Inferring the astrophysics of extreme mass ratio inspirals from gravitational wave observations

Mari Liis Pedak

Year 4 Project
School of Mathematics
University of Edinburgh
2018

Abstract

The purpose of this project is to quantify the ability of the Laser Interferometer Space Antenna (LISA), the space-based gravitational wave detector proposed by the European Space Agency, to probe the properties of the massive black hole (MBH) population of the universe and the stellar environments surrounding them. Currently, very little is known about the distribution and properties of MBHs. One of the gravitational wave sources whose properties LISA could measure are systems comprised of so-called compact objects, such as stellar-mass black holes, spiralling into massive black holes. These systems, called extreme mass ratio inspirals, provide a unique way of measuring MBH properties, and are the focus of this project. First of all, some models of the MBH population and EMRI astrophysics were explored. Then, data was generated from some of those models, and analysed using Markov Chain Monte Carlo within a Bayesian framework. We successfully reproduced results from previous work, which focused on LISA's ability to describe the parameters governing the MBH distribution. Subsequently, we extended upon these results with a more sophisticated model for EMRI astrophysics, and attempted to measure the parameters of that model as well. We discovered that it is not possible to measure the MBH distribution and the properties of stellar clusters that determine the frequency of EMRIs simultaneously, due to parameter degeneracies. However, assuming that we can measure the parameters of the MBH distribution by some other means (such as using MBH merger data), we found that we can improve upon the constraints of the parameters describing stellar clusters in current literature.

This project report is submitted in partial fulfilment of the requirements for the degree of *MMath Mathematics*.

Table of Contents

Introduction

In 1916, after Albert Einstein had formulated his theory of General Relativity, he predicted the existence of gravitational waves. According to General Relativity, spacetime can be modelled as a manifold in which objects with mass create curvature, and what we perceive to be the force of gravity actually arises from the curvature of spacetime. The theory also predicts that accelerating masses can produce 'ripples' in the fabric of spacetime, which are called gravitational waves (GWs) [1].

Gravitational waves were first observed in 2015, almost 100 years later, when the Laser Interferometer Gravitational-Wave Observatory (LIGO) observed a GW signal originating from the merger of a pair of stellar-mass black holes (BHs) [1]. Since then, the detector has observed gravitational wave signals emitted by several such stellar-mass BH binaries [2], and most recently by a pair of merging neutron stars [3].

Gravitational waves are also predicted to be emitted by much heavier black holes in the mass range of $10^4 - 10^7$ solar masses, which are called massive black holes (MBHs) and are found in the centers of galaxies. Similarly to stellar-mass BH mergers, MBH mergers are expected to produce gravitational waves. Also, GWs are predicted to be emitted by extreme mass ratio inspirals (EMRIs) - systems comprised of a stellar-mass compact object (CO), such as a stellar-mass BH or a neutron star, spiralling towards a massive black hole and eventually plunging into it. These events are predicted to be very interesting gravitational wave sources, with the potential to allow us to describe the MBH population of the universe and the stellar environments surrounding MBHs much more precisely than in current literature [2].

However, the gravitational waves emitted by MBH systems (MBH mergers and EMRIs) have lower frequencies than the mergers of stellar-mass COs, which have been observed by LIGO. Unfortunately, LIGO and other ground-based detectors are not sensitive to signals from these MBH systems, because seismic noise limits observation at the required frequencies. To solve this problem, the European Space Agency is planning to send a gravitational wave detector into space, where it would be free from seismic noise. That detector, the Laser Interferometer Space Antenna (LISA), is planned to be launched by the 2030s. It is expected to be able to observe GW signals from MBH mergers and EMRIs, giving us the ability to measure these MBH systems directly for the first time [2].

Currently, very little is known about the MBHs in the range of $10^4 - 10^7$ solar masses, since the observation methods that are currently available do not allow us to observe them directly. The parameters governing the MBH mass

function (the function that describes their distribution in the universe) are not very well constrained in current literature, and neither are the properties of the surrounding stellar clusters. Among other things, these stellar clusters govern the frequency of EMRIs, since that is where the inspiralling COs originate from. In current literature, the estimates of these parameters based on theory and numerical simulations can vary by several orders of magnitude. However, the new information obtained from the measurements of LISA should allow us to constrain the parameters of these models more precisely [2].

The objective of this project is to explore some of these models and quantify how well EMRI observations from LISA could constrain their parameters. We will extend upon previous work in Gair et al. [4], where specific models for the MBH population and the rate at which EMRIs occur were used to generate EMRI data, which was then analysed to see how well that data could be used to constrain the parameters of the MBH mass function. First, our aim will be to reproduce their results.

Gair et al. [4] assumed that the EMRI rate was known precisely, and they used a fixed EMRI rate function. However, the values of the parameters involved are not known very precisely [2] [5]. Also, according to Babak et al. [2], there are some additional factors that need to be taken into account. Hence, we will subsequently implement a more sophisticated model for the EMRI rate described in Babak et al. [2], and see whether we could also constrain the parameters of their model for the EMRI rate (which also describes the dynamics of the stellar clusters surrounding MBHs).

We will be using a Bayesian framework for the statistical analysis of the data. In order to estimate the parameters, we will be using Markov Chain Monte Carlo (MCMC), which is a method for sampling from probability distributions. Since our model will be highly complicated and we will not be able to estimate our parameters of interest analytically, MCMC provides an alternative way to do that [6].

The report is organised as follows. In Chapter 1, we are going to set up the theoretical framework, which includes a discussion of Bayesian inference and the construction of the astrophysical model we will be using. In Chapter 2, we give an overview of MCMC and how it was used in the context of this particular problem.

After that, we will move on to summarising our results. In chapter 3, we reproduce the results of Gair et al. [4] of estimating the MBH mass function. In Chapter 4, we discuss a more sophisticated model and ask if we could also measure the dynamics of stellar clusters surrounding MBHs. Finally, in the Conclusion we summarise our results and discuss further work that could be done in the future.

Chapter 1

Theoretical framework

1.1 Bayesian Inference

Bayesian inference relies on Bayes' formula, the most generic statement of which is given in equation 1.1. It relates the probabilities of any two events A and B:

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)} \tag{1.1}$$

A very similar version of the formula holds for continuous probability density functions of any two random variables.

Bayesian inference differs from frequentist statistics in that it considers the model parameters we aim to estimate as random variables (in frequentist statistics they are always regarded as constants). Bayes' formula is used in Bayesian inference to relate the probability distribution of the parameters to the probability distribution of the data. That interpretation of Bayes' formula is given in equation 1.2, with $\vec{\theta}$ referring to the parameters and $\vec{x}$ to the data [6].

$$\pi(\vec{\theta}|\vec{x}) = \frac{L(\vec{x}|\vec{\theta})p(\vec{\theta})}{f(\vec{x})} \tag{1.2}$$

Essentially, $p(\vec{\theta})$ and $f(\vec{x})$ reflect the probability density function of the model parameters and data, respectively. $p(\vec{\theta})$ is called the prior density, which is supposed to reflect the prior knowledge we have of the parameter values, e.g. physically acceptable ranges (it must be independent of the data observed). $f(\vec{x})$ is independent of the parameters, and hence merely a constant of normalization (we're looking for the probability distribution of the parameters, given the data - hence the data, once observed, can essentially be regarded as a constant). For our purposes, probability distributions do not need to be normalised (see Section 2.3), and hence we can ignore the denominator of the formula. $L(\vec{x}|\vec{\theta})$ stands for the likelihood; it reflects the probability of observing our data $\vec{x}$, given a specific set of model parameters $\vec{\theta}$ [6].

This formula is used to calculate $\pi(\vec{\theta}|\vec{x})$ - the posterior density. It represents the probability density of the parameters, conditioned on the observed data. If we want to infer something about the model parameters, e.g. their most likely values, we can use this probability distribution to infer those characteristics.

1.2 Deriving the likelihood for our model

In a similar fashion to Gair et al. [4], we are going to construct a hierarchical model for the number of EMRI events. We are going to perform a binned analysis, where the data $\vec{x} = (x_1, ..., x_K)$ are the numbers of EMRI events observed in each bin during the entire LISA mission. We will bin EMRIs based on the characteristics of the MBHs they are happening into. Therefore, first of all, we will divide the MBH population in the universe into K bins with log mass $(\ln M, \ln M + \Delta(\ln M))$ and redshift $(z, z + \Delta z)$. Redshift, denoted by z, is the most commonly used distance measure in astronomy.

Given a MBH with parameters $\vec{\lambda} = (\ln M, z)$, the number of EMRI events that occur into it in a given time period can be modelled as a Poisson process [4]. The rate of that Poisson process (the EMRI rate of the individual MBH) is predicted to depend on the characteristics of the MBH itself (encoded in $\vec{\lambda}$), but it also depends on our model assumptions, such as the characteristics of the surrounding stellar cluster [2]. The parameters $\vec{\theta}$ we are interested in estimating in this project are precisely (some of) the parameters that determine our astrophysical model. Let's denote the EMRI rate of an individual MBH $\mathcal{R}(\vec{\lambda}|\vec{\theta})$.

It is likely that there will be events from different MBHs in a given bin; however, the MBHs behave independently, and hence the number of EMRI events in the bin will follow a Poisson process with rate equal to the sum of the EMRI rates of the individual MBHs [4]. Let's denote the EMRI rate of bin $\mathcal{B}_i$ as $r_i(\vec{\theta})$ (see Section 1.2.2 for the calculation of $\mathcal{R}(\vec{\lambda}|\vec{\theta})$ and Section 1.2.4 for the precise relationship between $\mathcal{R}(\vec{\lambda}|\vec{\theta})$ and $r_i(\vec{\theta})$).

The likelihood of observing x_i events in bin $\mathcal{B}_i$ with EMRI rate $r_i(\vec{\theta})$ is therefore given by:

$$L(x_i|\vec{\theta}) = e^{-r_i(\vec{\theta})} \frac{(r_i(\vec{\theta}))^{x_i}}{x_i!} \tag{1.3}$$

The likelihood of all data is therefore [4]:

$$L(\vec{x}|\vec{\theta}) = \prod_{i=1}^{K} e^{-r_i(\vec{\theta})} \frac{(r_i(\vec{\theta}))^{x_i}}{x_i!} \tag{1.4}$$

$$\propto \prod_{i=1}^{K} e^{-r_i(\vec{\theta})} (r_i(\vec{\theta}))^{x_i} \tag{1.5}$$

where K is the number of bins. Note that for our purposes, we can ignore the $x_i!$ term in the denominator because of the likelihood principle, since it doesn't explicitly depend on $\vec{\theta}$.

In practice, we are going to work with the log likelihood. The effective log likelihood is given by:

$$l(\vec{x}|\vec{\theta}) = \sum_{i=1}^{K} \ln\left[e^{-r_i(\vec{\theta})}(r_i(\vec{\theta}))^{x_i}\right] \tag{1.6}$$

$$= \sum_{i=1}^{K}\left[-r_i(\vec{\theta}) + x_i \ln r_i(\vec{\theta})\right] \tag{1.7}$$

Now, to calculate the rates $r_i(\vec{\theta})$, we need to characterise the distribution of EMRI events that LISA will be able to detect. It depends on three factors: the distribution of MBHs in the universe that can host EMRIs, the rate at which EMRIs occur into MBHs with particular parameters (i.e. $\mathcal{R}(\vec{\lambda}|\vec{\theta})$), and the sensitivity of LISA to particular types of EMRI events [4]. We will now explore each of these factors in turn.

1.2.1 Our model for the massive black hole population

The frequency of observed EMRI events will clearly depend on the distribution of MBHs in the universe that can host EMRIs. The EMRI rate of a MBH (the rate at which EMRIs happen into it) is predicted to depend on the MBH mass [2]. The mass and redshift of a MBH also influence how well we are able to detect EMRIs happening into it [7]. Hence, in order to calculate the EMRI rate, we need to characterise the distribution of MBHs in the universe, and its dependence on mass and redshift.

In Gair et. al. [4], the differential number density of MBHs is taken to be a simple power law:

$$\mathrm{d}n = A\left(\frac{M}{M_*}\right)^{\alpha}\mathrm{d}(\ln M) \tag{1.8}$$

where n is the number density (per Mpc^3) of MBHs and M represents the MBH mass. Here, $M_* = 3\times10^6 M_\odot$, where $M_\odot$ is the solar mass. The units of A are Mpc^{-3}. The function represents the differential number density of MBHs of a given mass M.

We would like to find the differential number of black holes, in order to calculate the EMRI rate (see section 1.2.4). Hence, we need to multiply the number density by a suitable volume element. Also, we would like to include the redshift dependence of the MBH distribution in our model. There are different numbers of MBHs found at different redshifts, which is mainly due to the fact that the amount of volume encompassed by a redshift range $(z, z+\Delta z)$ depends on the redshift z.

The redshift-dependence of the volume of the universe is characterised by the comoving volume $V_c(z)$, which is the volume of the universe encompassed by redshifts less than z. The comoving volume element in solid angle $\mathrm{d}\Omega$ and redshift interval $\mathrm{d}z$ is [8]:

$$\mathrm{d}^2V_c = D_H(z)\frac{(1+z)^2[D_A(z)]^2}{E(z)}\mathrm{d}\Omega\mathrm{d}z \tag{1.9}$$

where $D_H(z)$, $D_A(z)$ and $E(z)$ are also functions of z; for their definitions, see [8]. We are going to use the package `AstroPy` to calculate d^2V_c, and multiply it by the solid angle 4π to get the differential comoving volume $\mathrm{d}V_c = (\mathrm{d}V_c/\mathrm{d}z)\mathrm{d}z$ corresponding to the entire sky [9]:

$$\mathrm{d}V_c = 4\pi D_H(z)\frac{(1+z)^2[D_A(z)]^2}{E(z)}\mathrm{d}z \tag{1.10}$$

$\mathrm{d}V_c/\mathrm{d}z$ is called the differential comoving volume. When integrated from z_1 to z_2, it gives us the volume of the universe encompassed by this redshift range. In order to derive the redshift-dependent differential number of MBHs, we will multiply the above mass function with the comoving volume element $\mathrm{d}V_c$:

$$\mathrm{d}^2N = A\left(\frac{M}{M_*}\right)^{\alpha}\frac{\mathrm{d}V_c}{\mathrm{d}z}\mathrm{d}(\ln M)\mathrm{d}z \tag{1.11}$$

Having multiplied the differential number density by the comoving volume element, we get the differential number d^2N of MBHs with log mass in the range $(\ln M, \ln M + \mathrm{d}(\ln M))$ and redshift $(z, z + \mathrm{d}z)$. The units of $\mathrm{d}V_c/\mathrm{d}z$ are Mpc^3, since z is unitless. Hence, N will also be unitless, as one would expect.

We are interested in constraining the parameters A and α. Currently, very little is known about the MBH mass function below 10^6 solar masses, since it hasn't been possible to detect these black holes directly. Based on current observational constraints on the mass function, α is anywhere in the range $(-0.3, 0.3)$, and A can range roughly between 0.0001 and 0.01 [2]. Notice how A can vary by several orders of magnitude! With LISA, we will hopefully be able to constrain these parameters further. One of the objectives of this project is to estimate how well LISA could constrain these parameters, given our model assumptions about the underlying MBH population and EMRI astrophysics. Hence, A and α will be included in $\vec{\theta}$, the vector of model parameters that we are interested in estimating.

