteams;
  int ngames;
  int home_team[ngames];
  int away_team[ngames];
  int<lower=0> home_goals[ngames];
  int<lower=0> away_goals[ngames];
}
parameters {
  real mu_home_att;
  real mu_away_att;
  real mu_home_def;
  real mu_away_def;
  real<lower=0> sigma2_att;
  real<lower=0> sigma2_def;
  real<lower=0> phi_home;
  real<lower=0> phi_away;

  simplex[nteams] home_att_raw;
  simplex[nteams] away_att_raw;
  simplex[nteams] home_def_raw;
  simplex[nteams] away_def_raw;

  real home_att_scale;
  real away_att_scale;
  real home_def_scale;
  real away_def_scale;
}
transformed parameters {
  vector[ngames] log_mu_home;
  vector[ngames] log_mu_away;
  vector[nteams] home_att;
  vector[nteams] away_att;
  vector[nteams] home_def;
  vector[nteams] away_def;

  home_att = home_att_scale * (home_att_raw - 1.0/nteams);
  away_att = away_att_scale * (away_att_raw - 1.0/nteams);
  home_def = home_def_scale * (home_def_raw - 1.0/nteams);
  away_def = away_def_scale * (away_def_raw - 1.0/nteams);

  // getting mu in log form
  log_mu_home = home_att[home_team] + away_def[away_team];
  log_mu_away = away_att[away_team] + home_def[home_team];
}
model {
  mu_home_att ~ normal(0.2, 1);
  mu_away_att ~ normal(0, 1);
  mu_home_def ~ normal(-0.2, 1);
  mu_away_def ~ normal(0, 1);
  sigma2_att ~ gamma(10, 10);
  sigma2_def ~ gamma(10, 10);
  phi_home ~ gamma(2.5, 0.05);
  phi_away ~ gamma(2.5, 0.05);

  home_att_scale ~ normal(0, 10);
  away_att_scale ~ normal(0, 10);
  home_def_scale ~ normal(0, 10);
  away_def_scale ~ normal(0, 10);

  home_att_raw ~ normal(mu_home_att, sigma2_att);
  away_att_raw ~ normal(mu_away_att, sigma2_att);
  home_def_raw ~ normal(mu_home_def, sigma2_def);
  away_def_raw ~ normal(mu_away_def, sigma2_def);

  home_goals ~ neg_binomial_2_log(log_mu_home, phi_home);
  away_goals ~ neg_binomial_2_log(log_mu_away, phi_away);
}
```

I.2 R Code

R code to implement the model, obtain parameter plots, examples of hypothesis testing and to create plots in Figures 18, 19 and 23.

```r
library(dplyr)
library(rstan)
require(shinystan)
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())

# Loading in the data into R

E0_2016 = read.csv('E0_2016.csv')
E0_2016 = E0_2016[rev(rownames(E0_2016)),][3:6]
E0_2016$HomeIndex = as.numeric(factor(E0_2016$HomeTeam))
E0_2016$AwayIndex = as.numeric(factor(E0_2016$AwayTeam))

# Preparing the data in R to use in Stan

teams = as.character(sort(unique(E0_2016$HomeTeam)))
HomeTeam = E0_2016$HomeIndex
AwayTeam = E0_2016$AwayIndex
HomeGoals = E0_2016$FTHG
AwayGoals = E0_2016$FTAG
nTeams = length(teams)
nGames = nrow(E0_2016)

data = list(nteams = nTeams, ngames = nGames, home_team = HomeTeam, away
    _team = AwayTeam, home_goals = HomeGoals, away_goals = AwayGoals)

# Getting the fit using the stan_model and sampling functions

fitting_model = stan_model(file = 'model_nb.stan')
fit = sampling(object = fitting_model, data = data, iter = 100000,
    chains = 4, control = list(max_treedepth = 15))

# Looking at the fit
print(fit)

# Making the parameter plot figures
# Rather than using labels with indexes, we rename them to the teams
    that they represent
new_labels=rev(c("ARS", "BOU", "BUR", "CHE", "CRY", "EVE", "HUL", "LEI",
     "LIV", "MCI", "MUN", "MBO", "SOU", "STO", "SUN", "SWA", "TOT", "WAT"
    , "WBA", "WHU"))

home_att_figure = plot(fit, pars = c("home_att[1]", "home_att[2]", "home
    _att[3]", "home_att[4]", "home_att[5]", "home_att[6]", "home_att[7]",
 "home_att[8]", "home_att[9]", "home_att[10]", "home_att[11]", "home_att[12]", "home_att[13]", "home_att[14]", "home_att[15]", "home_att[16]", "home_att[17]", "home_att[18]", "home_att[19]", "home_att[20]"), xlab='home_attack Effects', main='Home Attack Parameter Plot')
home_att_figure + scale_y_continuous(labels=new_labels, breaks=1:20) + ggtitle('   Home Attack Parameter Plot')

home_def_figure = plot(fit, pars = c("home_def[1]", "home_def[2]", "home_def[3]", "home_def[4]", "home_def[5]", "home_def[6]", "home_def[7]", "home_def[8]", "home_def[9]", "home_def[10]", "home_def[11]", "home_def[12]", "home_def[13]", "home_def[14]", "home_def[15]", "home_def[16]", "home_def[17]", "home_def[18]", "home_def[19]", "home_def[20]"), xlab='home_defack Effects')
home_def_figure + scale_y_continuous(labels=new_labels, breaks=1:20) + xlab('Home-Defence Effects') + ggtitle('   Home Defence Parameter Plot')

away_att_figure = plot(fit, pars = c("away_att[1]", "away_att[2]", "away_att[3]", "away_att[4]", "away_att[5]", "away_att[6]", "away_att[7]", "away_att[8]", "away_att[9]", "away_att[10]", "away_att[11]", "away_att[12]", "away_att[13]", "away_att[14]", "away_att[15]", "away_att[16]", "away_att[17]", "away_att[18]", "away_att[19]", "away_att[20]"))
away_att_figure + scale_y_continuous(labels=new_labels, breaks=1:20) + xlab('Away-Attack Effects') + ggtitle('   Away Attack Parameter Plot')

away_def_figure = plot(fit, pars = c("away_def[1]", "away_def[2]", "away_def[3]", "away_def[4]", "away_def[5]", "away_def[6]", "away_def[7]", "away_def[8]", "away_def[9]", "away_def[10]", "away_def[11]", "away_def[12]", "away_def[13]", "away_def[14]", "away_def[15]", "away_def[16]", "away_def[17]", "away_def[18]", "away_def[19]", "away_def[20]"))
away_def_figure + scale_y_continuous(labels=new_labels, breaks=1:20) + xlab('Away-Defence Effects') + ggtitle(' Away Defence Parameter Plot')

# Getting the shinystan page to make further analysis
fit_shinystan = as.shinystan(fit)
launch_shinystan(fit_shinystan)

### Example of hypothesis testing for Sunderland

list_of_draws = extract(fit)
SUN_home_att = list_of_draws$home_att[,15]
SUN_away_att = list_of_draws$away_att[,15]
SUN_home_def = list_of_draws$home_def[,15]
SUN_away_def = list_of_draws$away_def[,15]
```

```r
sum(SUN_home_att > SUN_away_att)/length(SUN_home_att)
sum(SUN_home_def > SUN_away_def)/length(SUN_home_def)

### Figure 18

home_att_sum = summary(fit, pars = c("home_att[1]", "home_att[2]", "home
    _att[3]", "home_att[4]", "home_att[5]", "home_att[6]", "home_att[7]",
     "home_att[8]", "home_att[9]", "home_att[10]", "home_att[11]", "home_
    att[12]", "home_att[13]", "home_att[14]", "home_att[15]", "home_att
    [16]", "home_att[17]", "home_att[18]", "home_att[19]", "home_att[20]"
    ))
home_att_mean = home_att_sum$summary[,'mean']

home_def_sum = summary(fit, pars = c("home_def[1]", "home_def[2]", "home
    _def[3]", "home_def[4]", "home_def[5]", "home_def[6]", "home_def[7]",
     "home_def[8]", "home_def[9]", "home_def[10]", "home_def[11]", "home_
    def[12]", "home_def[13]", "home_def[14]", "home_def[15]", "home_def
    [16]", "home_def[17]", "home_def[18]", "home_def[19]", "home_def[20]"
    ))
home_def_mean = home_def_sum$summary[,'mean']

new_labels=c("ARS", "BOU", "BUR", "CHE", "CRY", "EVE", "HUL", "LEI", "
    LIV", "MCI", "MUN", "MBO", "SOU", "STO", "SUN", "SWA", "TOT", "WAT",
    "WBA", "WHU")

cols = c('red','firebrick4','brown','darkblue','blue2','darkblue','
    orange','dodgerblue2','red','deepskyblue','firebrick','firebrick1','
    firebrick1','red','red','gray0','blue4', 'gold1', 'darkblue' ,'brown4
    ')

home_data = data.frame(home_att_mean, home_def_mean, new_labels)

plot(home_att_mean, home_def_mean, col= cols, xlab = 'Attack', ylab = '
    Defence', main='Home Effects', xlim=c(-0.8,1), ylim=c(-0.5,0.4))
abline(v=0); abline(h=0)
text(home_att_mean, home_def_mean, labels = home_data$new_labels, cex
    =0.7, pos=2)
text(0.75, -0.47, "Good Attack, Good Defence", cex=0.6)
text(-0.65, 0.38, "Bad Attack, Bad Defence", cex=0.6)