For now, we are going to assume the MBHs have zero spin, although it is predicted that the spin of a black hole influences from how far away we are able to see the EMRIs happening into it. However, it is predicted that if a MBH has nonzero spin, we will be able to see it from farther away; hence, if we assume that the spin of MBHs is zero, we will find a conservative estimate of the EMRI rate [7]. We would like to predict a large enough rate of EMRI events to be able to constrain our parameters of interest based on them. Thus, if we predict enough events under this model, then in reality LISA will be likely to measure even more, and we will probably be able to make even better constraints.

1.2.2 Our model for the EMRI rate

In Babak et al. [2], whose model we are going to follow, the EMRI rate $\mathcal{R}(\vec{\lambda}|\vec{\theta})$ of an individual MBH consists of various different factors.

We will be using two different models. One of them will be very similar to the one used in Gair et. al. [4], and our aim will be to reproduce their results. The second one will be more sophisticated, where we will take into account the

uncertainty in the first model, and also the COs that plunge directly into the MBH without emitting a GW signal - our aim will be to make sure the growth rate of MBHs is consistent with theory. We will now explore these models in turn.

Constraining a simple power law mass function

In Gair et. al. [4], the dependence of the EMRI rate of an individual MBH on its properties is taken to be:

$$\mathcal{R}(\vec{\lambda}|\vec{\theta}) = 400 \left(\frac{M}{3\times 10^6 M_\odot}\right)^{-0.15} \text{Gyr}^{-1} \tag{1.12}$$

$$= 400 \left(\frac{M}{M_*}\right)^{-0.15} \text{Gyr}^{-1} \tag{1.13}$$

$$= 400 \times 10^{-9} \left(\frac{M}{M_*}\right)^{-0.15} \text{yr}^{-1} \tag{1.14}$$

This rate function is fixed and assumed to be known. It only depends on the mass of the MBH. We are going to use this rate function to see if we can reproduce the results in [4].

Note that we calculated the EMRI rate per year, since this will be multiplied by the observable lifetime $\tau(\vec{\lambda})$, the units of which are years (see Section 1.2.3).

A more complicated model for the EMRI rate

Gair et al. [4] assumed that the EMRI rate is known precisely. However, there is some uncertainty in current literature about the values of the parameters involved. Also, according to Babak et al. [2], there are additional factors that influence the EMRI rate, which need to be taken into account. Hence, we are now going to implement a more sophisticated EMRI rate function, which is described in Babak et al. [2]. There, the rate $\mathcal{R}(\vec{\lambda}|\vec{\theta})$ is taken to be a product of several terms. We will use two of them, the first of which is very similar to the one in [4]. Let's call it the base rate:

$$R_0(\vec{\lambda}|\vec{\theta}) = 300 \times 10^{-9} \left(\frac{M}{10^6 M_\odot}\right)^{p_1} \text{yr}^{-1} \tag{1.15}$$

This factor tells us about the dynamics of the stellar clusters surrounding the MBH. The exponent p_1 is not well-constrained in current literature - not much is known about these stellar clusters. Note that in Gair et al. [4], the factor was fixed to be -0.15. In [2], its reference value was taken to be -0.19. These are results obtained from numerical simulations. However, a theoretical analysis predicted its value to be $3/8$ [5]. Since these predictions vary significantly, we are interested in seeing if LISA could constrain this parameter better.

Also, there is an additional factor, which was not taken into account by Gair et al. [4]: the number of COs in the stellar clusters surrounding MBHs is limited.

EMRIs, however, are not the only way in which these COs are accreted by the MBH: some COs plunge directly into the MBH without producing a GW signal. N_p, the ratio of the number of these 'direct plunges' to the number of EMRIs is not very well known, and this brings additional uncertainty into our model. Namely, COs are fed into stellar clusters at a particular rate. If N_p happened to be very large, then in some cases the EMRI rate given in Equation 1.14 would predict that COs are fed into the MBH faster than they are being added to the system, which is an inconsistency - we cannot have the MBH accreting more COs than there actually are in the stellar cluster surrounding it! Hence, we need to limit the EMRI rate in some cases [2].

In order to do that, we are introducing an additional factor $\Gamma(\vec{\lambda}|\vec{\theta})$ to our EMRI rate [2]:

$$\Gamma(\vec{\lambda}|\vec{\theta}) = \min(Q, 1) \tag{1.16}$$

where:

$$Q = \frac{1.2}{1+N_p}\left(\frac{m}{10M_\odot}\right)^{-1}\left(\frac{M}{10^6 M_\odot}\right)^{0.06} \tag{1.17}$$

$\Gamma(\vec{\lambda}|\vec{\theta})$ will be multiplied by the base rate given in Equation 1.15. It is a factor that essentially 'turns on' whenever the predicted EMRI rate would otherwise become too large. As long as $Q \geq 1$, we have that $\Gamma(\vec{\lambda}|\vec{\theta}) = 1$, which does not affect the rate. However, whenever $Q < 1$ (which corresponds to the EMRI rate predicted by Equation 1.15 becoming too large), we have that $\Gamma(\vec{\lambda}|\vec{\theta}) = Q$, which limits the EMRI rate [2].

Notice that we have introduced N_p as a parameter in the formula. As mentioned, N_p represents the ratio of the number of direct plunges to the number of EMRIs. Just like p_1, N_p also carries interesting information about the dynamics of stellar clusters; however, its value is not known very precisely. Under the constraints in current literature, it can vary roughly between 1 and 100 - hence, it can vary by several orders of magnitude [2] [10]. We would like to see if we can estimate this parameter better than the constraints in current literature; we are especially interested in its order of magnitude. Hence, we will use its logarithm $n_p = \ln(N_p)$ as our parameter instead of N_p. Thus, the factor $\Gamma(\vec{\lambda}|\vec{\theta})$ becomes:

$$\Gamma(\vec{\lambda}|\vec{\theta}) = \min\left(\frac{1.2}{1+e^{n_p}}\left(\frac{m}{10M_\odot}\right)^{-1}\left(\frac{M}{10^6 M_\odot}\right)^{0.06}, 1\right) \tag{1.18}$$

Note that we introduced another parameter m, which we will take to be constant: $m = 10.0$. It represents the characteristic mass of the COs [2].

1.2.3 LISA observable lifetime

The above sections construct a model for the total number of EMRI events in the universe. In reality, however, LISA will only be able to detect a fraction of

those: the distance to the MBH (characterised by the redshift), its mass and the time at which the EMRI plunges all influence whether we will be able to detect the event [4].

LISA will have a set time window during which it will be taking observations. In order to detect an EMRI event, we need to be able to accumulate a sufficient signal-to-noise ratio (SNR) during that time window. Clearly, if an EMRI plunges into the MBH before LISA starts taking observations, it won't be detected. Also, if it plunges too soon after LISA starts taking data, we will not have accumulated enough data (i.e. a large enough SNR) for a detection. This defines a time $t_{early}(\vec{\lambda})$, which specifies the earliest time at which an EMRI can plunge and still be detected by LISA. Similarly, if an EMRI plunges too long after LISA has stopped taking observations, we will not have accumulated enough data to be able to detect it. This defines a time $t_{late}(\vec{\lambda})$, which is the latest time at which an EMRI can plunge and still be detected by LISA.

Hence, we define the observable lifetime $\tau(\vec{\lambda})$ for EMRI events with parameters $\vec{\lambda}$ (the log mass and redshift of the MBH) to be:

$$\tau(\vec{\lambda}) = t_{late}(\vec{\lambda}) - t_{early}(\vec{\lambda}) \tag{1.19}$$

The fit to calculate $\tau(\vec{\lambda})$ was made similarly to the process described in [7]; however, the fit in that paper was for an old configuration of LISA, which has now been changed. Hence, to calculate $\tau(\vec{\lambda})$ for given parameters $\vec{\lambda}$, we used the fit that Amaro-Seoane et al. [11] used in their analysis for this purpose. It corresponds to the new LISA configuration. For each parameter set $\vec{\lambda}$, the function outputs the observable lifetime in years.

The function assumes that the spin of all MBHs is zero. This assumption is probably not true in general; however, this is a conservative estimate, as was explained in Section 1.2.1.

Now, we will multiply the EMRI rate with $\tau(\vec{\lambda})$ to get the sensitivity-corrected Poisson rate for each bin. Note that having multiplied by the observable lifetime, we now have the EMRI rate 'per LISA mission' (instead of the rate per year).

1.2.4 Combining these factors to calculate the bin rate

To calculate the bin rates $r_i(\vec{\theta})$, we need to combine the factors described above. By multiplying the MBH mass function d^2N (i.e. the differential number of MBHs; see equation 1.11) by the EMRI rate $\mathcal{R}(\vec{\lambda}|\vec{\theta})$ of an individual MBH and the observable lifetime $\tau(\vec{\lambda})$, we get the sensitivity-corrected differential EMRI rate d^2R of MBHs with log mass in the range $(\ln M, \ln M + \mathrm{d}(\ln M))$ and redshift $(z, z + \mathrm{d}z)$:

$$\mathrm{d}^2R = \mathcal{R}(\vec{\lambda}|\vec{\theta})\tau(\vec{\lambda})\mathrm{d}^2N = \mathcal{R}(\vec{\lambda}|\vec{\theta})\tau(\vec{\lambda})A\left(\frac{M}{M_*}\right)^{\alpha}\frac{\mathrm{d}V_c}{\mathrm{d}z}\mathrm{d}(\ln M)\mathrm{d}z \tag{1.20}$$

To calculate the rate $r_i(\vec{\theta})$ of a particular bin $\mathcal{B}_i$, we need to integrate this differential rate over that bin:

$$r_i(\vec{\theta}) = \int_{\mathcal{B}_i} \mathrm{d}^2 R \tag{1.21}$$

$$= \int_{\mathcal{B}_i} \mathcal{R}(\vec{\lambda}|\vec{\theta})\tau(\vec{\lambda})\mathrm{d}^2 N \tag{1.22}$$

$$= \int_z \int_{\ln M} \mathcal{R}(\vec{\lambda}|\vec{\theta})\tau(\vec{\lambda}) A \left(\frac{M}{M_*}\right)^{\alpha} \frac{\mathrm{d}V_c}{\mathrm{d}z} \mathrm{d}(\ln M)\mathrm{d}z \tag{1.23}$$

Recall that $\vec{\lambda} = (\ln M, z)$; hence, $\vec{\lambda}$ is integrated out, and $r_i(\vec{\theta})$ now only depends on $\vec{\theta}$. Note that $\mathrm{d}^2 N$ also depends on the model parameters $\vec{\theta}$ via A and α (see Section 1.2.1).

In practice, this integral can be very expensive to calculate. Hence, for the purpose of this project, the EMRI rate of each bin was taken to be the integrand evaluated at the midpoint of the bin. The bin rate $r_i(\vec{\theta})$ of bin $\mathcal{B}_i$ is therefore:

$$r_i(\vec{\theta}) = \mathcal{R}(\vec{\lambda_i}|\vec{\theta})\tau(\vec{\lambda_i}) A \left(\frac{M_i}{M_*}\right)^{\alpha} \frac{\mathrm{d}V_c}{\mathrm{d}z}(z_i) \tag{1.24}$$

where $\vec{\lambda_i} = (\ln M_i, z_i)$ represents the midpoint of bin $\mathcal{B}_i$. Now, by plugging $r_i(\vec{\theta})$ into the expression for the log likelihood in Equation 1.7, we get the log likelihood $l(\vec{x}|\vec{\theta})$ for our model.

1.3 Choosing the prior distribution

In Section 1.1, we mentioned that the posterior need not be normalised, and hence we can ignore the denominator of the formula:

$$\pi(\vec{\theta}|\vec{x}) = \frac{L(\vec{x}|\vec{\theta})p(\vec{\theta})}{f(\vec{x})} \tag{1.25}$$

$$\propto L(\vec{x}|\vec{\theta})p(\vec{\theta}) \tag{1.26}$$

In practice, we are going to work with the log posterior:

$$\ln \pi(\vec{\theta}|\vec{x}) = l(\vec{x}|\vec{\theta}) + \ln p(\vec{\theta}) \tag{1.27}$$

Now, having derived our log likelihood $l(\vec{x}|\vec{\theta})$ in the previous section, we can plug it into the above equation to calculate the posterior distribution. However, we still need to choose a prior distribution $p(\vec{\theta})$.

We would like to choose the least informative prior possible, so our inferences would mostly be based on the data (via the likelihood). Hence, for most of our parameters, we are going to assume a uniform prior, where the probability is uniform in a specified range, and zero elsewhere; see Equation 1.28 (where a_j and b_j are the lower and upper bounds of the region for θ_j, for $j = 1, 2, ..., p$). Since

MCMC doesn't require the function to be normalized, we can simply assign any constant value c to the specified region; let us choose the value $c = 1$.

$$p(\vec{\theta}) = \begin{cases} 1, & a_1 < \theta_1 < b_1, a_2 < \theta_2 < b_2, ..., a_p < \theta_p < b_p \\ 0 & otherwise \end{cases} \tag{1.28}$$

The specified range will be one that contains all the physically acceptable values for these parameters, based on theory. The reason for the choice is that the uniform prior prioritises all values inside the given range equally, and hence should not influence our results (apart from restricting it to the range specified, which is reasonable, since it will not be physically possible for the parameters to take any values outside the range); this way, our inference should mostly be based on the likelihood.

To get the log prior density, we simply take a logarithm of that, defining the logarithm of 0 as $-\infty$ (that is what the `emcee` package accepts as input as well - see Section 2.3.2); see equation 1.29.

$$\ln p(\vec{\theta}) = \begin{cases} 0, & a_1 < \theta_1 < b_1, a_2 < \theta_2 < b_2, ..., a_p < \theta_p < b_p \\ -\infty & otherwise \end{cases} \tag{1.29}$$

There will be one exception to this. For A, the uncertainty in it ranges two orders of magnitude; therefore, we do not want to give more weight to values with a higher order of magnitude (which would happen if we chose a prior that is uniform in A). Hence, it seems reasonable to choose a prior which is uniform in $\ln A$, to incorporate scale invariance.

For a scale parameter A_{scale}, $\frac{1}{A_{scale}}$ is its Jeffreys prior, which has the property of scale invariance: every order of magnitude of A_{scale} will have an equal weight [6]. Here, we define a scale parameter A_{scale} to be one such that $L(\vec{x}|\vec{\theta}) \propto A_{scale}^d$ for some constant d.

Here, A is not exactly a scale parameter: since $r_i(\vec{\theta}) \propto A$, we have that:

$$L(\vec{x}|\vec{\theta}) \propto e^{-A} A^{x_1+...+x_K} \tag{1.30}$$

Calculating the Jeffreys prior for A in our case is more complicated. However, note that $\frac{1}{A}$ is, in fact, uniform in $\ln A$, which is desirable in our case, as discussed above. Hence, we are going to choose $\frac{1}{A}$ as our prior, to incorporate scale invariance.

Therefore, when we are estimating the parameter A (amongst others), we will choose the prior to be:

$$p(\vec{\theta}) = \begin{cases} \frac{1}{A}, & a_1 < A < b_1, a_2 < \theta_2 < b_2, ..., a_p < \theta_p < b_p \\ 0 & otherwise \end{cases} \tag{1.31}$$

where $\theta_2, ..., \theta_p$ represent the rest of the parameters being estimated.

Thus, in this case, the log prior is:

$$\ln p(\vec{\theta}) = \begin{cases} -\ln(A), & a_1 < A < b_1, a_2 < \theta_2 < b_2, ..., a_p < \theta_p < b_p \\ -\infty & otherwise \end{cases} \tag{1.32}$$

Note that we also incorporated scale invariance in N_p by sampling in $\ln N_p$. Similarly, we could have incorporated scale invariance in A by sampling in $\ln A$, instead of assigning $\frac{1}{A}$ as the prior. However, we chose to initially sample in A, in order to be consistent with Gair et al. [4]. Later, we will try sampling in $\ln A$ and see that these methods turn out to be equivalent (see Section 3.2.2). We are also going to compare these two methods to determine if either choice of parameterisation yields better results (e.g. quicker convergence). When sampling in $\ln A$, we will use a uniform prior, just like for $n_p = \ln N_p$.

In terms of parameter ranges, we will confine A to a range of (0, 0.05), and α to (-1, 1). These intervals include the physically acceptable ranges for these parameters, namely (0.0001, 0.01) and (-0.3, 0.3), respectively (see Section 1.2.1). We made the prior wider to allow for uncertainty in our estimates.

Similarly, we will confine p_1 to (-1, 1) and $n_p = \ln(N_p)$ to (0, 5), which confines N_p to roughly (1, 150). These include the physically acceptable values for these parameters (see Section 1.2.2).

Chapter 2

Markov Chain Monte Carlo

2.1 Motivation

In the case of a simple likelihood function and prior density, we could infer characteristics of the posterior analytically, e.g. by integrating it to find the mean of the parameters, or differentiating it to find its maxima. For example, for a single parameter θ, we would simply integrate its product with its posterior density over all its possible values to find the mean:

$$\mu = \int_{-\infty}^{\infty} \theta \pi(\theta|\vec{x}) d\theta \tag{2.1}$$

However, if the posterior is a highly complex and possibly multi-dimensional function, integrating or finding its maxima analytically may be intractable (as will be the case here).

Hence, we need other methods for doing so. Markov Chain Monte Carlo (MCMC) can be very useful in this context. In essence, MCMC is an algorithm that generates a sample from a given probability distribution. Using MCMC, we can generate a sample from the posterior, from which many characteristics of the distribution can be inferred [6].

For example, if we have a sample of n data points $\theta^{(1)}, ..., \theta^{(n)}$ from the distribution, we can infer its mean approximately using formula 2.2 (this is an example of Monte Carlo approximation):

$$\mu \approx \frac{1}{n} \sum_{k=1}^{n} \theta^{(k)} \tag{2.2}$$

The sample can be used for many purposes: for example, to infer the mean, median, variance, covariance, etc. of the parameters [6]. However, generating random samples is generally a very difficult procedure. Also, if integrating the density is analytically intractable, then so is normalizing it, which may present us with another difficulty. MCMC, however, finds a way around these problems in a rather elegant manner.

2.2 Using Markov Chains for sampling

Markov Chain Monte Carlo (MCMC) is a way of generating a sample from a probability distribution. Instead of generating each element in the sample independently, the desired sample is generated using a kind of random walk in the parameter space, which is constructed in such a way that the number of times each region is visited is proportional to the posterior density in that region [6].

That random walk is constructed using Markov Chains. A Markov Chain is a stochastic sequence of values, where each value depends only on the previous one. The new value, $\vec{\theta}_{n+1}$, depends on the previous value $\vec{\theta}_n$ according to the *transition kernel* $p(\vec{\theta}_{n+1}|\vec{\theta}_n)$, which is assumed to be independent of n. It is important that the transition kernel only depends on the current state $\vec{\theta}_n$, and is independent of all other values $\vec{\theta}_0, ..., \vec{\theta}_{n-1}$ the chain has taken in the past [6].

Having chosen the transition kernel $p(\vec{\theta}_{n+1}|\vec{\theta}_n)$, we start from some value $\vec{\theta}_0$ (ideally close to a region with a high posterior density), and calculate the next value $\vec{\theta}_1$ using the transition kernel. Similarly, we use $\vec{\theta}_1$ to calculate $\vec{\theta}_2$, etc., and continue in this manner for as long as necessary (until we've reached a satisfactory sample size) [6].

We would like the values 'visited' by the Markov Chain to form a sample from the desired probability distribution. Hence it would make sense for the chain to be more likely to move towards values with a high associated posterior probability. However, we would like it to sometimes move towards lower probability regions - otherwise the chain would eventually get stuck in a maximum of the posterior density (possibly merely a local one). Hence, we need a transition kernel that makes it more likely for the chain to move towards areas with a higher posterior probability, but sometimes also makes it go the other way [6].

2.3 Overview of different samplers

There are many different sampling algorithms. The most commonly used ones are the Gibbs sampler and the Metropolis-Hastings family of samplers. The Gibbs sampler uses the marginal densities of the posterior in the transition kernel, so it is only useful if the marginal distributions of each parameter are of a standard form and we know how to sample from them directly. This is not possible in general, and will not be possible with our posterior distribution. Hence, we will not consider the Gibbs sampler in this project.

The Metropolis-Hastings samplers, however, do not make any such assumptions about the posterior. They rely on a common method, which we will describe in the following section.

2.3.1 The Metropolis-Hastings Samplers

The Metropolis-Hastings algorithm is, in fact, a family of samplers. They all use a common method, which involves drawing a random value $\vec{Y}$ from a probability distribution called the *proposal distribution* $q(\vec{Y}|\vec{\theta}_n)$ on each timestep, where $\vec{\theta}_n$ is the current state of the chain. Subsequently, the proposed value is evaluated

according to some criteria, and either accepted or rejected. If it is accepted, we set $\vec{\theta}_{n+1} = \vec{Y}$; if not, we set $\vec{\theta}_{n+1} = \vec{\theta}_n$ [6]. The Metropolis sampler is perhaps the simplest example of those, which we will now describe briefly. It works as follows [6]:

- Suppose you are at the position $\vec{\theta}_n$ in the parameter space. Now, a proposed move $\vec{Y}$ is generated, using a proposal distribution $q(\vec{Y}|\vec{\theta}_n)$ (e.g. a multidimensional normal distribution centered at $\vec{\theta}_n$). The only requirement on the proposal distribution is that it needs to be symmetric, i.e. $q(\vec{Y}|\vec{\theta}_n) = q(\vec{\theta}_n|\vec{Y})$.
- Now, we evaluate the proposed move $\vec{Y}$:
  - If $\pi(\vec{Y}|\vec{x}) > \pi(\vec{\theta}_n|\vec{x})$ (i.e. the proposed value has a higher associated posterior probability than our current state), we always accept the move;
  - If $\pi(\vec{Y}|\vec{x}) < \pi(\vec{\theta}_n|\vec{x})$ (i.e. the proposed value has a lower associated posterior probability), we accept the move with a certain probability, which is given by:

$$r = \frac{\pi(\vec{Y}|\vec{x})}{\pi(\vec{\theta}_n|\vec{x})} = \frac{\frac{L(\vec{x}|\vec{Y})p(\vec{Y})}{f(\vec{x})}}{\frac{L(\vec{x}|\vec{\theta}_n)p(\vec{\theta}_n)}{f(\vec{x})}} = \frac{L(\vec{x}|\vec{Y})p(\vec{Y})}{L(\vec{x}|\vec{\theta}_n)p(\vec{\theta}_n)} \tag{2.3}$$

  See Figure 2.1 for an illustration of this step.
- If $\vec{Y}$ was accepted, we set $\vec{\theta}_{n+1} = \vec{Y}$; if not, we set $\vec{\theta}_{n+1} = \vec{\theta}_n$;
- Now, repeat the above procedure with $\vec{\theta}_{n+1}$.

Another way of summarising the *acceptance probability* α of a proposed move $\vec{Y}$ is [6]:

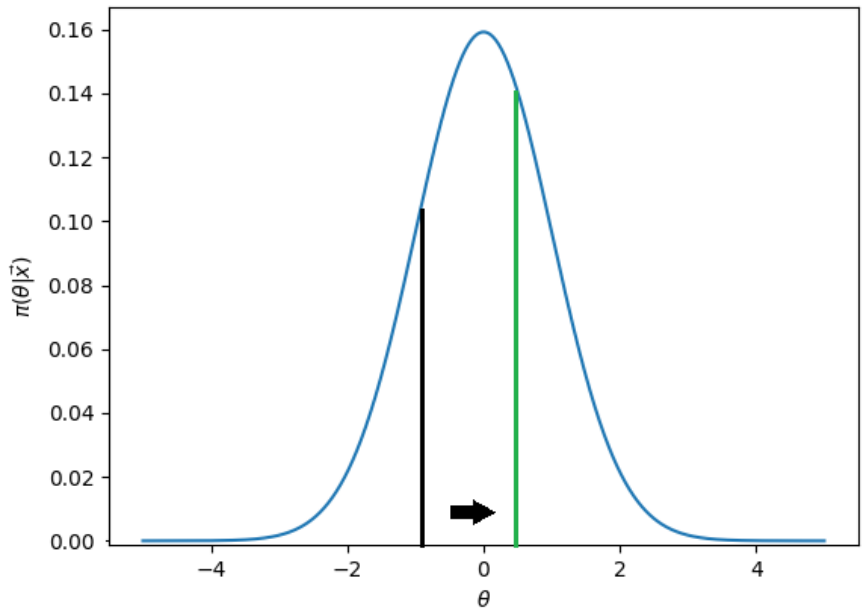
$$\alpha(\vec{\theta}_n, \vec{Y}) = \min(1, r) \tag{2.4}$$

$$= \min\left(1, \frac{L(\vec{x}|\vec{Y})p(\vec{Y})}{L(\vec{x}|\vec{\theta}_n)p(\vec{\theta}_n)}\right) \tag{2.5}$$

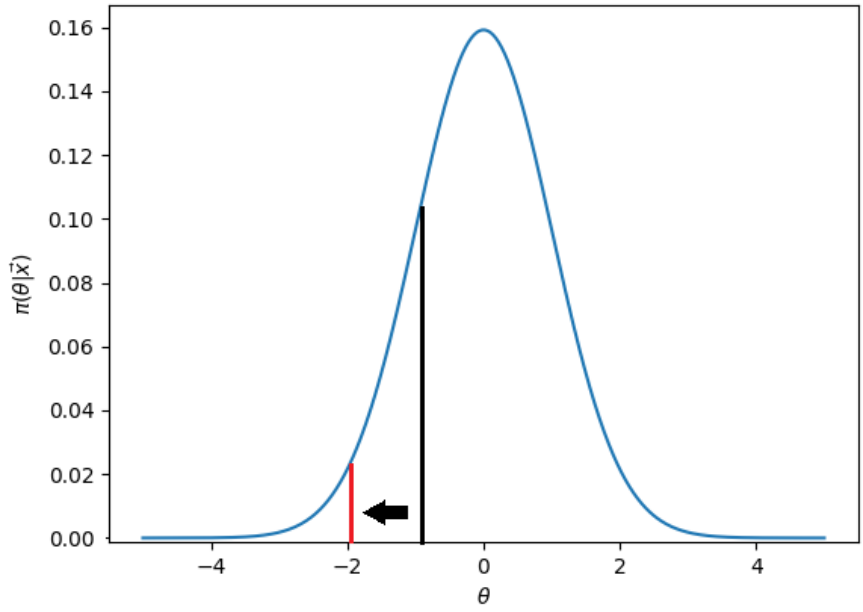
Now, in the case of generic Metropolis-Hastings samplers, the proposal distribution need not be symmetric. In that case, the acceptance probability is [6]:

$$\alpha(\vec{\theta}_n, \vec{Y}) = \min\left(1, \frac{L(\vec{x}|\vec{Y})p(\vec{Y})}{L(\vec{x}|\vec{\theta}_n)p(\vec{\theta}_n)} \frac{q(\vec{\theta}_n|\vec{Y})}{q(\vec{Y}|\vec{\theta}_n)}\right) \tag{2.6}$$

Note how in the case of symmetric proposal distributions (i.e. $q(\vec{Y}|\vec{\theta}_n) = q(\vec{\theta}_n|\vec{Y})$), the second fraction in Equation 2.6 reduces to 1 and we recover the acceptance probability given in 2.5.



(a) A proposed move of the Metropolis algorithm (shown in green). The black line indicates the current position of the chain. This move will always be accepted (i.e. its acceptance probability is 1), since the posterior probability of the proposed move is higher than that of the current position.



(b) A proposed move of the Metropolis algorithm (shown in red). The black line indicates the current position of the chain. Since the posterior probability of the proposed move is lower than that of the current position, its acceptance probability is given by the ratio of its posterior probability to the posterior probability of the current position (here, it is therefore given by the ratio of the length of the red line to the length of the black line).

Figure 2.1: An illustration of the Metropolis algorithm for a Gaussian posterior distribution.

If we follow this approach, the number of times the chain visits each region of the parameter space will be proportional to the posterior density in that region [6]. Note how we always accept moves that take us to areas with a higher associated posterior density, but we sometimes move towards lower probability areas as well - this agrees with our intuitive discussion in the previous section.

We mentioned in Section 1.1 that the posterior does not need to be normalised for our purposes, and hence the $f(\vec{x})$ term in the denominator of Bayes' formula (see Equation 1.2) can be ignored. In the case of Metropolis-Hastings samplers, the reason why we can do that can be seen from Equation 2.6: note how the $f(\vec{x})$ term cancels out in the calculation. This is a big advantage of these methods, as the $f(\vec{x})$ term can be very expensive to compute.

2.3.2 Our choice: the Affine-Invariant Ensemble Sampler

The Metropolis-Hastings family is a good candidate for the sampler to be used in this project, since it doesn't pose any restrictions on the posterior distribution (our posterior is going to be quite complicated). They are guaranteed to work in principle; however, unless optimised properly, these algorithms can be very inefficient. For example, most proposed moves might end up being rejected, which may result in successive samples being highly correlated, and it might take an extremely long time to obtain a proper sample. Optimising these algorithms can be very difficult - in the case of an N-dimensional parameter space, it requires the hand-tuning of $\sim N^2$ parameters (for example, in the case of a normal proposal distribution, its N-by-N variance-covariance matrix needs to be hand-tuned) [12]. This is what led us to use one of the Affine Invariant Ensemble Samplers proposed by Goodman and Weare [13], which require very little problem-specific optimisation.

Instead of a single Markov chain exploring the parameter space, these algorithms employ several Markov Chains called 'walkers' that explore the parameter space simultaneously. They also 'communicate' with one another and adjust their proposal distributions according to information they receive about the whole parameter space, which results in the sampling being much more efficient. They propose different methods for proposing moves, out of which we will be using the 'stretch move', which significantly outperforms standard Metropolis-Hastings algorithms. It is somewhat similar to the Nelder-Mead Simplex optimisation algorithm, which evolves many copies of the system toward a local minimum. It also has the property that it is invariant under affine transformations, and hence can sample skewed distributions very efficiently [13].

To implement the algorithm, we used the Python package `emcee` (also called MCMC Hammer), developed by Dan Foreman-Mackey and others. It is an open source pure-Python implementation of the Affine Invariant Ensemble Sampler (specifically the 'stretch move'). For details about the implementation, the reader is referred to [12].

The 'stretch move' works as follows. The position of each walker is updated in turn. To update the position of a walker at the position X_k, a walker at position, say, X_j, is drawn randomly from the other walkers and a new position is proposed [13]:

$$Y = X_j + Z(X_k - X_j) \tag{2.7}$$

where Z is a random variable drawn from a uniform probability distribution $g(z)$:

$$g(z) \propto \begin{cases} \frac{1}{\sqrt{(z)}}, & \text{if} z \in \left[\frac{1}{a}, a\right] \\ 0 & \text{otherwise} \end{cases} \tag{2.8}$$

Here, a is a parameter that can be optimised for performance [13]. The `emcee` package uses the value $a = 2$ [12]. After a move is proposed in this manner, the probability of accepting it is similar to Equation 2.6, and is given by:

$$\alpha(\vec{\theta}_n, \vec{Y}) = \min\left(1, Z^{N-1}\frac{L(\vec{x}|\vec{Y})p(\vec{Y})}{L(\vec{x}|\vec{\theta}_n)p(\vec{\theta}_n)}\right) \tag{2.9}$$

where N is the dimension of our parameter space.