### Figure 19

away_att_sum = summary(fit, pars = c("away_att[1]", "away_att[2]", "away
    _att[3]", "away_att[4]", "away_att[5]", "away_att[6]", "away_att[7]",
     "away_att[8]", "away_att[9]", "away_att[10]", "away_att[11]", "away_
    att[12]", "away_att[13]", "away_att[14]", "away_att[15]", "away_att
    [16]", "away_att[17]", "away_att[18]", "away_att[19]", "away_att[20]"
    ))
```

```r
away_att_mean = away_att_sum$summary[,'mean']

away_def_sum = summary(fit, pars = c("away_def[1]", "away_def[2]", "away_def[3]", "away_def[4]", "away_def[5]", "away_def[6]", "away_def[7]", "away_def[8]", "away_def[9]", "away_def[10]", "away_def[11]", "away_def[12]", "away_def[13]", "away_def[14]", "away_def[15]", "away_def[16]", "away_def[17]", "away_def[18]", "away_def[19]", "away_def[20]"))
away_def_mean = away_def_sum$summary[,'mean']

new_labels=c("ARS", "BOU", "BUR", "CHE", "CRY", "EVE", "HUL", "LEI", "LIV", "MCI", "MUN", "MBO", "SOU", "STO", "SUN", "SWA", "TOT", "WAT", "WBA", "WHU")

cols = c('red','firebrick4','brown','darkblue','blue2','darkblue','orange','dodgerblue2','red','deepskyblue','firebrick','firebrick1','firebrick1','red','red','gray0','blue4', 'gold1', 'darkblue' ,'brown4')

away_data = data.frame(away_att_mean, away_def_mean, new_labels)

plot(away_att_mean, away_def_mean, col= cols, xlab = 'Attack', ylab = 'Defence', main='Away Effects', xlim=c(-0.65,0.8), ylim=c(-0.5,0.475))
abline(v=0); abline(h=0)
text(away_att_mean, away_def_mean, labels = away_data$new_labels, cex=0.7, pos=2)
text(0.675, -0.475, "Good Attack, Good Defence", cex=0.6)
text(-0.5, 0.45, "Bad Attack, Bad Defence", cex=0.6)

# Figure 23

attack_average = (home_att_mean + away_att_mean)/2
defence_average = (home_def_mean + away_def_mean)/2

new_labels=c("ARS", "BOU", "BUR", "CHE", "CRY", "EVE", "HUL", "LEI", "LIV", "MCI", "MUN", "MBO", "SOU", "STO", "SUN", "SWA", "TOT", "WAT", "WBA", "WHU")

cols = c('red','firebrick4','brown','darkblue','blue2','darkblue','orange','dodgerblue2','red','deepskyblue','firebrick','firebrick1','firebrick1','red','red','gray0','blue4', 'gold1', 'darkblue' ,'brown4')

overall_data = data.frame(attack_average, defence_average, new_labels)

plot(attack_average, defence_average, col= cols, xlab = 'Attack', ylab = 'Defence', main='Overall Effects', xlim=c(-0.6,0.6))
abline(v=0); abline(h=0)
```

```
text(attack_average, defence_average, labels = overall_data$new_labels,
    cex=0.7, pos=2)
text(0.485, -0.035, "Good Attack, Good Defence", cex=0.6)
text(-0.525, 0.325, "Bad Attack, Bad Defence", cex=0.6)
```

I.3 Simulating from the priors from the Negative Binomial model

```
number_of_simulations=10000
# simulating mu parameters
mu_home_att = rnorm(number_of_simulations, mean = 0.2, sd = 1)
mu_away_att = rnorm(number_of_simulations, mean = 0, sd = 1)
mu_home_def = rnorm(number_of_simulations, mean = -0.2, sd = 1)
mu_away_def = rnorm(number_of_simulations, mean = 0, sd = 1)

hist(mu_home_att, breaks=50, xlim=c(-3,3))
hist(mu_away_att, breaks=50, xlim=c(-3,3))
hist(mu_home_def, breaks=50, xlim=c(-3,3))
hist(mu_away_def, breaks=50, xlim=c(-3,3))

# simulating sigma2 and phi parameters
sigma2_att = rgamma(number_of_simulations, 10, 10)
sigma2_def = rgamma(number_of_simulations, 10, 10)
phi_home = rgamma(number_of_simulations, 2.5, 0.05)
phi_away = rgamma(number_of_simulations, 2.5, 0.05)

hist(sigma2_att, breaks=50)
hist(sigma2_def, breaks=50)
hist(phi_home, breaks=50)
hist(phi_away, breaks=50)

# simulating attack and defence parameters for each team
home_att = c()
home_def = c()
away_att = c()
away_def = c()
for (i in 1:number_of_simulations){
  home_att[i]=rnorm(1, mean = mu_home_att, sd = sqrt(sigma2_att))
  away_att[i]=rnorm(1, mean = mu_away_att, sd = sqrt(sigma2_att))
  home_def[i]=rnorm(1, mean = mu_home_def, sd = sqrt(sigma2_def))
  away_def[i]=rnorm(1, mean = mu_away_def, sd = sqrt(sigma2_def))
}

hist(home_att, breaks=50, xlim=c(-4,4))
hist(away_att, breaks=50, xlim=c(-4,4))
hist(home_def, breaks=50, xlim=c(-4,4))
hist(away_def, breaks=50, xlim=c(-4,4))

# fitting in log linear model
```

```
log_mu_1 = c()
log_mu_2 = c()
for (i in 1:number_of_simulations){
  log_mu_1[i] = home_att[i] + away_def[i]
  log_mu_2[i] = away_att[i] + home_def[i]
}

hist(log_mu_1, breaks=50, xlim=c(-5,5))
hist(log_mu_2, breaks=50, xlim=c(-5,5))

# getting the scores by taking a random sample from a Poisson
    distribution
y1=c()
y2=c()
for (i in 1:number_of_simulations){
  y1[i]=rnbinom(1, size = phi_home[i], mu = exp(log_mu_1[i]))
  y2[i]=rnbinom(1, size = phi_away[i], mu = exp(log_mu_2[i]))
}

hist(y1, breaks=50)
hist(y2, breaks=50)