Sampling in this way, using the information about the position of other walkers, turns out to be very efficient. The interested reader is referred to [13], where these algorithms are described in more detail.

Again, note how the $f(\vec{x})$ term in Bayes' formula is not needed for this calculation.

Note that if we have, say, 100 walkers (in [12], the recommended number is 100 or more), we generate 100 samples per timestep. Also, thanks to the clever method of updating the positions of the walkers, the autocorrelation of subsequent samples will be relatively low. Hence, we do not need to run the algorithm for a very long time to generate a very large sample which is representative of the posterior distribution [12].

2.4 Convergence and the burn-in period

When a Markov Chain is initialized at a low-probability value, the first few steps it takes will not follow the posterior density - it may need time to 'find' an area of higher posterior probability. This initial period is referred to as the *burn-in period* - the period during which the values sampled by the chain are dependent on the initial position (and hence cannot be regarded as a sample from the posterior, and need to be discarded) [6].

The easiest way to assess convergence is by inspecting the trace plot(s) of the Markov Chain(s) by eye. Trace plots are graphs that show the values particular Markov Chains took at each timestep. For an example, see figure 2.2, which is the trace plot of a single Markov Chain. It is clear that during the first ~ 100 iterations the sampled values are dependent on the initial position of the chain, and hence not representative of the distribution we are sampling from. However, after those ~ 100 timesteps, the chain starts oscillating around one particular value (presumably the maximum of the posterior density function), and appears to have *converged* to the posterior distribution. These values now should be representative of the posterior distribution, and can be used as a sample. The burn-in period, however, is discarded before making any inferences [6].

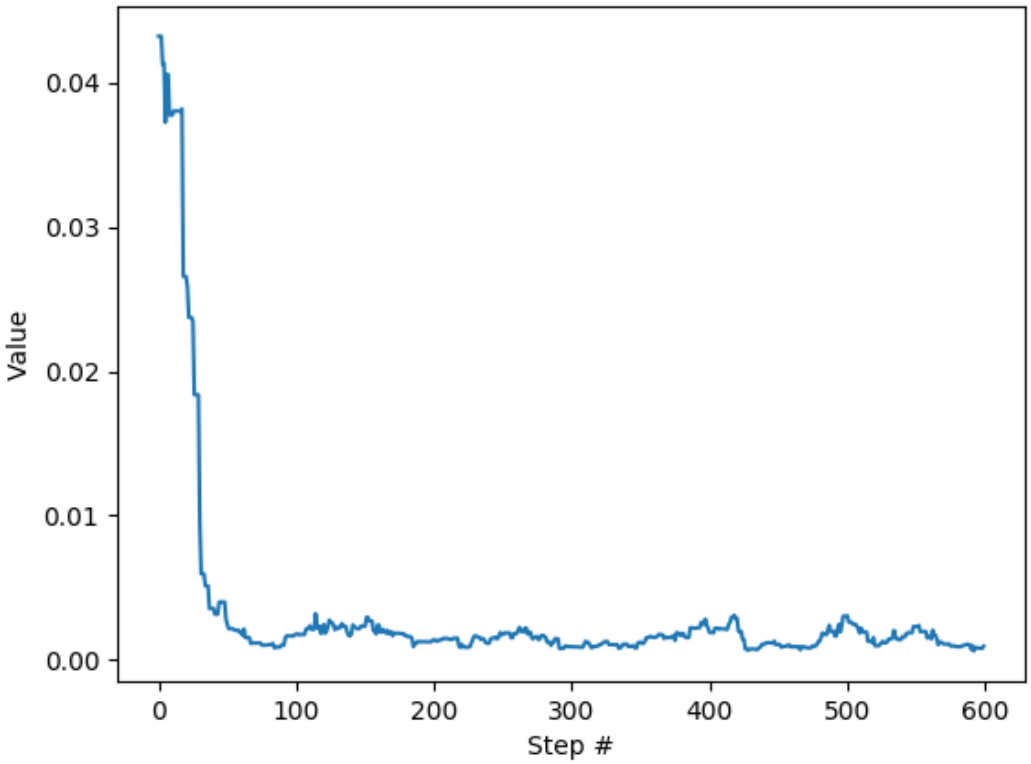


Figure 2.2: A sample trace plot of a Markov chain. The graph represents the values the Markov Chain took at each timestep. On the x axis there are timestep numbers, and on the y axis the parameter values of the chain at those timesteps. Here, the chain has a burn-in of ~ 100 timesteps, after which it converges. The true value of the parameter being estimated here is 0.002 (we can see that the chain starts oscillating very close to that number, once it has converged).

However, assessing convergence by eye is not always foolproof. Even if it appears as though the chain has converged, it is possible that it has merely got stuck in a local maximum, and has not reached the global maximum yet. In principle, there is no way to tell by eye whether we have run the chain for long enough for it to truly converge. In general, more robust methods are recommended [6].

One such method is starting several different Markov chains in different locations in the parameter space, to see if they all eventually converge to the same region [6]. Hence, when using the Affine Invariant Ensemble Sampler, we should be able to ensure the global maximum will be found by making sure the walkers initially spread out over a reasonable range in the parameter space. See Figure 2.3 for an example of a trace plot of the Affine Invariant Ensemble Sampler.

We will not discuss optimisation in great detail (mainly because the `emcee` package does almost all of it for us); however, an important thing to mention is the *acceptance fraction*. It denotes the fraction of proposed moves that are accepted. It can be optimised by choosing an appropriate *step size*, which is determined by the proposal distribution: it essentially denotes the average length of the 'jumps' proposed (which are then accepted or rejected). There is no common agreement on what an ideal value is for the acceptance fraction, but it definitely should not be too close to zero: the chain will barely move, because most moves are rejected; hence, subsequent values in the chain will be highly correlated. This mostly happens when the step size is too large. Also, the acceptance fraction should not be too close to 1, which means that nearly all moves are accepted, which usually happens when the step size is too small. In this case, subsequent samples will be very close to each other, and thus also highly correlated. In general, we want to achieve a combination of a high acceptance fraction and a low autocorrelation of the chains.

2.5 Parameter and interval estimation

After running MCMC and obtaining a sample from the posterior distribution, we will use it to estimate our parameters. There are different conventions for quoting parameter estimates; we are going to quote the median and the 90% symmetric credible interval in each dimension individually, following the convention used in Abbott et al. [1] and Abbott et al. [14]. The 90% credible interval is the region that contains the true parameter with a 90% probability [6].

In order to find these estimates, we are simply going to sort the samples resulting from a MCMC run (in each dimension), and calculate the 5%, 50% and 95% quantiles.

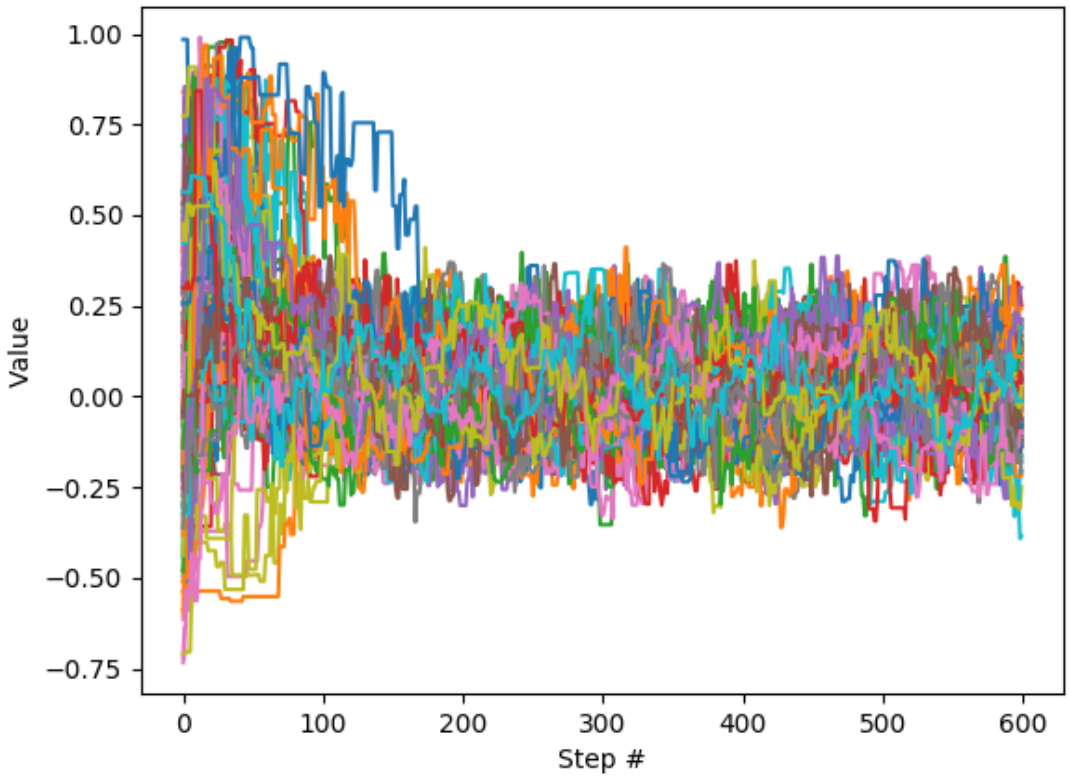


Figure 2.3: A sample trace plot of all the 'walkers' (simultaneous Markov chains) of the MCMC Hammer algorithm. The graph represents the values the walkers took at each timestep - each walker is represented by a different colour. On the x axis we have the timestep numbers, and on the y axis the parameter values. Here, the chains have a burn-in of ~ 200 timesteps, after which they converge.

Chapter 3

Constraints on a simple power law mass function

In this chapter, our objective shall be to reproduce the results from Gair et al. [4], where the aim was to estimate the parameters A and α that determine the MBH mass function (see Equation 1.11). Hence, here our parameter vector will be $\vec{\theta} = (A, \alpha)$, and our parameter space will be two-dimensional.

We are going to use the function defined in Equation 1.14 as the intrinsic EMRI rate of individual MBHs, which is the same function that was used in Gair et al. [4]. That will be inserted into Equation 1.24 to calculate the bin rates for a given parameter set $\vec{\theta}$.

3.1 Data generation and implementation of the algorithm

We implemented the Affine Invariant Ensemble Sampler in Python, using the `emcee` package. First, 'true' parameters were chosen and several sets of data were generated according to the model described above. That was done by first creating a vector which contained the EMRI rate $r_i(\vec{\theta})$ of each bin (calculated based on the true parameter values), and then using the `random.poisson` function from the `NumPy` package to draw a vector of samples from Poisson distributions with the respective rates. That vector of samples was passed on to MCMC, which was used to estimate the parameters $\vec{\theta}$.

The computer code for this particular problem can be found in Appendix B.

3.2 Results

First, we analysed some datasets generated using the reference values $A = 0.002$, $\alpha = 0.0$ that were used in Gair et al. [4]. Initially, we ran MCMC for 600 timesteps, using 100 walkers.

Here, we are going to summarise the results for one particular dataset (let us call it Dataset 1). The total number of EMRI events in this dataset was 81 (the other datasets had roughly similar numbers of EMRI events). The trace plots

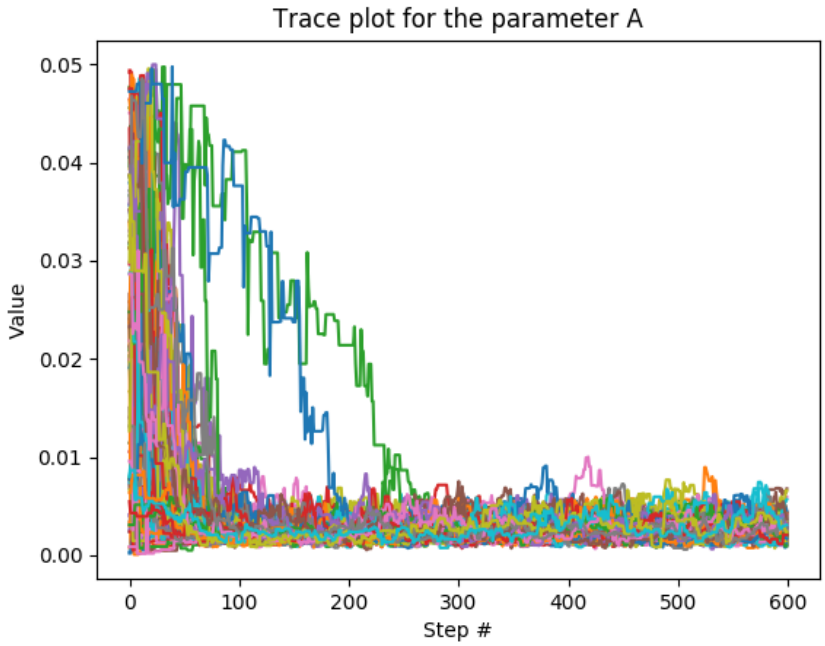


Figure 3.1: Trace plot of all the walkers in the A dimension of the posterior of a dataset with true parameters $A = 0.002$, $\alpha = 0.0$. Number of walkers: 100, number of MCMC timesteps: 600.

resulting from MCMC can be seen in Figures 3.1 and 3.2. Note how the majority of the walkers converge to one particular area after ~ 100 timesteps, but it takes about 250 timesteps for all the walkers to converge.

In order to be conservative, we chose our burn-in length to be 400 timesteps. This choice turned out to be reasonable for a large majority of our MCMC runs (see Section 3.2.1 for more details on convergence). Note that if we keep the samples from the last 200 timesteps, our sample size will be $200 \times 100 = 20,000$, since with 100 walkers we generate 100 samples per timestep!

The 'corner plot' of the posterior can be seen in Figure 3.3. It contains histograms of the sampled values in each dimension (A and α) and a 2-dimensional plot of the posterior density as a whole (which shows the correlation between parameters). The middle dashed line on the histograms indicates the median in each dimension, and the other two dashed lines indicate the 90% credible region. The blue lines indicate the true values of the parameters that we used to generate the data. The first 400 timesteps were discarded as the burn-in before plotting the sampled values (we will always discard the burn-in before using a sample for estimating parameters or generating corner plots).

After running MCMC and obtaining our sample, we used the sample (having discarded the burn-in, of course) to obtain parameter estimates, as described in Section 2.5.

For Dataset 1, the median estimate of A was 0.0024, and the corresponding 90% symmetric credible interval was (0.0013, 0.0045). The median esti-

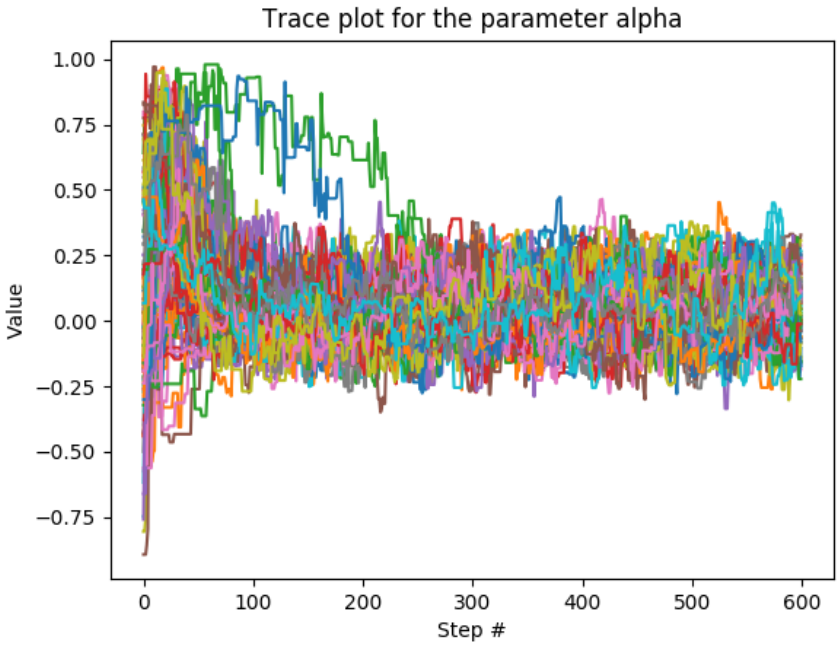


Figure 3.2: Trace plot of all the walkers in the α dimension of the posterior of a dataset with true parameters $A = 0.002$, $\alpha = 0.0$. Number of walkers: 100, number of MCMC timesteps: 600.

mate of α was 0.05, and the corresponding 90% symmetric credible interval was $(-0.13, 0.24)$. Note that the true values of A and α are contained within the respective credible intervals, which is an indicator of success.

As for the other datasets, the 90% credible intervals contain the true parameter values in 13 out of 15 cases. This is consistent with our choice of significance level. Also, the convergence behaviour and the shape of the posterior was fairly similar in all cases.

The mean acceptance fraction was 0.65, which also indicates reasonably good performance.

3.2.1 Assessing convergence

In order to assess whether the walkers had truly converged, we ran MCMC on the same datasets, but this time with 200 walkers for 1500 timesteps, to see if that changes our results. First, we briefly summarise the results for Dataset 1, the same dataset as in Section 3.2.

The resulting trace plots can be seen in Figures 3.4 and 3.5. Note that the trace plots are very similar to the trace plots considered in the previous section (Figures 3.1 and 3.2) - after the walkers appeared to converge to one area (after 250 timesteps), their behaviour did not change during the next 1250 timesteps. Thus, we are inclined to deduce that they indeed converge after the first 250 timesteps.

Also, note how the walkers initially spread out over the whole parameter space,

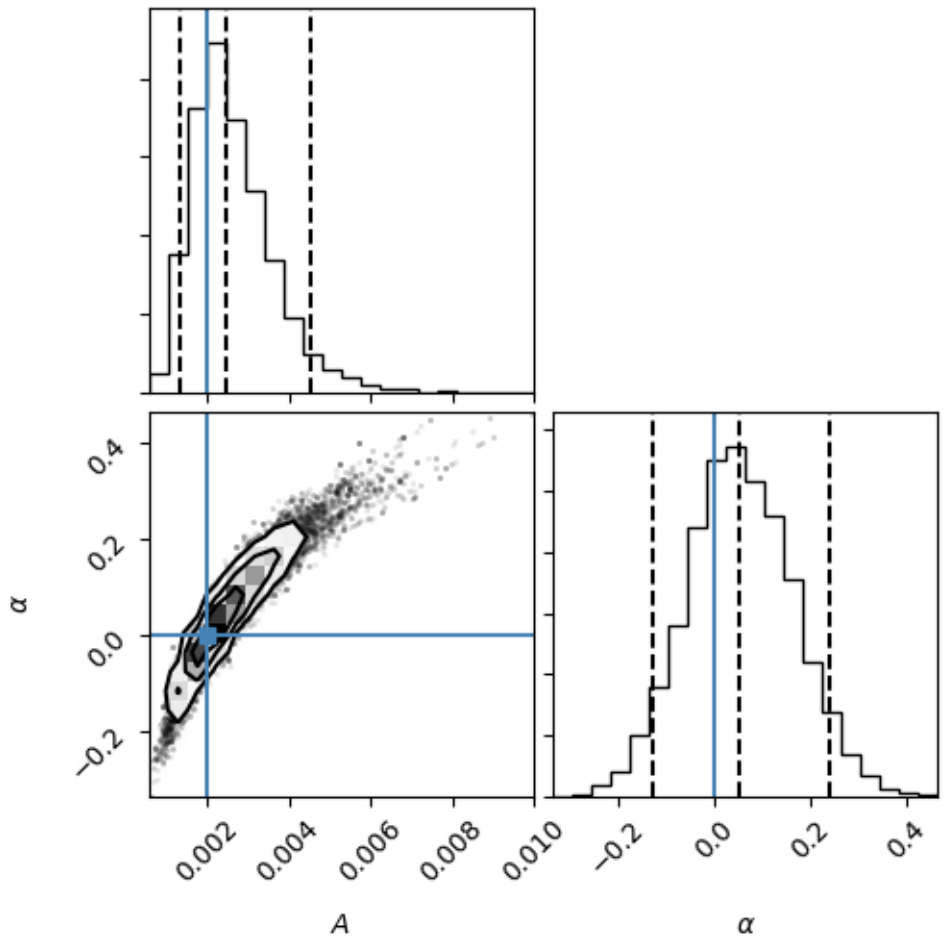


Figure 3.3: The corner plot of the MCMC sample from the posterior of a dataset with true parameters $A = 0.002$, $\alpha = 0.0$ (the blue lines indicate the true values of the parameters). The histograms represent the sampled values in each dimension (A and α), and the bottom left plot is of the whole 2-dimensional posterior. The middle dashed line indicates the median in each dimension, and the other two dashed lines indicate the 90% credible region. Number of walkers: 100, number of MCMC timesteps: 600, burn-in length: 400.

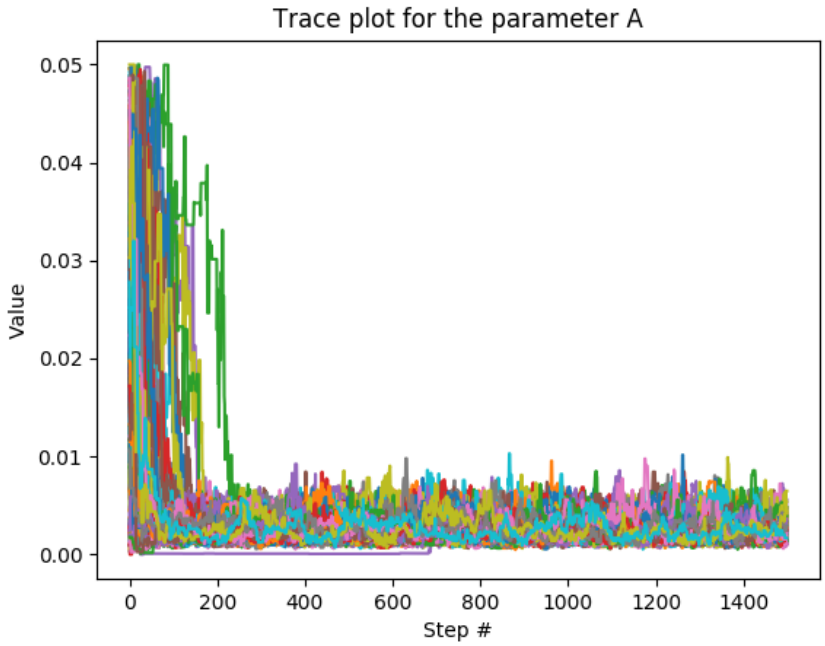


Figure 3.4: Trace plot of all the walkers in the A dimension of the posterior of a dataset with true parameters $A = 0.002$, $\alpha = 0.0$. Number of walkers: 200, number of MCMC timesteps: 1500.

before converging to one particular area. This makes it likely that the area they finally converge to is indeed the global maximum, and not merely a local one (as we discussed in Section 2.4).

The corner plot of the posterior can be seen in Figure 3.6 (this time we discarded the first 1300 steps as the burn-in, so that the size of the final sample would be the same). Note how its shape is also mostly unchanged.

This time, the mean acceptance fraction was 0.68 - roughly the same as before.

We also compared the parameter estimates from shorter and longer runs (600 timesteps with 100 walkers, and 1500 timesteps with 200 walkers), to see if there was a meaningful difference. A few examples can be seen in Tables 3.1 and 3.2. All the parameter estimates were roughly equal, differing only marginally.

These comparisons (visual comparisons between plots and comparison of parameter estimates) gave similar results for all other 15 datasets that we analysed this way. Hence, we concluded that for this problem, running MCMC for 600 timesteps using 100 walkers should be enough to obtain meaningful results.

3.2.2 Improving the efficiency of sampling

As discussed in Section 1.3, we are going to try sampling in $\ln A$, to see if this alternative way of specifying our model is consistent with our previous analysis, and to determine if either choice of parameterisation yields better results (e.g. quicker convergence). Hence, now our parameter vector is $\vec{\theta} = (\ln A, \alpha)$ (of course,

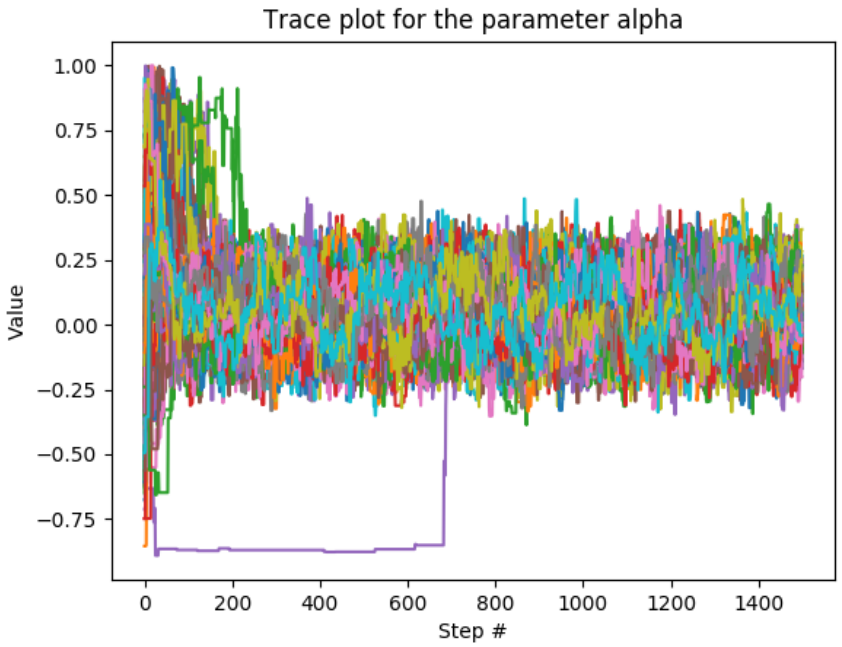


Figure 3.5: Trace plot of all the walkers in the α dimension of the posterior of a dataset with true parameters $A = 0.002$, $\alpha = 0.0$. Number of walkers: 200, number of MCMC timesteps: 1500.

	600 timesteps		1500 timesteps	
Dataset	Median	90% Credible Interval	Median	90% Credible Interval
1	0.0024	(0.0013, 0.0045)	0.0024	(0.0013, 0.0044)
2	0.0025	(0.0014, 0.0047)	0.0026	(0.0013, 0.0049)
3	0.0014	(0.0008, 0.0026)	0.0014	(0.0007, 0.0026)
4	0.0016	(0.0008, 0.0032)	0.0016	(0.0008, 0.0030)

Table 3.1: These are the estimates of A from 4 different MCMC runs (with different datasets; $A = 0.002$, $\alpha = 0.0$). Dataset 1 in the table is for the sample dataset discussed in Section 3.2.

	600 timesteps		1500 timesteps	
Dataset	Median	90% Credible Interval	Median	90% Credible Interval
1	0.05	(-0.13, 0.24)	0.05	(-0.12, 0.23)
2	0.11	(-0.07, 0.29)	0.11	(-0.08, 0.30)
3	-0.12	(-0.28, 0.06)	-0.11	(-0.29, 0.06)
4	-0.02	(-0.20, 0.19)	-0.02	(-0.22, 0.17)

Table 3.2: These are the estimates of α from 4 different MCMC runs (with different datasets; $A = 0.002$, $\alpha = 0.0$). Dataset 1 in the table is for the sample dataset discussed in Section 3.2.

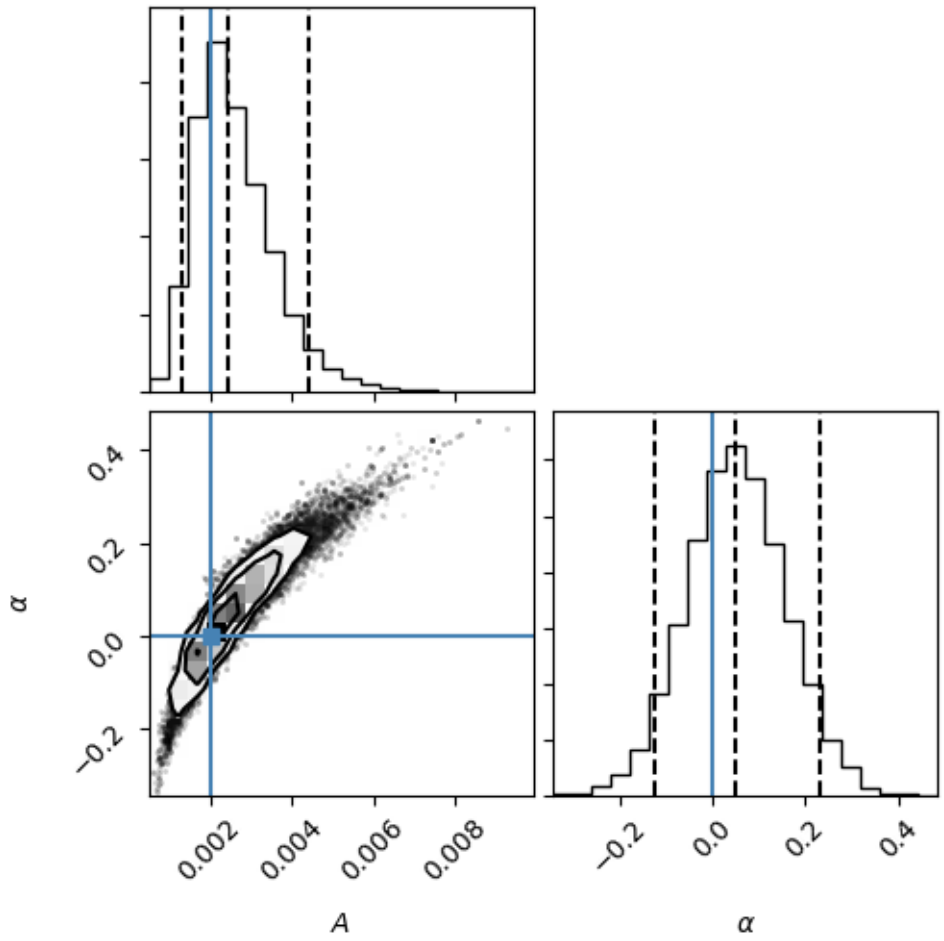


Figure 3.6: The corner plot of the MCMC sample from the posterior of a dataset with true parameters $A = 0.002$, $\alpha = 0.0$ (the blue lines indicate the true values of the parameters). Number of walkers: 200, number of MCMC timesteps: 1500, burn-in length: 1300.