scores=data.frame(y1, y2)
```

I.4 Extra Tables and Figures for Section 6

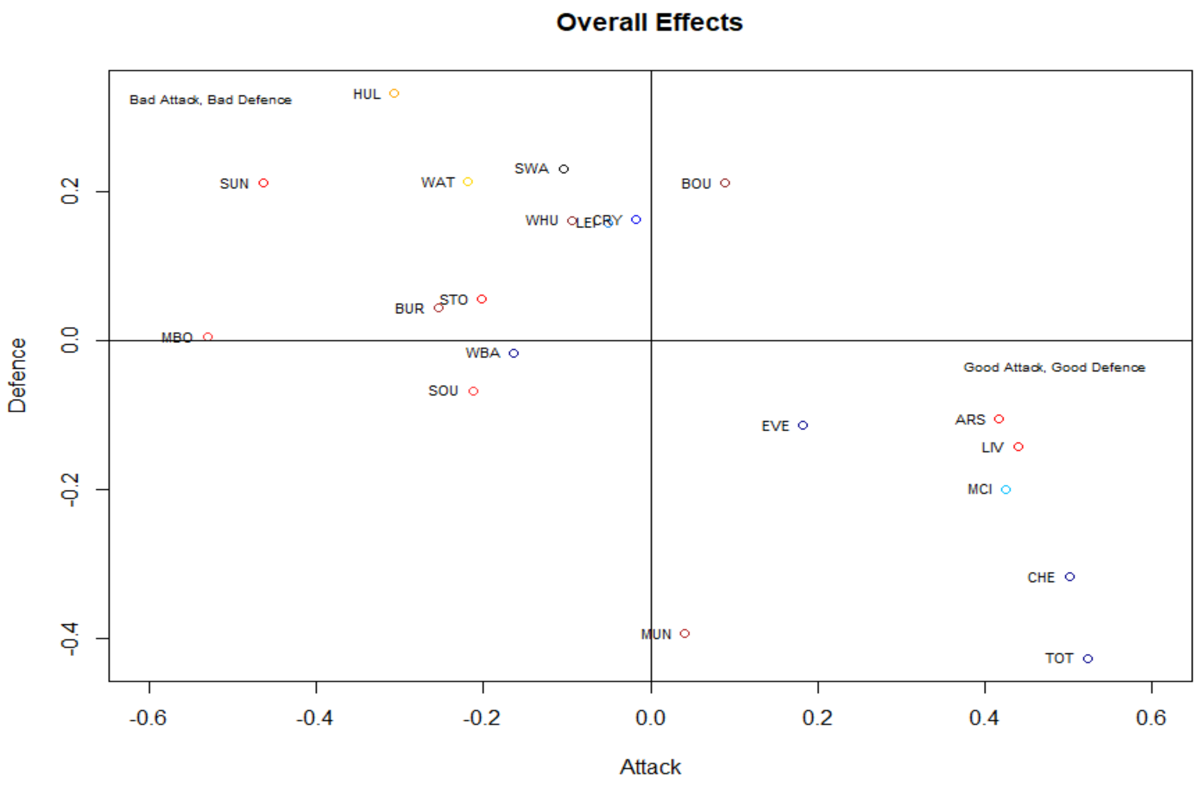


Figure 23: A plot of the posterior means of the attack and defence parameters for each team

	Home Attack					Home Defence				
Team	mean	sd	2.5%	50%	97.5%	mean	sd	2.5%	50%	97.5%
ARS	0.344	0.176	-0.012	0.347	0.675	-0.174	0.181	-0.548	-0.167	0.165
BOU	0.227	0.185	-0.149	0.231	0.578	0.175	0.158	-0.139	0.177	0.476
BUR	-0.125	0.206	-0.534	-0.121	0.270	-0.076	0.173	-0.425	-0.072	0.256
CHE	0.706	0.152	0.409	0.706	1.005	-0.149	0.178	-0.512	-0.144	0.185
CRY	-0.210	0.209	-0.619	-0.208	0.193	0.084	0.164	-0.245	0.085	0.399
EVE	0.427	0.168	0.087	0.432	0.744	-0.191	0.182	-0.568	-0.184	0.150
HUL	-0.028	0.201	-0.431	-0.025	0.353	0.284	0.152	-0.015	0.286	0.580
LEI	0.085	0.195	-0.306	0.089	0.454	0.071	0.163	-0.254	0.073	0.385
LIV	0.498	0.162	0.169	0.503	0.806	-0.113	0.177	-0.471	-0.108	0.224
MCI	0.271	0.179	-0.093	0.275	0.609	-0.137	0.180	-0.506	-0.132	0.203
MUN	-0.145	0.205	-0.551	-0.142	0.249	-0.312	0.200	-0.746	-0.300	0.043
MBO	-0.521	0.211	-0.964	-0.517	-0.117	0.008	0.167	-0.328	0.009	0.330
SOU	-0.524	0.210	-0.963	-0.520	-0.123	-0.035	0.171	-0.378	-0.031	0.293
STO	-0.218	0.207	-0.624	-0.217	0.184	0.042	0.167	-0.292	0.044	0.361
SUN	-0.557	0.216	-1.022	-0.549	-0.153	0.271	0.152	-0.027	0.273	0.566
SWA	-0.079	0.204	-0.488	-0.076	0.311	0.274	0.153	-0.026	0.276	0.568
TOT	0.540	0.158	0.219	0.544	0.840	-0.402	0.224	-0.902	-0.381	-0.015
WAT	-0.163	0.209	-0.576	-0.161	0.238	0.167	0.159	-0.147	0.169	0.469
WBA	-0.091	0.202	-0.496	-0.087	0.295	-0.014	0.169	-0.352	-0.011	0.312
WHU	-0.437	0.208	-0.846	-0.439	-0.029	0.226	0.155	-0.081	0.229	0.526

Table 22: Summary of the home parameter results from the implementation the Negative Binomial model

	Away Attack					Away Defence				
Team	mean	sd	2.5%	50%	97.5%	mean	sd	2.5%	50%	97.5%
ARS	0.490	0.152	0.180	0.495	0.775	-0.039	0.184	-0.410	-0.035	0.311
BOU	-0.048	0.202	-0.455	-0.044	0.336	0.248	0.160	-0.078	0.253	0.547
BUR	-0.384	0.223	-0.840	-0.379	0.040	0.162	0.167	-0.175	0.166	0.477
CHE	0.294	0.175	-0.063	0.300	0.618	-0.486	0.234	-0.993	-0.468	-0.074
CRY	0.176	0.187	-0.205	0.180	0.530	0.241	0.160	-0.086	0.246	0.540
EVE	-0.062	0.203	-0.473	-0.059	0.324	-0.037	0.184	-0.408	-0.033	0.315
HUL	-0.585	0.250	-1.122	-0.568	-0.134	0.380	0.148	0.084	0.381	0.672
LEI	-0.185	0.211	-0.608	-0.181	0.213	0.245	0.160	-0.080	0.250	0.546
LIV	0.380	0.166	0.040	0.386	0.687	-0.173	0.194	-0.566	-0.169	0.195
MCI	0.579	0.145	0.289	0.579	0.867	-0.262	0.198	-0.667	-0.257	0.113
MUN	0.227	0.180	-0.139	0.232	0.564	-0.476	0.227	-0.967	-0.459	-0.072
MBO	-0.539	0.242	-1.056	-0.526	-0.097	0.002	0.179	-0.360	0.006	0.343
SOU	0.099	0.193	-0.290	0.103	0.465	-0.100	0.185	-0.472	-0.096	0.255
STO	-0.186	0.211	-0.608	-0.182	0.216	0.069	0.173	-0.279	0.073	0.397
SUN	-0.371	0.224	-0.831	-0.366	0.052	0.152	0.167	-0.185	0.155	0.4656
SWA	-0.129	0.210	-0.548	-0.125	0.271	0.189	0.165	-0.147	0.194	0.497
TOT	0.506	0.150	0.199	0.511	0.790	-0.452	0.225	-0.937	-0.437	-0.047
WAT	-0.274	0.218	-0.712	-0.270	0.139	0.259	0.158	-0.064	0.264	0.555
WBA	-0.236	0.213	-0.664	-0.232	0.172	-0.020	0.182	-0.386	-0.016	0.326
WHU	0.250	0.180	-0.117	0.254	0.586	0.097	0.172	-0.247	0.100	0.421

Table 23: Summary of the away parameter results from the implementation the Negative Binomial model

J Model Assessment

The following code is used in all of the following functions in this section to fit a stan model (`fit_model`), obtain the mode of a vector (which is used to find the maximum apriori estimate) (`get_mode`) and to obtain probabilities of a match or predict a score (`predict_game`).

```r
fit_model = function(stan_file, data, iterations, chains) {
  # function to fit a stan model
  fitting_model = stan_model(file = stan_file)
  return(sampling(object = fitting_model, data = data, iter = iterations
    , chains = chains, control = list(max_treedepth = 15, adapt_delta =
    0.99)))
}

getmode = function(vector) {
  # function to find the mode of a vector
   uniqv = unique(vector)
   uniqv[which.max(tabulate(match(vector, uniqv)))]
}

predict_game = function(model_fit, home_index, away_index, data, model,
    scores = F){
  # function to predict a game between two teams depending on what model
     we have fitted
  # NB = negative binomial ; BB = model given by Baio & Blangiardo
  # scores = T if want to return prediction for goal scored, but by
    default returns posterior probabilities
  list_of_draws = extract(model_fit)
  number_of_samples = length(list_of_draws$lp__)
  if (model == 'NB') {
    # extracting the samples that we need
    home_att = list_of_draws$home_att[,home_index]
    home_def = list_of_draws$home_def[,home_index]
    away_att = list_of_draws$away_att[,away_index]
    away_def = list_of_draws$away_def[,away_index]
    size_home = list_of_draws$phi_home
    size_away = list_of_draws$phi_away

    # creating the log_mu parameters
    log_mu1 = home_att + away_def
    log_mu2 = home_def + away_att

    # simulating from a negative binomial distribution to obtain
    predictive distribution
    y1 = NULL; y2 = NULL
    for (i in 1:number_of_samples) {
```

```
    y1[i] = rnbinom(1, size = size_home, mu = exp(log_mu1[i]))
    y2[i] = rnbinom(1, size = size_away, mu = exp(log_mu2[i]))
  }
} else if (model == 'BB') {
  # extracting the samples that we need
  home = list_of_draws$home
  att_home = list_of_draws$att[,home_index]
  att_away = list_of_draws$att[,away_index]
  def_home = list_of_draws$def[,home_index]
  def_away = list_of_draws$def[,away_index]

  # creating the log_theta parameters
  log_theta1 = home + att_home + def_away
  log_theta2 = att_away + def_home

  # simulating from a Poisson distribution to obtain predictive
 distribution
  y1 = NULL; y2 = NULL
  for (i in 1:number_of_samples) {
    y1[i] = rpois(1, lambda = exp(log_theta1[i]))
    y2[i] = rpois(1, lambda = exp(log_theta2[i]))
  }
}

teams = sort(as.character(unique(c(as.vector(data$HomeTeam), as.vector
 (data$AwayTeam)))))

# calculating the estimated scores (posterior predictive mean of goals
  scored)
score = data.frame(getmode(y1), getmode(y2))
colnames(score) = c('FTHG', 'FTAG')

# calculating the estimated probabilities of events in data frame
 format
probabilities = data.frame(sum(round(y1,1)>round(y2,1))/number_of_
 samples,
                           sum(round(y1,1)==round(y2,1))/number_of_
 samples,
                           sum(round(y1,1)<round(y2,1))/number_of_
 samples)
colnames(probabilities) = c(teams[home_index], 'Draw', teams[away_
 index])

if (scores == T) {
  # return posterior predictive mean of goals scored by each team
  return(score)
} else {
  # return data frame of probabilities
  return(probabilities)
}
```

J.1 Cross Validation

```
WLD_validate = function(stan_file, test_set, train_set, model) {
  # Win-Lose-Draw validate
  # function to validate the correct outcome of a match by choosing the
   outcome with the highest probability
  # returns a vector of TRUEs and FALSEs for where the model predicted
   the right outcome of the game
  # to find cross validation score, take the mean of this vector (i.e.
   percentage of correct estimates)

  # building Stan data object for the training data set
  training_data = list(nteams = length(as.character(sort(unique(c(as.
   vector(train_set$HomeTeam), as.vector(train_set$AwayTeam)))))),
                       ngames = nrow(train_set),
                       home_team = train_set$HomeIndex,
                       away_team = train_set$AwayIndex,
                       home_goals = train_set$FTHG,
                       away_goals = train_set$FTAG)

  # fitting a stan model to our training data set
  train = fit_model(stan_file = stan_file, training_data, iterations =
   20000, chains = 4)

  # choosing event with highest probability
  # add T to prediction_log if prediction correct and F otherwise
  prediction_log = NULL
  for (i in 1:nrow(test_set)) {
    probabilities = predict_game(model_fit = train,
                                 home_index = test_set[i,]$HomeIndex,
                                 away_index = test_set[i,]$AwayIndex,
                                 data = train_set,
                                 model = model)

    # determine whether the highest probable event according to the
   model is the observed outcome
    if (test_set[i,]$FTHG > test_set[i,]$FTAG) {
      prediction_log = c(prediction_log, (probabilities[,1] == max(
   probabilities)))
    } else if (test_set[i,]$FTHG == test_set[i,]$FTAG) {
      prediction_log = c(prediction_log, (probabilities[,2] == max(
   probabilities)))
    } else if (test_set[i,]$FTHG < test_set[i,]$FTAG) {
      prediction_log = c(prediction_log, (probabilities[,3] == max(
   probabilities)))
```

```r
    }
  }

  # returning the vector of TRUEs and FALSEs for each game prediction
  return(prediction_log)
}

MCCV = function(stan_file, iterations, test_percentage, data, model) {
  # function to obtain MCCV score
  # the training data is randomly split by taking a percentage of the data
  # we repeat this for number of iterations to obtain a set of CV scores - call it Monte Carlo CV

  MCCV_scores = NULL
  for (i in 1:iterations) {
    # splitting up the data into (test_percentage)% and (1-test_percentage)% randomly
    random = sample(nrow(data), test_percentage*nrow(data))
    test_set = data[random,]
    train_set = data[-random,]

    # finding a CV score for this particular test/train set combination and adding this to the MCCV scores
    MCCV_scores[i] = mean(WLD_validate(stan_file, test_set, train_set, model))

    # print progress of the algorithm
    print(paste(i, '. MCCV Score: ', MCCV_scores[i]))
  }

  # return set of CV scores
  return(MCCV_scores)
}

seqCV = function(stan_file, data, model, minimum_games) {
  # function to obtain CV score
  # training data is always past games
  # assume that data is already ordered with the older games at the top

  game_predictions_log = NULL

  # splitting the data set to games we want to always use as the training set and the test set
  total_number_of_games = nrow(data)
  train_set = data[1:minimum_games,]
  test_set = data[(minimum_games+1):nrow(data),]
  data = data[(minimum_games+1):nrow(data),]

  # set loop to carry on until there are still games to try and predict
  while(!(nrow(train_set)==total_number_of_games)){
    # to choose test set, we go through each game carry on until we find
     a repeat in team
    # get the games that we can use as our test set by going through
   each game until we count a team twice
    # if we add only 1 team or 0 teams, then we have found a repeating
   team
    teams = NULL; additions = 2; i=1
    while (additions == 2) {
      additions = 0
      if (!(is.element(data[i,]$HomeIndex, teams))) {
        teams = c(teams, data[i,]$HomeIndex)
        additions = additions + 1
      }
      if (!(is.element(data[i,]$AwayIndex, teams))) {
        teams = c(teams, data[i,]$AwayIndex)
        additions = additions + 1
      }
      if (additions == 2) {
        i = i + 1
      }
    }

    # test_set now becomes the data frame with the games where no team
   plays twice in
    # re-label the data as the data minus the train_set games
    test_set = data[1:i-1,]
    data = data[i:nrow(data),]

    # predicting games by choosing the highest probable outcome
    game_predictions_log = c(game_predictions_log, WLD_validate(stan_
   file, test_set, train_set, model))

    # print progress of the algorithm
    print(paste('There are ', nrow(data), ' games left in the data set')
   )

    # train_set now becomes all the data and games that have happened
   before the test_set games
    train_set = rbind(train_set, data[1:i-1,])
  }

  # return the vector of TRUEs and FALSEs for each game prediction
  return(game_predictions_log)
}

```

```
### Calculating MCCV score for BB model

BB_MCCV = MCCV(stan_file = 'sum_to_zero.stan', iterations = 50, test_
    percentage = 0.2, data = E0_2016, model = 'BB')
mean(BB_MCCV) # mean
sd(BB_MCCV) # standard deviation
sd(BB_MCCV)/sqrt(length(BB_MCCV)) # standard error

### Calculating MCCV score for NB model

NB_MCCV = MCCV(stan_file = 'model_nb.stan', iterations = 50, test_
    percentage = 0.2, data = E0_2016, model = 'NB')
mean(NB_MCCV) # mean
sd(NB_MCCV) # standard deviation
sd(NB_MCCV)/sqrt(length(NB_MCCV)) # standard error

### Calculating SeqCV score for BB model

BB_SeqCV = seqCV(stan_file = 'sum_to_zero.stan', data = E0_2016, model =
     'BB', minimum_games=10)
mean(BB_SeqCV) # sequential CV score

### Calculating SeqCV score for NB model

NB_SeqCV = seqCV(stan_file = 'model_nb.stan', data = E0_2016, model = '
    NB', minimum_games=10)
mean(NB_SeqCV) # sequential CV score
```

J.2 Brier Score

```
calculate_BS = function(stan_file, test_set, train_set, model) {
  # BS = 1/N sum(O_i - p_i) where O_i = 1 if event occured and 0
    otherwise and p_i is probability of event
  # function to calculate the Brier Score for a set of games given in
    the test_set
  # returns the Brier score for the test_set

  # building Stan data object for the training data set
  training_data = list(nteams = length(as.character(sort(unique(c(as.
    vector(train_set$HomeTeam), as.vector(train_set$AwayTeam)))))),
                       ngames = nrow(train_set),
                       home_team = train_set$HomeIndex,
                       away_team = train_set$AwayIndex,
                       home_goals = train_set$FTHG,
                       away_goals = train_set$FTAG)

  # fitting a stan model to our training data set
  train = fit_model(stan_file = stan_file, training_data, iterations =
    20000, chains = 4)

  # calulating Brier score by choosing event with highest probability
    and seeing if the event occured
  Brier_score = 0
  for (i in 1:nrow(test_set)) {
    probabilities = predict_game(model_fit = train,
                                 home_index = test_set[i,]$HomeIndex,
                                 away_index = test_set[i,]$AwayIndex,
                                 data = train_set,
                                 model = model)

    # need to find our forecast (i.e. highest probable event) and then
    check if it occured
    # then we add appropriately to the Brier score
    if (test_set[i,]$FTHG > test_set[i,]$FTAG) {
      Brier_score = Brier_score + (1-probabilities[,1])^(2) + (0-
    probabilities[,2])^(2) + (0-probabilities[,3])^(2)
    } else if (test_set[i,]$FTHG == test_set[i,]$FTAG) {
      Brier_score = Brier_score + (0-probabilities[,1])^(2) + (1-
    probabilities[,2])^(2) + (0-probabilities[,3])^(2)
    } else if  (test_set[i,]$FTHG < test_set[i,]$FTAG) {
      Brier_score = Brier_score + (0-probabilities[,1])^(2) + (0-
    probabilities[,2])^(2) + (1-probabilities[,3])^(2)
    }
  }

  Brier_score = (Brier_score/nrow(test_set))

  # return Brier score
  return(Brier_score)
}

MCBrier = function(stan_file, iterations, test_percentage, data, model)
   {
  # function to obtain MCBrier score
  # the training data is randomly split by taking a percentage of the
    data
  # we repeat this for number of iterations to obtain a set of Brier
    scores - call it Monte Carlo Brier score

  Brier_scores = NULL
  for (i in 1:iterations) {
    # splitting up the data into (test_percentage)% and (1-test_
    percentage)% randomly
  random = sample(nrow(data), test_percentage*nrow(data))
    test_set = data[random,]
    train_set = data[-random,]

    # finding a Brier score for this particular test/train set
   combination and adding this to the Brier scores
    Brier_scores[i] = calculate_BS(stan_file, test_set, train_set, model
   )

    # print progress of the algorithm
    print(paste(i, '. Brier Score: ', Brier_scores[i]))
  }

  # return set of Brier scores
  return(Brier_scores)
}

seqBrier = function(stan_file, data, model, minimum_games) {
  # function to obtain sequential Brier score
  # training data is always past games
  # assume that data is already ordered with the older games at the top

  Brier_score = 0
  total_number_of_test_games = nrow(data) - minimum_games

  # splitting the data set to games we want to always use as the
   training set and the test set
  total_number_of_games = nrow(data)
  train_set = data[1:minimum_games,]
  test_set = data[(minimum_games+1):nrow(data),]
  data = data[(minimum_games+1):nrow(data),]

  # set loop to carry on until there are still games to try and predict
  while(!(nrow(train_set)==total_number_of_games)){
    # to choose test set, we go through each game carry on until we find
    a repeat in team
    # get the games that we can use as our test set by going through
   each game until we count a team twice
    # if we add only 1 team or 0 teams, then we have found a repeating
   team
    teams = NULL; additions = 2; i=1
    while (additions == 2) {
      additions = 0
      if (!(is.element(data[i,]$HomeIndex, teams))) {
        teams = c(teams, data[i,]$HomeIndex)
        additions = additions + 1
      }
      if (!(is.element(data[i,]$AwayIndex, teams))) {
        teams = c(teams, data[i,]$AwayIndex)
        additions = additions + 1
      }
      if (additions == 2) {
```

```
      i = i + 1
    }
  }

  # test_set now becomes the data frame with the games where no team
  plays twice in
  # re-label the data as the data minus the train_set games
  test_set = data[1:i-1,]
  data = data[i:nrow(data),]

  # calculating BS for test data set and adding to the overall Brier
  score
  # but need to multiply by nrow(test_set) and then divide by total
  number of games at the end
  Brier_score = Brier_score + (nrow(test_set)*calculate_BS(stan_file,
  test_set, train_set, model))