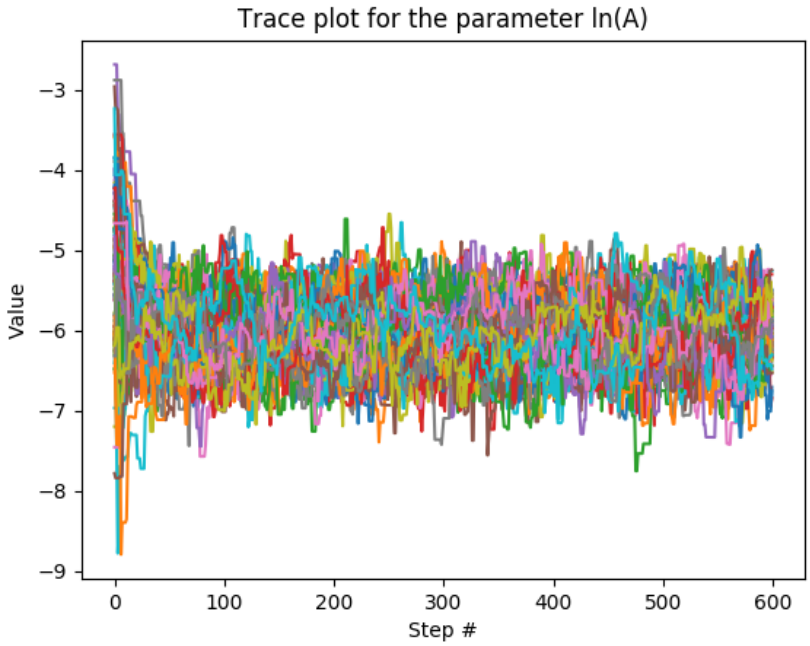


Figure 3.7: Trace plot of all the walkers in the $\ln A$ dimension of the posterior of a dataset with true parameters $A = 0.002$, $\alpha = 0.0$. Sampling in $\ln A$ and α. Number of walkers: 100, number of MCMC timesteps: 600.

we also had to modify the rate function appropriately, replacing A with $e^{\ln A}$).

We used this method of sampling on the same datasets as in the previous section. The results for Dataset 1 are summarised here.

The trace plots from analysing Dataset 1 can be seen in Figures 3.7 and 3.8. Note how in this case the burn-in is almost instantaneous - it appears to take less than 50 timesteps for all the walkers to converge. Comparing this to the burn-in of 250 timesteps in Figures 3.1 and 3.2 (where we were sampling in A) seems to indicate that sampling in $\ln A$ is much more efficient.

The corner plot of the posterior distribution can be seen in Figure 3.9. Note how now the posterior looks approximately Gaussian, and the posterior appears to be smoother. This is also an indicator that our algorithm performs better when sampling in $\ln A$.

The median estimate of A was 0.0025, and the corresponding 90% credible interval was $(0.0013, 0.0044)$. For α, the median estimate was 0.05, and the corresponding 90% credible interval was $(-0.12, 0.23)$. Note how these estimates agree perfectly with the estimates obtained from the run with 1500 timesteps and 200 walkers (when we were sampling in A). Hence, our new results from the sampling in $\ln A$ are consistent with the ones obtained previously.

Parameter estimates from several other datasets were compared, and the results were very similar to those seen in Tables 3.1 and 3.2, showing that the parameter estimates produced by these two alternative sampling methods are consistent with each other.

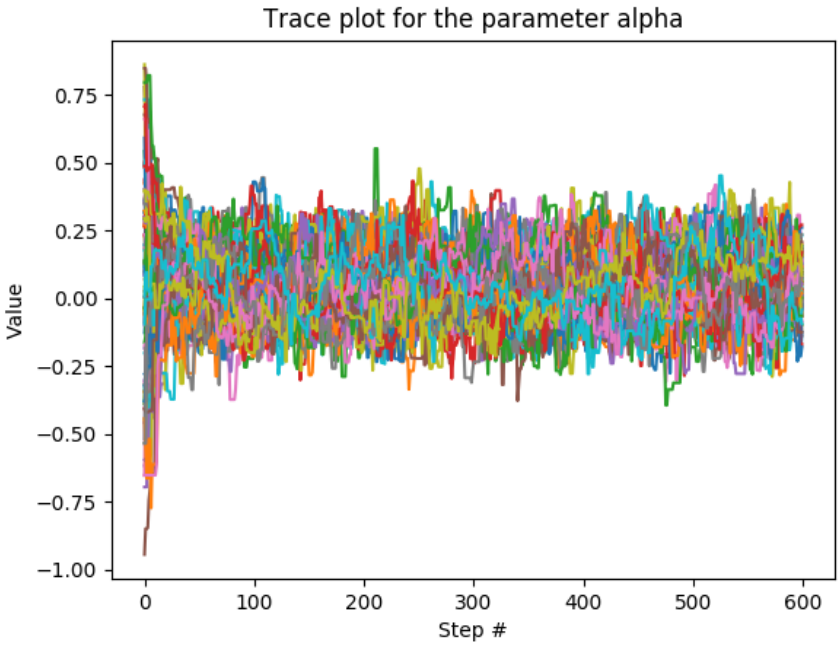


Figure 3.8: Trace plot of all the walkers in the α dimension of the posterior of a dataset with true parameters $A = 0.002$, $\alpha = 0.0$. Sampling in $\ln A$ and α. Number of walkers: 100, number of MCMC timesteps: 600.

3.3 Analysis of results

In Gair et al. [4] the authors find that the posteriors in $\ln A$ and α are very well fit by Gaussians. The shape of our posterior when sampling in $\ln A$ appears to be consistent with that description.

We cannot compare our parameter estimates with theirs directly, since the comoving volume function V_c and observable lifetime function (see Section 1.2) they use are slightly different from ours (the configuration of LISA has changed since they obtained their results [2]). Also, they use a different measure for estimating the uncertainty in the parameter estimates (they use the standard deviation of the best fit Gaussian). However, the shape of our posterior and the correlation between parameters appear to be consistent with Figures 2 and 3 in Gair et al. [4], which we shall count as a success.

The 90% credible interval of A that we obtained is $(0.0013, 0.0044)$. Note that this successfully determines the order of magnitude of A. As we mentioned in Section 1.2.1, the uncertainty in A in current literature ranges several orders of magnitude. Here, we have shown that if $A = 0.002$, we would be able to determine the order of magnitude of A quite precisely, which would be a big improvement over the current knowledge of the value of the parameter.

For α, the 90% credible interval was $(-0.12, 0.23)$. Again, note how the interval is significantly narrower than the uncertainty in current literature, which is $(-0.3, 0.3)$. This shows that if our assumptions hold, applying our method to the EMRI data from LISA would be able to produce meaningful results and

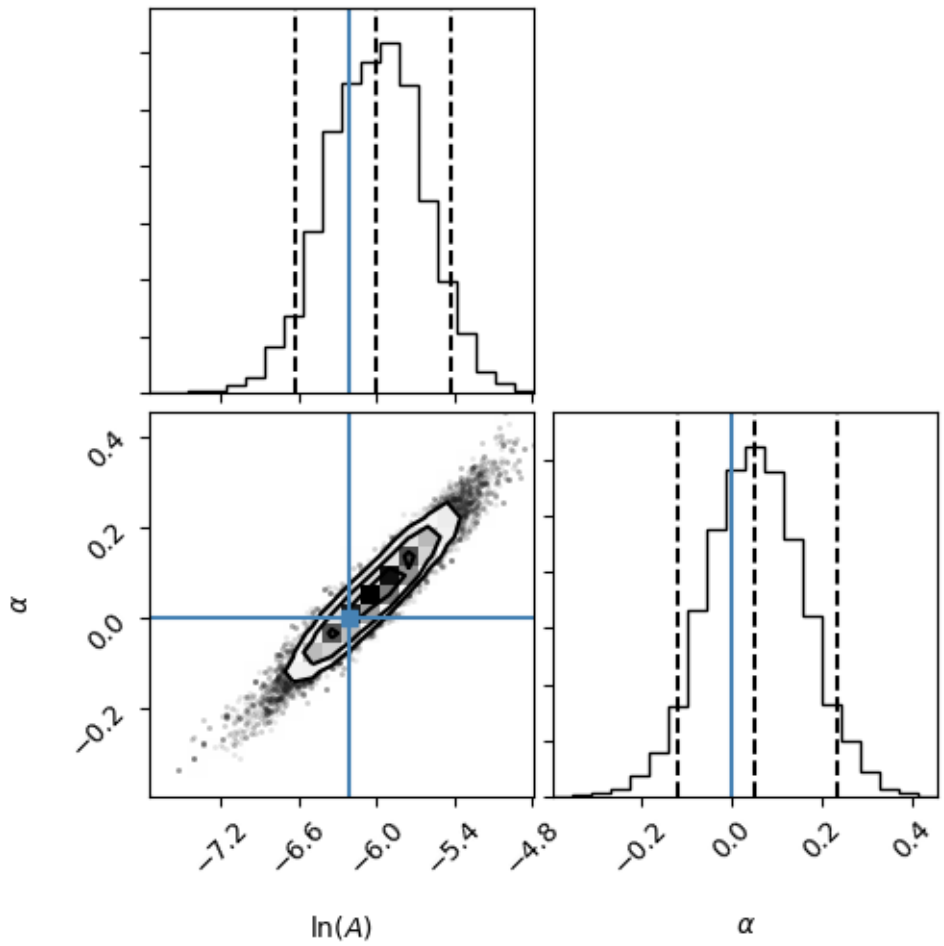


Figure 3.9: The corner plot of the MCMC sample from the posterior of a dataset with true parameters $A = 0.002$, $\alpha = 0.0$. Sampling in $\ln A$ and α. Number of walkers: 100, number of MCMC timesteps: 600, burn-in length: 400.

improve upon the parameter constraints in current literature.

Some more plotted results produced using different true parameters can be found in Appendix A.

Chapter 4

Constraints on a more realistic population model

In the previous chapter, we assumed that the EMRI rate given in Equation 1.14 is known precisely. This, however, is not true: the parameter -0.15 is actually not very well constrained in literature (see Section 1.2.2). Truth is, we do not know EMRI astrophysics perfectly, and we should take into account the uncertainty in current models. Hence, following Babak et al. [2], we will use Equation 1.15 instead, replacing the exponent (which was previously assumed to be known) with a parameter p_1.

Also, as mentioned in Section 1.2.2, the EMRI rate in Equation 1.14 does not take into account the fact that the supply of COs to the stellar clusters surrounding MBHs is limited. Hence, in this chapter, we introduce an additional factor $\Gamma(\vec{\lambda}|\vec{\theta})$ (see Equation 1.16), which limits the EMRI rate accordingly. Along with that, we introduce another new variable N_p - the ratio of the number of COs that plunge directly into the MBH to the number of COs that end up as EMRIs. Also, as noted before, we are going to sample in $n_p = \ln N_p$ (see Section 1.2.2).

One thing we immediately noticed was that, having introduced the $\Gamma(\vec{\lambda}|\vec{\theta})$ factor, the number of overall EMRI events was reduced significantly. Using the reference values from the previous chapter ($A = 0.002$ and $\alpha = 0.0$), and taking $p_1 = -0.19$ and $N_p = 10.0$ (the reference values from Babak et al. [2]), we found that the model generated only a couple of EMRI events. Only a couple of data points are clearly not enough to reliably estimate our parameters. Hence, we had to conclude that if these indeed are the true parameters, we will not be able to estimate our model parameters. However, this does not necessarily mean that LISA will only observe a few events in reality - as was mentioned in Chapter 1, we made some conservative assumptions that most likely are not true (such as the spin of black holes being zero). When allowing for nonzero spin, Babak et al. [2] predict 100-300 detected EMRI events based on this model.

For the purpose of this analysis, however, we will change the values of some of the aforementioned parameters, in order to generate more EMRI events and conduct a meaningful analysis. In particular, we will use $A = 0.01$ and $\alpha = -0.3$, keeping p_1 and N_p the same as above. This choice of parameters will keep the overall number of EMRI events roughly the same as in Chapter 3. Note that the parameter choices we made are still consistent with theory.

Since we found in Chapter 3 that sampling in $\ln A$ is more efficient than sampling in A, we will sample in $\ln A$ in this chapter.

We also confirmed that we are still able to measure $\ln A$ and α when using the new model (keeping p_1 and N_p fixed); however, the results do not differ significantly from the previous chapter, and we will not display them here.

The code that was used for the problems in this section is very similar to the one described in Section 3.1, with some slight modifications to the rate functions, parameters, etc.

We will not discuss convergence in this chapter; however, similar checks as in Chapter 3 were performed to ensure convergence had been achieved.

4.1 Estimating the MBH mass function and properties of stellar clusters simultaneously

In this chapter, we are interested in describing the properties of stellar clusters by estimating the parameters p_1 and N_p, which govern the EMRI rate. First, we are going to see if it will be possible to estimate those parameters and the parameters governing the MBH mass function simultaneously.

Estimating A, α and p_1 simultaneously

First, we shall introduce p_1 as another variable (and keep N_p fixed). This makes our parameter space 3-dimensional: $\vec{\theta} = (\ln A, \alpha, p_1)$. We generated some datasets using the true parameter values discussed at the beginning of this chapter.

Similarly to Chapter 3, we ran MCMC for 600 timesteps, using 100 walkers. The resulting trace plots for one particular dataset can be seen in Figures 4.1, 4.2 and 4.3. Note how this time, the uncertainty in the parameters is extremely high, especially in the case of α and p_1; the likelihood appears to have support over a range that is wider than the prior. The uncertainty here is certainly larger than the constraints in current theory.

The posterior distribution can be seen in Figure 4.4. Here, there are three 2-dimensional plots, which indicate the correlation between all three pairs of parameters. Note how the parameters α and p_1 appear to be almost perfectly (negatively) correlated. When looking at the formulae in Equations 1.11 (the MBH mass function) and 1.15 (the base rate for EMRIs), the reason for this degeneracy becomes clear: in the mass function, M/M_* is raised to the power α, and in the EMRI rate formula, $M/(10^6 M_\odot)$ is raised to the power p_1 (remember that these factors are multiplied by each other in the formula for the bin rates). The function of both parameters is evidently very similar. We can see that even though $M_* \neq 10^6 M_\odot$, we cannot distinguish the parameters from each other.

We also tried taking A to be constant and estimating just α and p_1, which gave similar results as above. Hence, unfortunately, it turns out that α and p_1 are degenerate, and we cannot estimate both of them simultaneously.

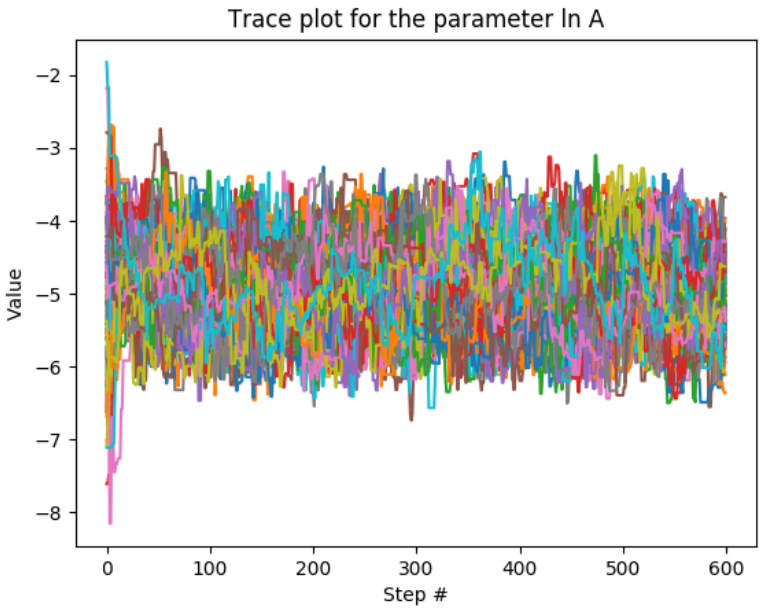


Figure 4.1: Trace plot of all the walkers in the $\ln A$ dimension of the posterior of a dataset with true parameters $A = 0.01$, $\alpha = -0.3$, $p_1 = -0.19$, $N_p = 10.0$ (fixed). Estimating $\ln A$, α and p_1 simultaneously. Number of walkers: 100, number of MCMC timesteps: 600.