  # print progress of the algorithm
  print(paste('There are ', nrow(data), ' games left in the data set')
  )

  # train_set now becomes all the data and games that have happened
  before the test_set games
  train_set = rbind(train_set, data[1:i-1,])
 }

 Brier_score = (Brier_score/total_number_of_test_games)
 # return the Brier score
 return(Brier_score)
}

### Calculating the Brier score for BB model

BB_SeqBrier = seqBrier(stan_file = 'sum_to_zero.stan', data = E0_2016,
  model = 'BB', minimum_games = 10)

### Calculating the Brier score for NB model

NB_SeqBrier = seqBrier(stan_file = 'model_nb.stan', data = E0_2016,
  model = 'NB', minimum_games = 10)
```

J.3 Rank Probability Score

```
calculate_RPS = function(stan_file, test_set, train_set, model) {
 # function to calculate the Rank Probability Score for a set of games
  given in the test_set
 # returns the sum of the RPS scores for each game by default, if mean
  ==T, then returns the mean RPS per game

  # building Stan data object for the training data set
  training_data = list(nteams = length(as.character(sort(unique(c(as.
   vector(train_set$HomeTeam), as.vector(train_set$AwayTeam)))))),
                       ngames = nrow(train_set),
                       home_team = train_set$HomeIndex,
                       away_team = train_set$AwayIndex,
                       home_goals = train_set$FTHG,
                       away_goals = train_set$FTAG)

  # fitting a stan model to our training data set
  train = fit_model(stan_file = stan_file, training_data, iterations =
   20000, chains = 4)

  # calulating RPS
  RPS = 0
  for (i in 1:nrow(test_set)) {
    probabilities = predict_game(model_fit = train,
                                 home_index = test_set[i,]$HomeIndex,
                                 away_index = test_set[i,]$AwayIndex,
                                 data = train_set,
                                 model = model)

    # calculating RPS formula and adding to RPS variable
    sum_probabilities_vector = c(sum(probabilities[1]), sum(
   probabilities[1:2]), sum(probabilities[1:3]))
    if (test_set[i,]$FTHG > test_set[i,]$FTAG) {
      observed_vector = c(1,1,1)
    } else if (test_set[i,]$FTHG == test_set[i,]$FTAG) {
      observed_vector = c(0,1,1)
    } else if  (test_set[i,]$FTHG < test_set[i,]$FTAG) {
      observed_vector = c(0,0,1)
    }
    RPS = RPS + ((sum_probabilities_vector[1] - observed_vector[1])^(2)
   +
                 (sum_probabilities_vector[2] - observed_vector[2])^(2)
   +
                 (sum_probabilities_vector[3] - observed_vector[3])^(2))
   /2
  }

  # return total sum of RPS for the games
  return(RPS)
}

seqRPS = function(stan_file, data, model, minimum_games) {
  # function to obtain RPS
  # training data is always past games
  # assume that data is already ordered with the older games at the top
```

```

RPS = 0
total_number_of_test_games = nrow(data) - minimum_games

# splitting the data set to games we want to always use as the
 training set and the test set
total_number_of_games = nrow(data)
train_set = data[1:minimum_games,]
test_set = data[(minimum_games+1):nrow(data),]
data = data[(minimum_games+1):nrow(data),]

# set loop to carry on until there are still games to try and predict
while(!(nrow(train_set)==total_number_of_games)){
  # to choose test set, we go through each game carry on until we find
  a repeating in team
  # get the games that we can use as our test set by going through
 each game until we count a team twice
  # if we add only 1 team or 0 teams, then we have found a repeating
 team
  teams = NULL; additions = 2; i=1
  while (additions == 2) {
    additions = 0
    if (!(is.element(data[i,]$HomeIndex, teams))) {
      teams = c(teams, data[i,]$HomeIndex)
      additions = additions + 1
    }
    if (!(is.element(data[i,]$AwayIndex, teams))) {
      teams = c(teams, data[i,]$AwayIndex)
      additions = additions + 1
    }
    if (additions == 2) {
      i = i + 1
    }
  }

  # test_set now becomes the data frame with the games where no team
 plays twice in
  # re-label the data as the data minus the train_set games
  test_set = data[1:i-1,]
  data = data[i:nrow(data),]

  # calculating RPS for test data set and adding to the overall RPS
  RPS = RPS + calculate_RPS(stan_file, test_set, train_set, model,
 mean = F)

  # print progress of the algorithm
  print(paste('There are ', nrow(data), ' games left in the data set')
 )
```

```

    # train_set now becomes all the data and games that have happened
    before the test_set games
    train_set = rbind(train_set, data[1:i-1,])
  }

  # return the Rank Probability Score - to get mean RPS for each game,
   need to divide by nrow(data) - minimum games
  return(RPS)
}

### Calculating the RPS for BB model

BB_SeqRPS = seqRPS(stan_file = 'sum_to_zero.stan', data = E0_2016, model
   = 'BB', minimum_games = 10)

BB_SeqRPS/370 # average RPS per game for BB model

### Calculating the RPS for NB model

NB_SeqRPS = seqRPS(stan_file = 'model_nb.stan', data = E0_2016, model =
   'NB', minimum_games = 10)

NB_SeqRPS/370 # average RPS per game for NB model
```

J.4 Betting Assessment

```
calculate_PL = function(stan_file, test_set, train_set, model, wager) {
  # function to calculate the profit/loss for a set of games given in
   the test_set
  # returns the profit/loss given a wager bet on each game

  # building Stan data object for the training data set
  training_data = list(nteams = length(as.character(sort(unique(c(as.
   vector(train_set$HomeTeam), as.vector(train_set$AwayTeam)))))),
                       ngames = nrow(train_set),
                       home_team = train_set$HomeIndex,
                       away_team = train_set$AwayIndex,
                       home_goals = train_set$FTHG,
                       away_goals = train_set$FTAG)

  # fitting a stan model to our training data set
  train = fit_model(stan_file = stan_file, training_data, iterations =
   20000, chains = 4)

  # calulating P/L by choosing event with highest probability and seeing
    if the event occured
  profit_loss = c()
  for (i in 1:nrow(test_set)) {
```

```r
    probabilities = predict_game(model_fit = train,
                                 home_index = test_set[i,]$HomeIndex,
                                 away_index = test_set[i,]$AwayIndex,
                                 data = train_set,
                                 model = model)

    # need to find highest probable event and then add to returns if the forecast is correct
    # add profit/loss returns to profit_loss
    if ((probabilities[,1] == max(probabilities)) & (test_set[i,]$FTHG > test_set[i,]$FTAG)) {
      profit_loss = c(profit_loss, wager*test_set[i,]$B365H - wager) # forecast for home win correct
    } else if ((probabilities[,2] == max(probabilities)) & (test_set[i,]$FTHG == test_set[i,]$FTAG)) {
      profit_loss = c(profit_loss, wager*test_set[i,]$B365D - wager) # forecast for draw correct
    } else if ((probabilities[,3] == max(probabilities)) & (test_set[i,]$FTHG < test_set[i,]$FTAG)) {
      profit_loss = c(profit_loss, wager*test_set[i,]$B365A - wager) # forecast for away win correct
    } else {
      profit_loss = c(profit_loss, - wager) # forecast was incorrect so no returns, just loss
    }
  }

  # return profit/loss
  return(profit_loss)
}

seqBetting = function(stan_file, data, model, minimum_games) {
  # function to obtain profit loss when betting on the highest probable event at closing price at B365
  # training data is always past games
  # assume that data is already ordered with the older games at the top

  profit_loss = c()

  # splitting the data set to games we want to always use as the training set and the test set
  total_number_of_games = nrow(data)
  train_set = data[1:minimum_games,]
  test_set = data[(minimum_games+1):nrow(data),]
  data = data[(minimum_games+1):nrow(data),]

  # set loop to carry on until there are still games to try and predict
  while(!(nrow(train_set)==total_number_of_games)){
    # to choose test set, we go through each game carry on until we find a repeat in team
    # get the games that we can use as our test set by going through each game until we count a team twice
    # if we add only 1 team or 0 teams, then we have found a repeating team
    teams = NULL; additions = 2; i=1
    while (additions == 2) {
      additions = 0
      if (!(is.element(data[i,]$HomeIndex, teams))) {
        teams = c(teams, data[i,]$HomeIndex)
        additions = additions + 1
      }
      if (!(is.element(data[i,]$AwayIndex, teams))) {
        teams = c(teams, data[i,]$AwayIndex)
        additions = additions + 1
      }
      if (additions == 2) {
        i = i + 1
      }
    }

    # test_set now becomes the data frame with the games where no team plays twice in
    # re-label the data as the data minus the train_set games
    test_set = data[1:i-1,]
    data = data[i:nrow(data),]

    # calculating profit/loss for test data set and adding to the overall profit/loss
    profit_loss = c(profit_loss, calculate_PL(stan_file, test_set, train_set, model, wager = 10))

    # print progress of the algorithm
    print(paste('There are ', nrow(data), ' games left in the data set'))

    # train_set now becomes all the data and games that have happened before the test_set games
    train_set = rbind(train_set, data[1:i-1,])
  }

  # return the final profit/loss for each game
  return(profit_loss)
}

### Calculating the profit/loss by using BB model

```

```
BB_Bet = seqBetting(stan_file = 'sum_to_zero.stan', data = E0_2016,
   model = 'BB', minimum_games=10)
sum(BB_Bet)

# Looking at table of profit/loss returns by using BB model
table(BB_Bet)

### Calculating the profit/loss by using NB model

NB_Bet = seqBetting(stan_file = 'model_nb.stan', data = E0_2016, model =
    'NB', minimum_games=10)
sum(NB_Bet)

# Looking at table of profit/loss returns by using NB model
table(NB_Bet)

### Figure 20

plot(cumsum(NB_Bet), main = 'Profit-Loss Returns for each model',
     xlab = 'Game Number', ylab = 'Profit-Loss (in ?s)', pch='.', col =
   'red', ylim = c(0,1100))
lines(cumsum(NB_Bet), lwd=1, col = 'red')
points(cumsum(BB_Bet), pch='.', col = 'blue')
lines(cumsum(BB_Bet), lwd=1, col = 'blue')
legend('topleft',legend = c('Baio & Blangiardo Poisson Model', 'Negative
    Binomial'), fill = c('blue', 'red'))
```

J.5 Predict League Table

```
get_table = function(data) {
  # function to obtain a league table
  # data passed must be a data frame of results with team names and
    scores passed using column names FTHG (home goals) and FTAG (away
    goals)

  # getting the team names in alphabetical order
  teams = as.character(sort(unique(c(as.vector(data$HomeTeam), as.vector
    (data$AwayTeam)))))

  # creating empty league table
  # GP = games played
  # HW = home win, HD = home draw, HL = home loss, HF = home goals for,
    HA = home goals against, HGD = home goal difference
  # AW = away win, AD = away draw, AL = away loss, AF = away goals for,
    AA = away goals against, AGD = away goal difference
  # W = win, D = draw, L = loss, GF = goals for, GA = goals against, GD
    = goal difference
  table = data.frame(Team = teams, GP = 0, HW = 0, HD = 0, HL = 0, HF =
    0, HA = 0, HGD = 0, AW = 0, AD = 0, AL = 0, AF = 0, AA = 0, AGD = 0,
  W = 0, D = 0, L = 0, GF = 0, GA = 0, GD = 0, Points = 0)

for (i in 1:nrow(data)) {
  # updating the goal tallys
  # adding goals scored by home team
  table[(which(teams == data[i,]$HomeTeam)), 'GF'] = table[(which(
 teams == data[i,]$HomeTeam)), 'GF'] + data[i,]$FTHG
  table[(which(teams == data[i,]$HomeTeam)), 'HF'] = table[(which(
 teams == data[i,]$HomeTeam)), 'HF'] + data[i,]$FTHG
  table[(which(teams == data[i,]$AwayTeam)), 'GA'] = table[(which(
 teams == data[i,]$AwayTeam)), 'GA'] + data[i,]$FTHG
  table[(which(teams == data[i,]$AwayTeam)), 'AA'] = table[(which(
 teams == data[i,]$AwayTeam)), 'AA'] + data[i,]$FTHG
  # adding goals scored by away team
  table[(which(teams == data[i,]$AwayTeam)), 'GF'] = table[(which(
 teams == data[i,]$AwayTeam)), 'GF'] + data[i,]$FTAG
  table[(which(teams == data[i,]$AwayTeam)), 'AF'] = table[(which(
 teams == data[i,]$AwayTeam)), 'AF'] + data[i,]$FTAG
  table[(which(teams == data[i,]$HomeTeam)), 'GA'] = table[(which(
 teams == data[i,]$HomeTeam)), 'GA'] + data[i,]$FTAG
  table[(which(teams == data[i,]$HomeTeam)), 'HA'] = table[(which(
 teams == data[i,]$HomeTeam)), 'HA'] + data[i,]$FTAG
  # updating goal difference for home team
  table[(which(teams == data[i,]$HomeTeam)), 'GD'] = table[(which(
 teams == data[i,]$HomeTeam)), 'GF'] - table[(which(teams == data[i,]$
 HomeTeam)), 'GA']
  table[(which(teams == data[i,]$HomeTeam)), 'HGD'] = table[(which(
 teams == data[i,]$HomeTeam)), 'HF'] - table[(which(teams == data[i,]$
 HomeTeam)), 'HA']
  # updating goal difference for away team
  table[(which(teams == data[i,]$AwayTeam)), 'GD'] = table[(which(
 teams == data[i,]$AwayTeam)), 'GF'] - table[(which(teams == data[i,]$
 AwayTeam)), 'GA']
  table[(which(teams == data[i,]$AwayTeam)), 'AGD'] = table[(which(
 teams == data[i,]$AwayTeam)), 'AF'] - table[(which(teams == data[i,]$
 AwayTeam)), 'AA']

  # updating the point tallys and team records
  if (data[i,]$FTHG > data[i,]$FTAG) {
    # update points
    table[(which(teams == data[i,]$HomeTeam)), 'Points'] = table[(
 which(teams == data[i,]$HomeTeam)), 'Points'] + 3
    # update win / losses
    table[(which(teams == data[i,]$HomeTeam)), 'HW'] = table[(which(
 teams == data[i,]$HomeTeam)), 'HW'] + 1
    table[(which(teams == data[i,]$HomeTeam)), 'W'] = table[(which(
 teams == data[i,]$HomeTeam)), 'W'] + 1
    table[(which(teams == data[i,]$AwayTeam)), 'AL'] = table[(which(
```

```r
teams == data[i,]$AwayTeam)), 'AL'] + 1
    table[(which(teams == data[i,]$AwayTeam)), 'L'] = table[(which(teams == data[i,]$AwayTeam)), 'L'] + 1
    # update games played
    table[(which(teams == data[i,]$HomeTeam)), 'GP'] = table[(which(teams == data[i,]$HomeTeam)), 'GP'] + 1
    table[(which(teams == data[i,]$AwayTeam)), 'GP'] = table[(which(teams == data[i,]$AwayTeam)), 'GP'] + 1
  } else if (data[i,]$FTHG == data[i,]$FTAG) {
    # update points
    table[(which(teams == data[i,]$HomeTeam)), 'Points'] = table[(which(teams == data[i,]$HomeTeam)), 'Points'] + 1
    table[(which(teams == data[i,]$AwayTeam)), 'Points'] = table[(which(teams == data[i,]$AwayTeam)), 'Points'] + 1
    # update draws
    table[(which(teams == data[i,]$HomeTeam)), 'HD'] = table[(which(teams == data[i,]$HomeTeam)), 'HD'] + 1
    table[(which(teams == data[i,]$HomeTeam)), 'D'] = table[(which(teams == data[i,]$HomeTeam)), 'D'] + 1
    table[(which(teams == data[i,]$AwayTeam)), 'AD'] = table[(which(teams == data[i,]$AwayTeam)), 'AD'] + 1
    table[(which(teams == data[i,]$AwayTeam)), 'D'] = table[(which(teams == data[i,]$AwayTeam)), 'D'] + 1
    # update games played
    table[(which(teams == data[i,]$HomeTeam)), 'GP'] = table[(which(teams == data[i,]$HomeTeam)), 'GP'] + 1
    table[(which(teams == data[i,]$AwayTeam)), 'GP'] = table[(which(teams == data[i,]$AwayTeam)), 'GP'] + 1
  } else if (data[i,]$FTHG < data[i,]$FTAG) {
    # update points
    table[(which(teams == data[i,]$AwayTeam)), 'Points'] = table[(which(teams == data[i,]$AwayTeam)), 'Points'] + 3
    # update draws
    table[(which(teams == data[i,]$HomeTeam)), 'HL'] = table[(which(teams == data[i,]$HomeTeam)), 'HL'] + 1
    table[(which(teams == data[i,]$HomeTeam)), 'L'] = table[(which(teams == data[i,]$HomeTeam)), 'L'] + 1
    table[(which(teams == data[i,]$AwayTeam)), 'AW'] = table[(which(teams == data[i,]$AwayTeam)), 'AW'] + 1
    table[(which(teams == data[i,]$AwayTeam)), 'W'] = table[(which(teams == data[i,]$AwayTeam)), 'W'] + 1
    # update games played
    table[(which(teams == data[i,]$HomeTeam)), 'GP'] = table[(which(teams == data[i,]$HomeTeam)), 'GP'] + 1
    table[(which(teams == data[i,]$AwayTeam)), 'GP'] = table[(which(teams == data[i,]$AwayTeam)), 'GP'] + 1
  }
}
```

```r

  # sorting the teams by total points and by goal difference
  table = table[order(-table$Points, -table$GD),]

  # returing table
  return(table)
}

track_points_progress = function(data, results = T, start_values = matrix(data = rep(0, length(as.character(sort(unique(c(as.vector(data$HomeTeam), as.vector(data$AwayTeam)))))))) {
  # function to obtain a data frame that tracks the number of points that a team obtains over a period
  # results is set to T by default - data passed must be a data frame of results with team names and scores passed using column names FTHG (home goals) and FTAG (away goals)
  # if results if F, data passed must be a data frame of outcomes with team names and the final result (HW = home win, D = draw, AW = away win)
  # start values are a vector of points that the team has before the games have been played - by default is zero for each team

  # getting the team names in alphabetical order
  teams = as.character(sort(unique(c(as.vector(data$HomeTeam), as.vector(data$AwayTeam)))))

  # creating a data frame with each team as a column
  # progress = setNames(data.frame(matrix(data = 0, ncol = length(teams), nrow = 1)),teams)
  progress = matrix(data = start_values, ncol = 1, nrow = length(teams))
  rownames(progress) = teams

  # need to make a data frame to count how many games each team has played, so we know at what position in the matrix we want to add the new value of points
  GP = setNames(data.frame(matrix(data = 0, ncol = length(teams), nrow = 1)),teams)

  for (i in 1:nrow(data)) {
   # updating the number of games played
   GP[,(which(teams == data[i,]$HomeTeam))] = GP[,(which(teams == data[i,]$HomeTeam))] + 1
   GP[,(which(teams == data[i,]$AwayTeam))] = GP[,(which(teams == data[i,]$AwayTeam))] + 1