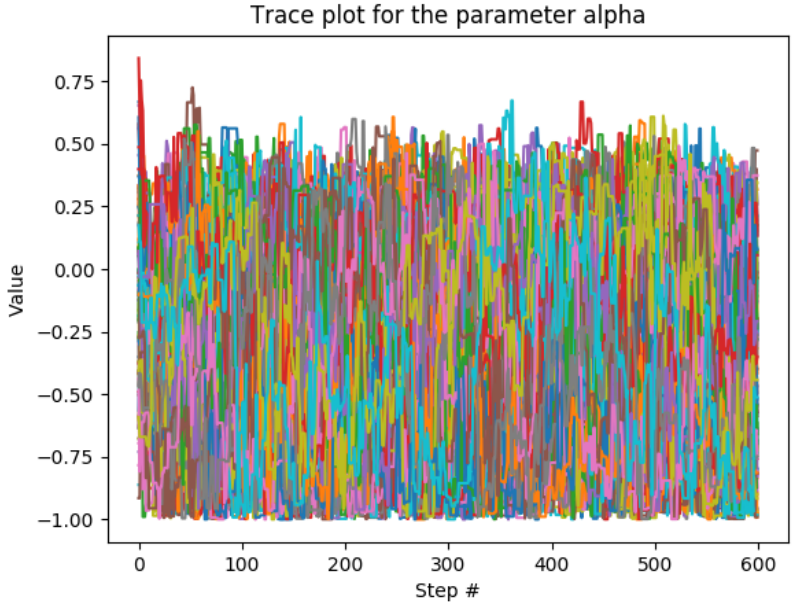


Figure 4.2: Trace plot of all the walkers in the α dimension of the posterior of a dataset with true parameters $A = 0.01$, $\alpha = -0.3$, $p_1 = -0.19$, $N_p = 10.0$ (fixed). Estimating $\ln A$, α and p_1 simultaneously. Number of walkers: 100, number of MCMC timesteps: 600.

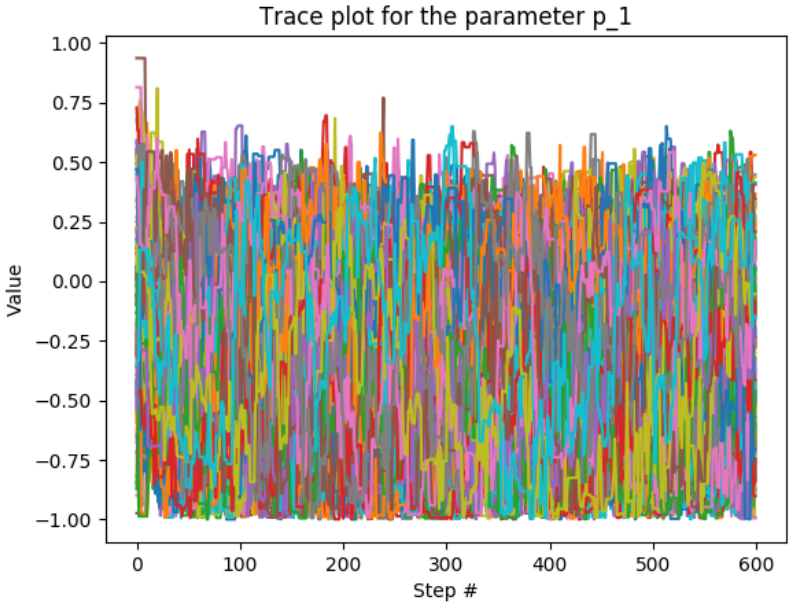


Figure 4.3: Trace plot of all the walkers in the p_1 dimension of the posterior of a dataset with true parameters $A = 0.01$, $\alpha = -0.3$, $p_1 = -0.19$, $N_p = 10.0$ (fixed). Estimating $\ln A$, α and p_1 simultaneously. Number of walkers: 100, number of MCMC timesteps: 600.

Estimating A, α and N_p simultaneously

Now, we are going to see if we have better luck with the N_p parameter. We shall introduce N_p as another variable (this time keeping p_1 fixed), which implies that $\vec{\theta} = (\ln A, \alpha, \ln N_p)$. We generated some datasets using the true parameter values discussed at the beginning of the chapter.

We ran MCMC on these datasets for 600 timesteps, using 100 walkers. The resulting trace plots for one particular dataset can be seen in Figures 4.5, 4.6 and 4.7. Note how this time, the walkers converge to a reasonably narrow range in the α dimension; however, the uncertainty in $\ln A$ appears to be very high, and in the $\ln N_p$ dimension the walkers appear to be completely uniformly distributed across the range specified by the prior.

The posterior distribution can be seen in Figure 4.8. Note how this time, the parameters $\ln A$ and $\ln N_p$ appear to be almost perfectly correlated. This indicates the parameters are degenerate. Thus, let us look at the formulae in Equations 1.11 (the MBH mass function) and 1.16 (the Gamma factor in the intrinsic EMRI rate of MBHs). A is essentially a scale parameter, and when A increases, the overall EMRI rate increases. The function of N_p depends on the value of Q: as long as $Q \geq 1$, N_p does not affect the rate. However, if $Q < 1$, it also functions almost as a scale parameter: when $1 + N_p$ increases, the overall rate decreases (see 1.2.2). Since N_p and A appear to be degenerate here, it seems likely that Q is always less than 1 for our choice of parameters. We checked whether that is true, and indeed Q varies between 0.08 and 0.13, when varying

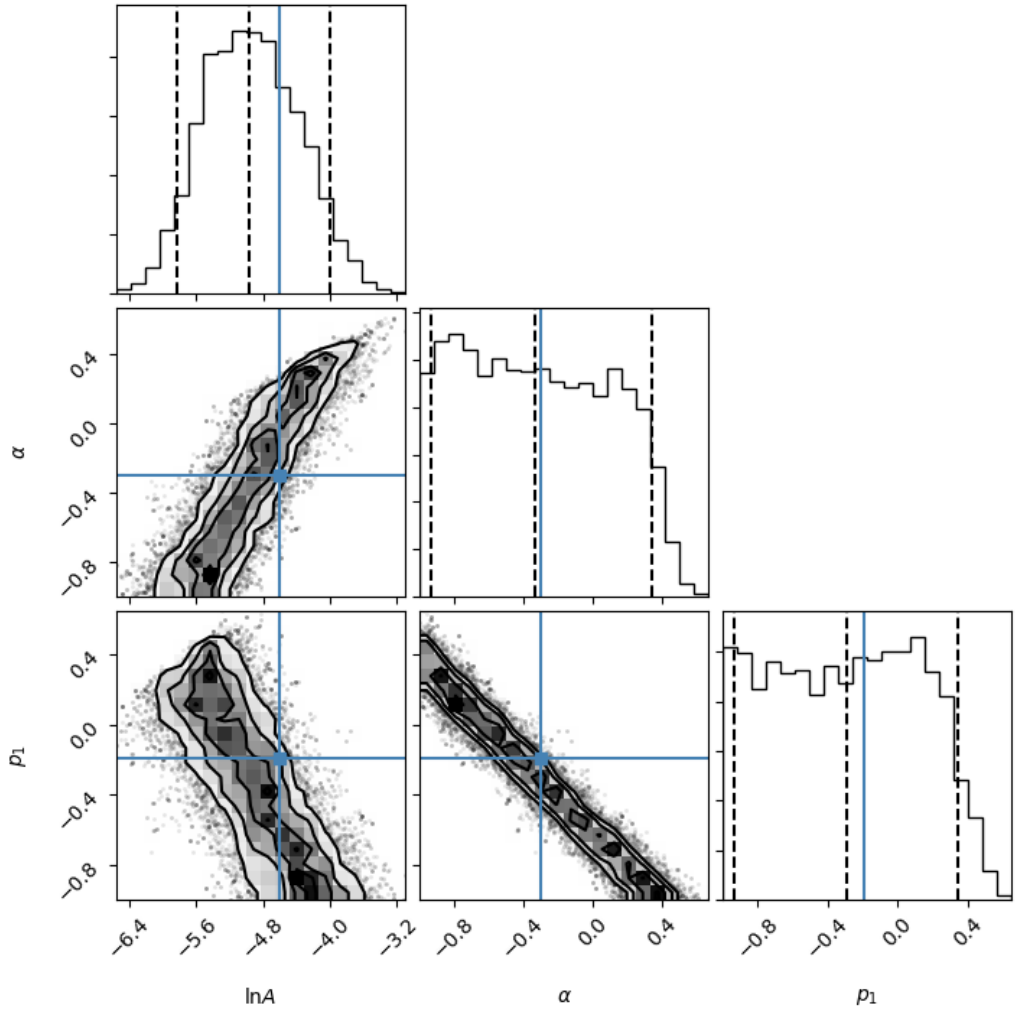


Figure 4.4: The corner plot of the MCMC sample from the posterior of a dataset with true parameters $A = 0.01$, $\alpha = -0.3$, $p_1 = -0.19$, $N_p = 10.0$ (fixed) (the blue lines indicate the true values of the parameters). Estimating $\ln A$, α and p_1 simultaneously. Number of walkers: 100, number of MCMC timesteps: 600, burn-in length: 400.

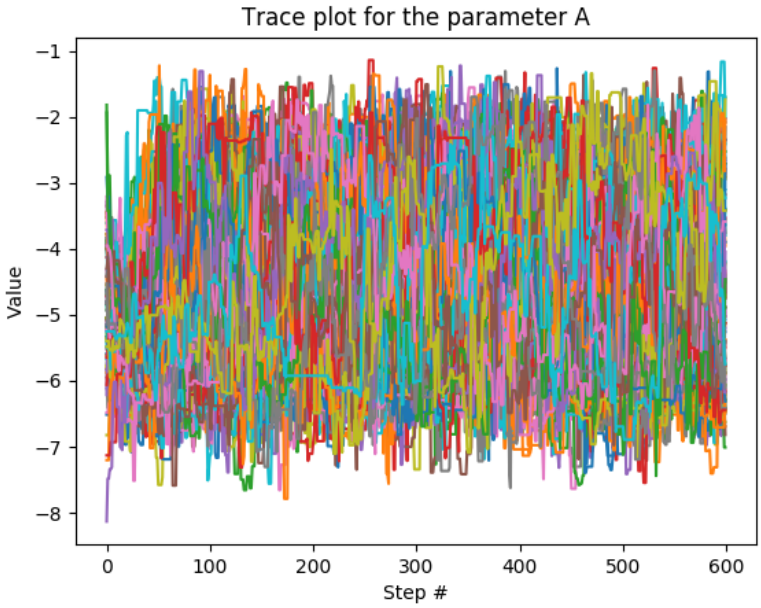


Figure 4.5: Trace plot of all the walkers in the $\ln A$ dimension of the posterior of a dataset with true parameters $A = 0.01$, $\alpha = -0.3$, $p_1 = -0.19$ (fixed), $N_p = 10.0$. Estimating $\ln A$, α and $\ln N_p$ simultaneously. Number of walkers: 100, number of MCMC timesteps: 600.

M from 10^4 to 10^7. It turns out that even if we take N_p to be as small as 1, Q will still be less than 1 in our mass range, which does not allow us to break the degeneracy between A and N_p.

We also tried fixing α and estimating just A and N_p, which also did not work. Therefore, even though the degeneracy between A and N_p is not perfect, it is large enough for us not to be able to distinguish these parameters from each other.

4.1.1 Analysis of the results

In the first case, when we were estimating $\ln A$, α and p_1 simultaneously, we found that the credible intervals for α and p_1 are much wider than the constraints in current theory. Also, it is evident that our choice of prior is influencing the estimation very much. This means we cannot make meaningful inferences about these parameters. The credible interval in A is quite wide as well.

In the second case (when estimating $\ln A$, α and $\ln N_p$), the credible intervals for A and N_p are much wider than theoretical constraints. Interestingly enough, we measured α reasonably well; however, it is not particularly useful to measure only one parameter well, if we could measure two of them well instead!

It turns our that A and p_1 are not degenerate and we were able to measure them simultaneously. Also, we found that α and N_p are not degenerate. However, we are much more interested in measuring p_1 and N_p simultaneously, since EMRIs provide quite a unique opportunity of constraining these parameters through the EMRI rate function.

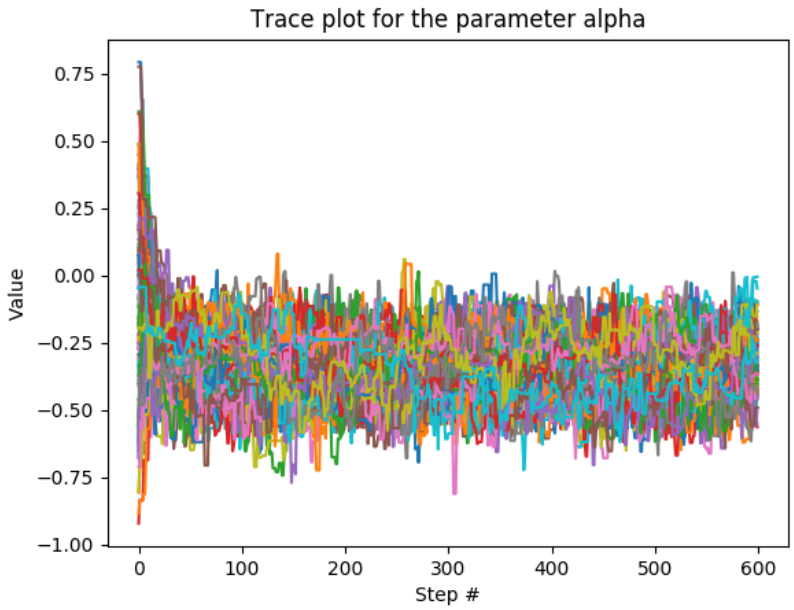


Figure 4.6: Trace plot of all the walkers in the α dimension of the posterior of a dataset with true parameters $A = 0.01$, $\alpha = -0.3$, $p_1 = -0.19$ (fixed), $N_p = 10.0$. Estimating $\ln A$, α and $\ln N_p$ simultaneously. Number of walkers: 100, number of MCMC timesteps: 600.