   # making sure that there is a row to add the next points tally for the teams
   if (!(ncol(progress) >= (GP[,(which(teams == data[i,]$HomeTeam))]+1)
 )) {
     progress = cbind(progress, matrix(data=0, ncol = 1, nrow = length(
 teams)))
   }
   if (!(ncol(progress) >= (GP[,(which(teams == data[i,]$AwayTeam))]+1)
 )) {
     progress = cbind(progress, matrix(data=0, ncol = 1, nrow = length(
 teams)))
   }

   # adding to the points tally and adding this value to the next
 column in the matrix for the team
   if (results == T) {
     if (data[i,]$FTHG > data[i,]$FTAG) {
       progress[(which(teams == data[i,]$HomeTeam)), (GP[,(which(teams
 == data[i,]$HomeTeam))]+1)] = progress[(which(teams == data[i,]$
 HomeTeam)),
                                                           (GP[,(which(teams == data[i,]$
 HomeTeam))])] + 3
       progress[(which(teams == data[i,]$AwayTeam)), (GP[,(which(teams
 == data[i,]$AwayTeam))]+1)] = progress[(which(teams == data[i,]$
 AwayTeam)),
                                                           (GP[,(which(teams == data[i,]$
 AwayTeam))])]
     } else if (data[i,]$FTHG == data[i,]$FTAG) {
       progress[(which(teams == data[i,]$HomeTeam)), (GP[,(which(teams
 == data[i,]$HomeTeam))]+1)] = progress[(which(teams == data[i,]$
 HomeTeam)),
                                                           (GP[,(which(teams == data[i,]$
 HomeTeam))])] + 1
       progress[(which(teams == data[i,]$AwayTeam)), (GP[,(which(teams
 == data[i,]$AwayTeam))]+1)] = progress[(which(teams == data[i,]$
 AwayTeam)),
                                                           (GP[,(which(teams == data[i,]$
 AwayTeam))])] + 1
     } else if (data[i,]$FTHG < data[i,]$FTAG) {
       progress[(which(teams == data[i,]$HomeTeam)), (GP[,(which(teams
 == data[i,]$HomeTeam))]+1)] = progress[(which(teams == data[i,]$
 HomeTeam)),
                                                           (GP[,(which(teams == data[i,]$
 HomeTeam))])]
       progress[(which(teams == data[i,]$AwayTeam)), (GP[,(which(teams
 == data[i,]$AwayTeam))]+1)] = progress[(which(teams == data[i,]$
 AwayTeam)),
                                                           (GP[,(which(teams == data[i,]$
 AwayTeam))])] + 3
     }
   } else if (results == F) {
     if (data[i,]$Result == 'HW') {
       progress[(which(teams == data[i,]$HomeTeam)), (GP[,(which(teams
 == data[i,]$HomeTeam))]+1)] = progress[(which(teams == data[i,]$
 HomeTeam)),
                                                           (GP[,(which(teams == data[i,]$
 HomeTeam))])] + 3
       progress[(which(teams == data[i,]$AwayTeam)), (GP[,(which(teams
 == data[i,]$AwayTeam))]+1)] = progress[(which(teams == data[i,]$
 AwayTeam)),
                                                           (GP[,(which(teams == data[i,]$
 AwayTeam))])]
     } else if (data[i,]$Result == 'D') {
       progress[(which(teams == data[i,]$HomeTeam)), (GP[,(which(teams
 == data[i,]$HomeTeam))]+1)] = progress[(which(teams == data[i,]$
 HomeTeam)),
                                                           (GP[,(which(teams == data[i,]$
 HomeTeam))])] + 1
       progress[(which(teams == data[i,]$AwayTeam)), (GP[,(which(teams
 == data[i,]$AwayTeam))]+1)] = progress[(which(teams == data[i,]$
 AwayTeam)),
                                                           (GP[,(which(teams == data[i,]$
 AwayTeam))])] + 1
     } else if (data[i,]$Result == 'AW') {
       progress[(which(teams == data[i,]$HomeTeam)), (GP[,(which(teams
 == data[i,]$HomeTeam))]+1)] = progress[(which(teams == data[i,]$
 HomeTeam)),
                                                           (GP[,(which(teams == data[i,]$
 HomeTeam))])]
       progress[(which(teams == data[i,]$AwayTeam)), (GP[,(which(teams
 == data[i,]$AwayTeam))]+1)] = progress[(which(teams == data[i,]$
 AwayTeam)),
                                                           (GP[,(which(teams == data[i,]$
 AwayTeam))])] + 3
     }
   }
 }
```

```r

  # return points progress matrix
  return(progress)
}

predict_table = function(stan_file, data, model, minimum_games) {
  # function to obtain a prediction of a league table after some games
  # also returns a prediction for how the teams points progress during
   the games in form of a matrix

  # splitting the data set to games we want to always use as the
   training set and the test set
  total_number_of_games = nrow(data)
  train_set = observed_games = data[1:minimum_games,]
  test_set = data[(minimum_games+1):nrow(data),]
  data = data[(minimum_games+1):nrow(data),]

  # getting the current points progress of train set
  points_progress = track_points_progress(observed_games)

  # need to predict scores for the rest of the data set
  game_predictions = NULL

  # set loop to carry on until there are still games to try and predict
  while(!(nrow(train_set)==total_number_of_games)){
    # to choose test set, we go through each game carry on until we find
     a repeat in team
    # get the games that we can use as our test set by going through
   each game until we count a team twice
    # if we add only 1 team or 0 teams, then we have found a repeating
   team
    teams = NULL; additions = 2; i=1
    while (additions == 2) {
      additions = 0
      if (!(is.element(data[i,]$HomeIndex, teams))) {
        teams = c(teams, data[i,]$HomeIndex)
        additions = additions + 1
      }
      if (!(is.element(data[i,]$AwayIndex, teams))) {
        teams = c(teams, data[i,]$AwayIndex)
        additions = additions + 1
      }
      if (additions == 2) {
        i = i + 1
      }
    }

    # test_set now becomes the data frame with the games where no team
```

```
 plays twice in
  # re-label the data as the data minus the train_set games
  test_set = data[1:i-1,]
  data = data[i:nrow(data),]

  # building Stan data object for the training data set
training_data = list(nteams = length(as.character(sort(unique(c(as.
 vector(train_set$HomeTeam), as.vector(train_set$AwayTeam)))))),
                       ngames = nrow(train_set),
                       home_team = train_set$HomeIndex,
                       away_team = train_set$AwayIndex,
                       home_goals = train_set$FTHG,
                       away_goals = train_set$FTAG)

  # fitting a stan model to our training data set
  train = fit_model(stan_file = stan_file, training_data, iterations =
  20000, chains = 4)

  for (i in 1:nrow(test_set)) {
    score = predict_game(model_fit = train,
                         home_index = test_set[i,]$HomeIndex,
                         away_index = test_set[i,]$AwayIndex,
                         data = train_set,
                         model = model,
                         scores = T)

    # record the score prediction in the data frame predictions - need
   to round the mean and rename the columns to FTHG and FTAG
    game = data.frame('Date' = test_set[i,]$Date,
                      'HomeTeam' = test_set[i,]$HomeTeam,
                      'AwayTeam' = test_set[i,]$AwayTeam,
                      'FTHG' = getmode(score$FTHG),
                      'FTAG' = getmode(score$FTAG),
                      'B365H' = test_set[i,]$B365H,
                      'B365D' = test_set[i,]$B365D,
                      'B365A' = test_set[i,]$B365A,
                      'HomeIndex' = test_set[i,]$HomeIndex,
                      'AwayIndex' = test_set[i,]$AwayIndex)

    # combinine this with predictions
    game_predictions = rbind(game_predictions, game)
  }

  # print progress of the algorithm
  print(paste('There are ', nrow(data), ' games left in the data set')
 )

  # train_set now becomes all the data and games that have happened
```

```
     before the test_set games
    train_set = rbind(train_set, data[1:i-1,])
  }

  # create table with the new predicted outcomes
  predicted_outcomes = rbind(observed_games, game_predictions)
  predicted_table = get_table(predicted_outcomes)

  # obtain complete predicted progress for the whole data set
  predicted_progress = track_points_progress(data = game_predictions,
     start_values = points_progress[,ncol(points_progress)])
  overall_progress = cbind(points_progress[,1:(ncol(points_progress)-1)
     ], predicted_progress)

  # return in list form
  predictions = list('predicted_progress' = overall_progress, 'predicted
     _table' = predicted_table)
  return(predictions)
}

### Calculate the actual observed table from 2016/17 season

get_table(E0_2016)

### Predict league table from BB model

progression_bb = predict_table('sum_to_zero.stan', data = E0_2016, model
      = 'BB', minimum_games = 10)

# Printing predicted league table from BB model
progression_bb$predicted_table

# Printing predicted progress from BB model
progression_bb$predicted_progress

### Predict league table from NB model

progression_nb = predict_table('model_nb.stan', data = E0_2016, model =
     'NB', minimum_games = 10)

# Printing predicted league table from NB model
progression_nb$predicted_table

# Printing predicted progress from NB model
progression_nb$predicted_progress

### Figure 21

```

```
# Calculate the actual observed points progress over the 380 games of
    2016/17 season
actual_progress = track_points_progress(E0_2016)

# Printing plot of points progression for each team

for (team in rownames(actual_progress)) {
  plot(actual_progress[team,], col='black', ylim=c(0,max(c(actual_
    progress[team,], progression_bb$predicted_progress[team,],
    progression_nb$predicted_progress[team,])+5)), xlab='Game Week', ylab
    ='Number of points', main=team, pch='.')
  lines(1:39, actual_progress[team,], col='black', pch='.', lwd=1.5)

  points(progression_bb$predicted_progress[team,], col='blue', pch='.')
  lines(1:39, progression_bb$predicted_progress[team,], col='blue', lwd
    =1.5)

  points(progression_nb$predicted_progress[team,], col='red', pch='.')
  lines(1:39, progression_nb$predicted_progress[team,], col='red', lwd
    =1.5)

  legend('topleft', max(c(actual_progress[team,], progression_bb$
    predicted_progress[team,], progression_nb$predicted_progress[team,],
    progression_nbb$predicted_progress[team,])), legend = c('Observed
    points', 'Baio & Blangiardo Poisson Model', 'Negative Binomial'),
    fill = c('black', 'red', 'blue'))
}
```

K League Tables: Observed and Predicted

GP = ‘Games Played’
HW = ‘Home Wins’
HD = ‘Home Draws’
HL = ‘Home Losses’
HF = ‘Home-goals For’
HA = ‘Home-goals Against’
HGD = ‘Home Goal Difference’
AW = ‘Away Wins’
AD = ‘Away Draws’
AL = ‘Away Losses’
AF = ‘Away-goals For’
AA = ‘Away-goals Against’
W = ‘Wins (Total)’
D = ‘Draws (Total)’
L = ‘Losses (Total)’ GF = ‘Goals For (Total)’
GA = ‘Goals Against (Total)’
GD = ‘Goal Difference (Total)’

K.1 Observed League Table (2016/17)

```
             Team GP HW HD HL HF HA HGD AW AD AL AF AA AGD  W  D  L GF GA  GD Points
1         Chelsea 38 17  0  2 55 17  38 13  3  3 30 16  14 30  3  5 85 33  52     93
2       Tottenham 38 17  2  0 47  9  38  9  6  4 39 17  22 26  8  4 86 26  60     86
3        Man City 38 11  7  1 37 17  20 12  2  5 43 22  21 23  9  6 80 39  41     78
4       Liverpool 38 12  5  2 45 18  27 10  5  4 33 24   9 22 10  6 78 42  36     76
5         Arsenal 38 14  3  2 39 16  23  9  3  7 38 28  10 23  6  9 77 44  33     75
6      Man United 38  8 10  1 26 12  14 10  5  4 28 17  11 18 15  5 54 29  25     69
7         Everton 38 13  4  2 42 16  26  4  6  9 20 28  -8 17 10 11 62 44  18     61
8     Southampton 38  6  6  7 17 21  -4  6  4  9 24 27  -3 12 10 16 41 48  -7     46
9     Bournemouth 38  9  4  6 35 29   6  3  6 10 20 38 -18 12 10 16 55 67 -12     46
10      West Brom 38  9  2  8 27 22   5  3  7  9 16 29 -13 12  9 17 43 51  -8     45
11       West Ham 38  7  4  8 19 31 -12  5  5  9 28 33  -5 12  9 17 47 64 -17     45
12      Leicester 38 10  4  5 31 25   6  2  4 13 17 38 -21 12  8 18 48 63 -15     44
13          Stoke 38  7  6  6 24 24   0  4  5 10 17 32 -15 11 11 16 41 56 -15     44
14 Crystal Palace 38  6  2 11 24 25  -1  6  3 10 26 38 -12 12  5 21 50 63 -13     41
15        Swansea 38  8  3  8 27 34  -7  4  2 13 18 36 -18 12  5 21 45 70 -25     41
16        Burnley 38 10  3  6 26 20   6  1  4 14 13 35 -22 11  7 20 39 55 -16     40
17        Watford 38  8  4  7 25 29  -4  3  3 13 15 39 -24 11  7 20 40 68 -28     40
18           Hull 38  8  4  7 28 35  -7  1  3 15  9 45 -36  9  7 22 37 80 -43     34
19  Middlesbrough 38  4  6  9 17 23  -6  1  7 11 10 30 -20  5 13 20 27 53 -26     28
20     Sunderland 38  3  5 11 16 34 -18  3  1 15 13 35 -22  6  6 26 29 69 -40     24
```

K.2 Predicted League Table using Baio & Blangiardo's model (2010)

	Team	GP	HW	HD	HL	HF	HA	HGD	AW	AD	AL	AF	AA	AGD	W	D	L	GF	GA	GD	Points
1	Chelsea	38	18	1	0	30	2	28	9	7	3	16	10	6	27	8	3	46	12	34	89
2	Tottenham	38	17	2	0	25	1	24	10	5	4	16	9	7	27	7	4	41	10	31	88
3	Arsenal	38	16	2	1	31	6	25	5	12	2	19	17	2	21	14	3	50	23	27	77
4	Man City	38	15	4	0	27	5	22	5	13	1	18	14	4	20	17	1	45	19	26	77
5	Man United	38	16	3	0	20	3	17	5	8	6	13	13	0	21	11	6	33	16	17	74
6	Everton	38	14	5	0	24	5	19	4	10	5	13	14	-1	18	15	5	37	19	18	69
7	Liverpool	38	13	6	0	31	6	25	2	15	2	20	20	0	15	21	2	51	26	25	66
8	West Brom	38	13	4	2	18	5	13	1	6	12	5	16	-11	14	10	14	23	21	2	52
9	Southampton	38	11	7	1	16	6	10	1	5	13	5	18	-13	12	12	14	21	24	-3	48
10	Stoke	38	11	5	3	16	8	8	1	5	13	6	23	-17	12	10	16	22	31	-9	46
11	Crystal Palace	38	8	9	2	18	11	7	0	9	10	8	22	-14	8	18	12	26	33	-7	42
12	Watford	38	8	8	3	16	11	5	1	6	12	7	24	-17	9	14	15	23	35	-12	41
13	Middlesbrough	38	7	8	4	10	7	3	2	3	14	2	16	-14	9	11	18	12	23	-11	38
14	Leicester	38	9	6	4	14	9	5	0	5	14	6	23	-17	9	11	18	20	32	-12	38
15	Bournemouth	38	5	12	2	18	16	2	0	7	12	5	22	-17	5	19	14	23	38	-15	34
16	Burnley	38	8	6	5	12	9	3	0	4	15	3	21	-18	8	10	20	15	30	-15	34
17	Hull	38	3	13	3	17	18	-1	1	1	17	1	24	-23	4	14	20	18	42	-24	26
18	Swansea	38	2	13	4	16	18	-2	1	3	15	6	29	-23	3	16	19	22	47	-25	25
19	Sunderland	38	4	9	6	13	15	-2	0	2	17	3	24	-21	4	11	23	16	39	-23	23
20	West Ham	38	3	8	8	12	17	-5	0	5	14	6	25	-19	3	13	22	18	42	-24	22

K.3 Predicted League Table using Negative Binomial model

	Team	GP	HW	HD	HL	HF	HA	HGD	AW	AD	AL	AF	AA	AGD	W	D	L	GF	GA	GD	Points
1	Chelsea	38	15	4	0	20	3	17	12	6	1	12	1	11	27	10	1	32	4	28	91
2	Tottenham	38	13	6	0	15	1	14	13	6	0	16	2	14	26	12	0	31	3	28	90
3	Man City	38	11	4	4	14	7	7	15	4	0	20	3	17	26	8	4	34	10	24	86
4	Liverpool	38	12	6	1	17	5	12	13	3	3	19	9	10	25	9	4	36	14	22	84
5	Arsenal	38	11	5	3	17	9	8	13	6	0	18	4	14	24	11	3	35	13	22	83
6	Man United	38	9	7	3	9	3	6	15	2	2	17	3	14	24	9	5	26	6	20	81
7	Everton	38	8	7	4	10	5	5	6	9	4	7	5	2	14	16	8	17	10	7	58
8	West Brom	38	9	5	5	11	7	4	5	11	3	5	3	2	14	16	8	16	10	6	58
9	Crystal Palace	38	5	7	7	6	8	-2	3	13	3	12	12	0	8	20	10	18	20	-2	44
10	Southampton	38	1	11	7	2	8	-6	7	8	4	7	4	3	8	19	11	9	12	-3	43
11	Bournemouth	38	6	9	4	11	10	1	2	7	10	4	14	-10	8	16	14	15	24	-9	40
12	West Ham	38	1	6	12	2	14	-12	8	2	9	11	12	-1	9	8	21	13	26	-13	35
13	Leicester	38	7	7	5	7	5	2	0	4	15	2	18	-16	7	11	20	9	23	-14	32
14	Middlesbrough	38	1	11	7	2	8	-6	1	14	4	1	4	-3	2	25	11	3	12	-9	31
15	Swansea	38	2	7	10	4	13	-9	4	5	10	6	12	-6	6	12	20	10	25	-15	30
16	Watford	38	3	9	7	7	12	-5	2	6	11	3	13	-10	5	15	18	10	25	-15	30
17	Burnley	38	5	9	5	5	5	0	0	5	14	0	14	-14	5	14	19	5	19	-14	29
18	Stoke	38	3	4	12	3	12	-9	2	7	10	3	11	-8	5	11	22	6	23	-17	26
19	Sunderland	38	1	4	14	2	15	-13	1	10	8	2	10	-8	2	14	22	4	25	-21	20
20	Hull	38	2	5	12	5	16	-11	0	5	14	1	15	-14	2	10	26	6	31	-25	16