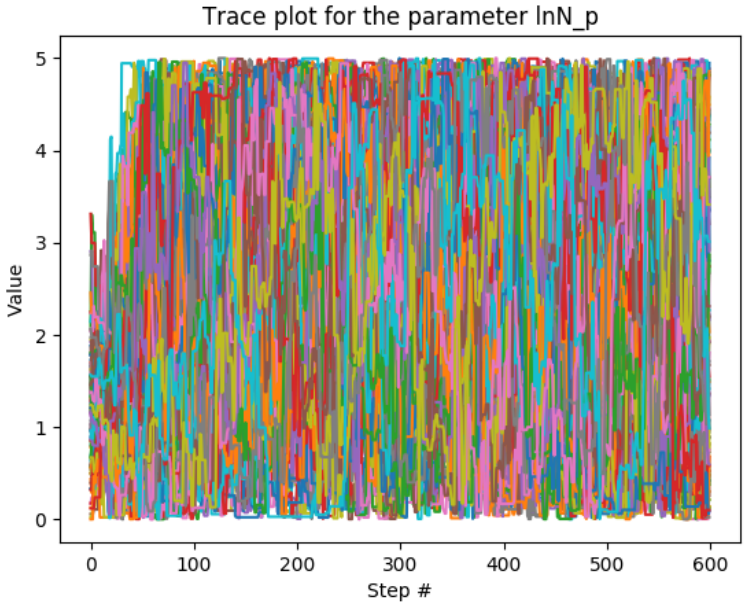


Figure 4.7: Trace plot of all the walkers in the $\ln N_p$ dimension of the posterior of a dataset with true parameters $A = 0.01$, $\alpha = -0.3$, $p_1 = -0.19$ (fixed), $N_p = 10.0$. Estimating $\ln A$, α and $\ln N_p$ simultaneously. Number of walkers: 100, number of MCMC timesteps: 600.

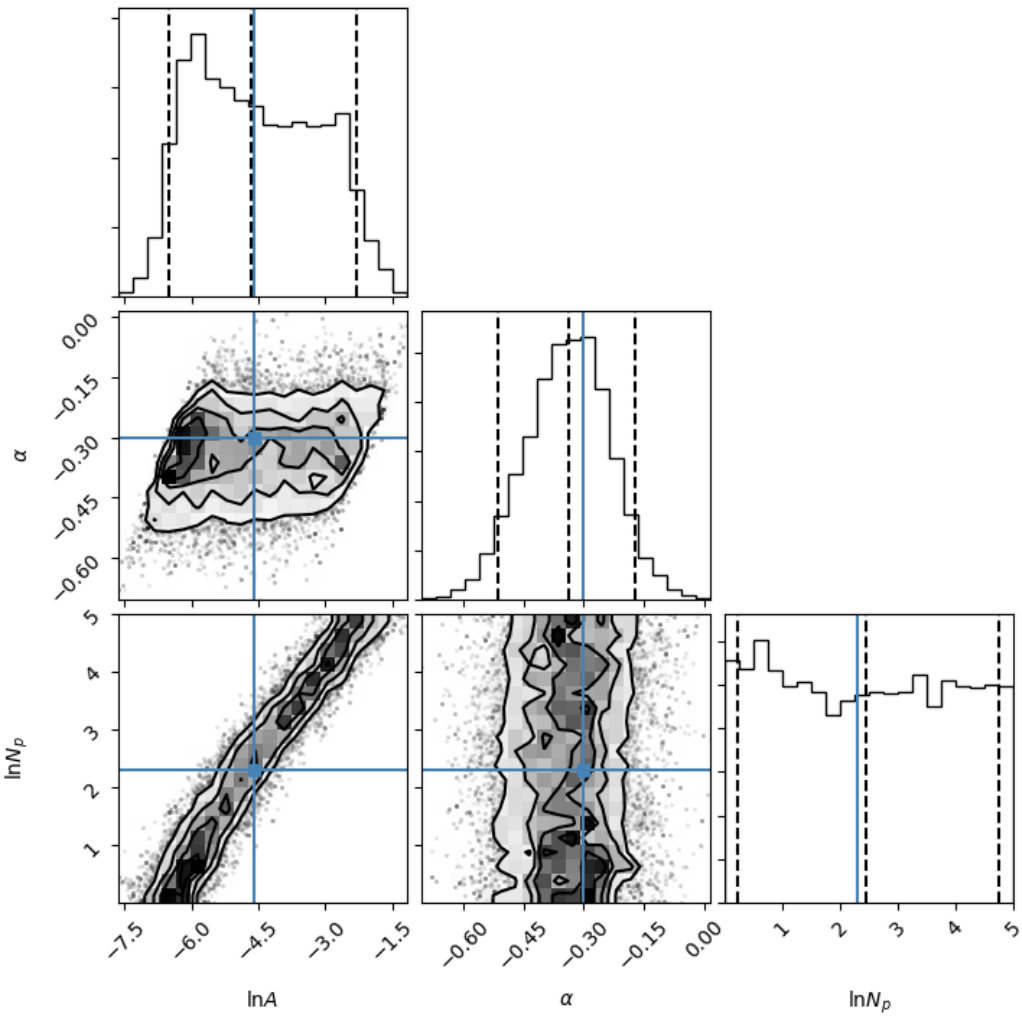


Figure 4.8: The corner plot of the MCMC sample from the posterior of a dataset with true parameters $A = 0.01$, $\alpha = -0.3$, $p_1 = -0.19$ (fixed), $N_p = 10.0$ (the blue lines indicate the true values of the parameters). Estimating $\ln A$, α and $\ln N_p$ simultaneously. Number of walkers: 100, number of MCMC timesteps: 600, burn-in length: 400.

Now, it seems plausible that if only we knew the parameters A and α precisely, we might be able to measure p_1 and N_p. It turns out that there might be another way of constraining the MBH mass function. As was mentioned in the introduction, LISA will also be able to measure MBH mergers. That data should help us constrain the MBH mass function - we should be able to measure the parameters A and α to a reasonably high precision. Hence, let us assume that we know A and α, and try to estimate p_1 and N_p.

4.2 Estimating the properties of stellar clusters

As we discussed at the end of the previous section, we shall fix A and α and try to estimate p_1 and N_p. As before, we generated some datasets using the true parameter values given at the beginning of the chapter.

The resulting trace plots from one particular dataset can be seen in Figures 4.9 and 4.10. This time the walkers converge rather quickly. The posterior distribution is summarised in Figure 4.11. Note how this time the parameters do not appear to be extremely correlated, and visually it seems it is possible to estimate them well.

The resulting interval estimates confirm this. The median estimate of N_p is 9.0, with a corresponding 90% credible interval $(5.4, 15.3)$. Note how this is one tenth of the uncertainty in N_p in current literature! Hence, using this analysis, we would effectively be able to determine N_p to a much higher precision than before.

The median estimate of p_1 is -0.14, with a corresponding 90% credible interval $(-0.32, 0.03)$. This range contains both of our reference values -0.15 and -0.19. However, for example, it does not contain $3/8$, which is a value that has been theoretically predicted in the past. Hence, if our assumptions hold, our methods of analysis would be able to rule that possibility out. The other datasets we analysed gave similar results.

In Appendix A, there are some more plots of results produced with different true parameters.

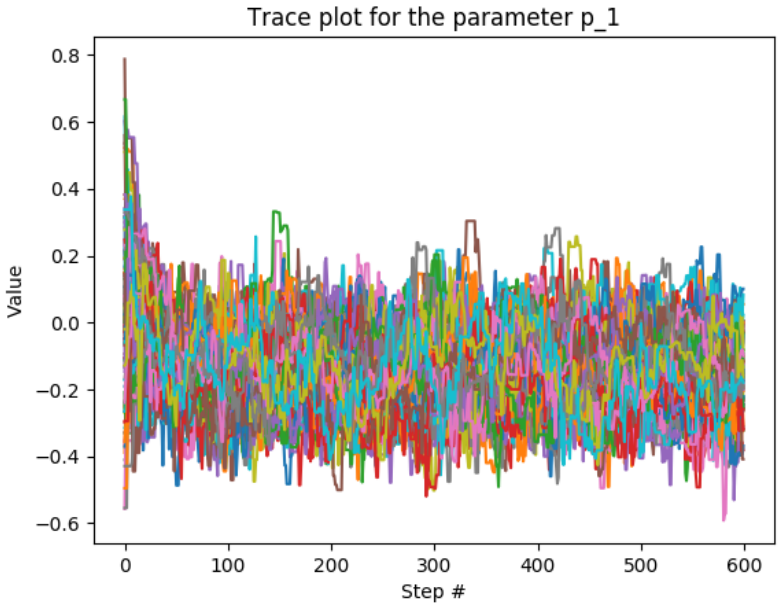


Figure 4.9: Trace plot of all the walkers in the p_1 dimension of the posterior of a dataset with true parameters $A = 0.01$ (fixed), $\alpha = -0.3$ (fixed), $p_1 = -0.19$, $N_p = 10.0$. Estimating p_1 and $\ln N_p$ simultaneously. Number of walkers: 100, number of MCMC timesteps: 600.

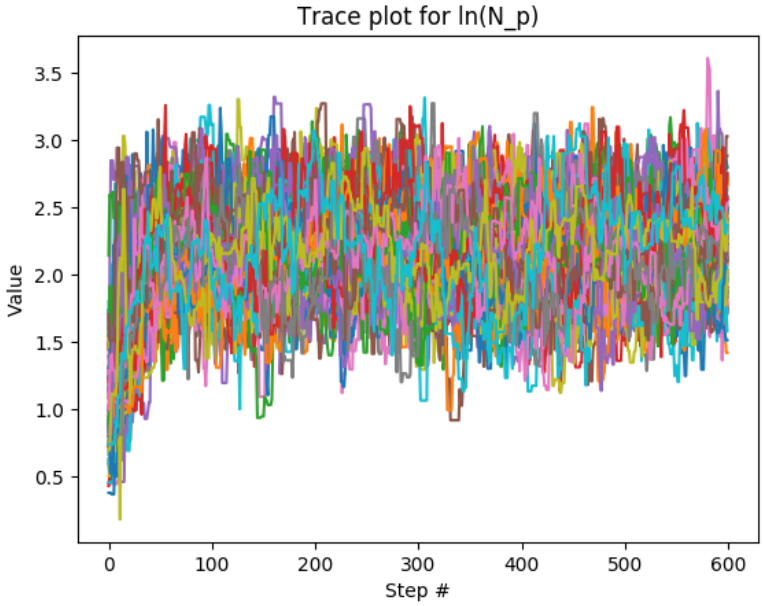


Figure 4.10: Trace plot of all the walkers in the $\ln N_p$ dimension of the posterior of a dataset with true parameters $A = 0.01$ (fixed), $\alpha = -0.3$ (fixed), $p_1 = -0.19$, $N_p = 10.0$. Estimating p_1 and $\ln N_p$ simultaneously. Number of walkers: 100, number of MCMC timesteps: 600.

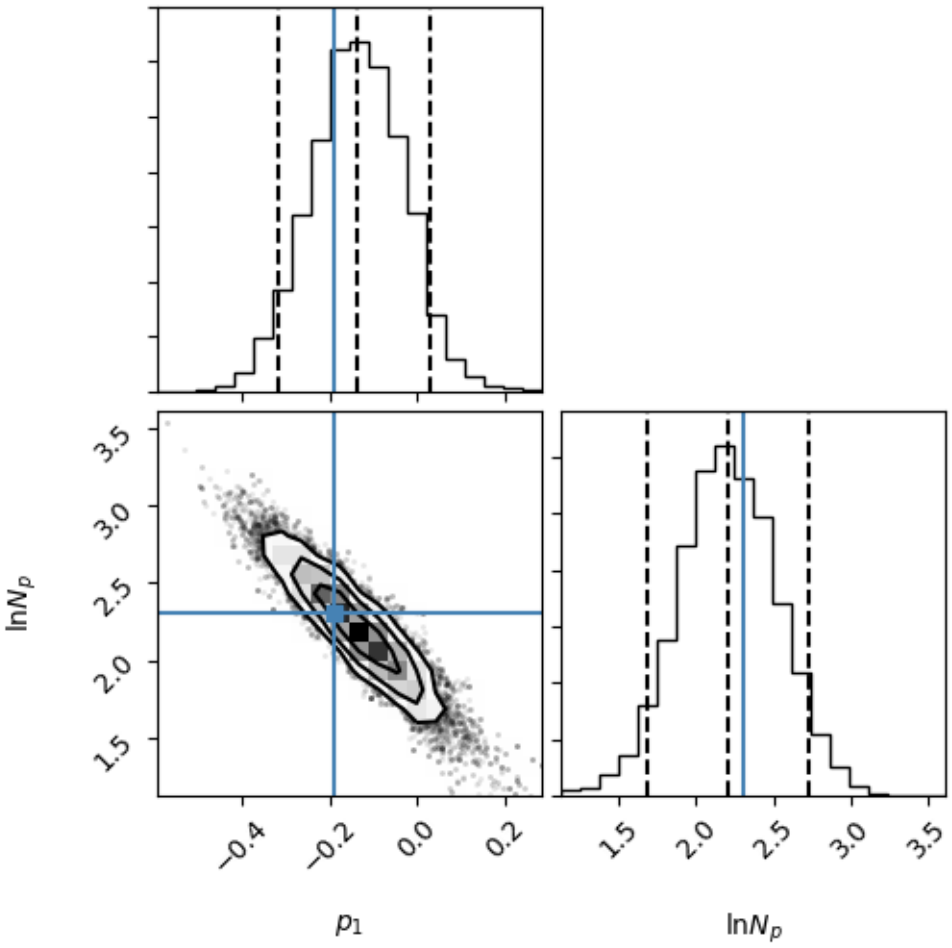


Figure 4.11: The corner plot of the MCMC sample from the posterior of a dataset with true parameters $A = 0.01$ (fixed), $\alpha = -0.3$ (fixed), $p_1 = -0.19$, $N_p = 10.0$. Estimating p_1 and $\ln N_p$ simultaneously. Number of walkers: 100, number of MCMC timesteps: 600, burn-in length: 400.

Conclusion

In our analysis, we have explored the ability of the Laser Interferometer Space Antenna (LISA) to determine the properties of the massive black hole (MBH) distribution in the universe and the dynamics of the stellar clusters surrounding them, using the gravitational wave signals emitted by EMRI events. We introduced a theoretical model for these events, which combined our knowledge of the MBH population (the MBH mass function), the intrinsic EMRI rate of particular MBHs (which relates to the stellar clusters where EMRIs originate from), and the sensitivity of LISA to different EMRI events. Then, we generated data from these models, and analysed it within a Bayesian framework using MCMC.

First, we implemented the model from Gair et al. [4] and attempted to constrain the parameters governing the MBH mass function. We successfully managed to reproduce their results and constrain our parameters of interest to a much narrower range than the constraints in current literature.

Then, we implemented a more sophisticated model described in Babak et al. [2], which takes into account the fact that the dependence of the intrinsic EMRI rate of a MBH on its mass is not well known, and the fact that compact objects (such as stellar-mass BHs and neutron stars) plunging directly into the MBH also contribute to its mass growth. We introduced new parameters describing the dynamics of the stellar clusters where EMRIs originate from. Using this model, we attempted to estimate these new parameters and the parameters governing the MBH mass function simultaneously, and found that some of the parameters are degenerate. Thus, we concluded that we cannot estimate the MBH mass function and properties of stellar clusters simultaneously.

However, we may be able to extract information about the MBH mass function from other sources, such as MBH merger data, whereas EMRIs are a unique source of information for probing the properties of these stellar clusters. We found that if we assume we know the MBH mass function, we will be able to estimate the parameters of the EMRI rate function (which gives us information about stellar clusters) to a much higher precision than the constraints in current literature.

Future work

In the future, it would be interesting to run MCMC on more sets of data generated using different true parameter values, and see how the precision of our estimate depends on the values of these parameters. This way, we could determine what results we expect to see if we make particular assumptions about our models.

Also, we made the conservative assumption that all MBHs have zero spin, which most likely is not true. Hence, in the future this analysis should be repeated

without this restrictive assumption.

In the future, we should also analyse MBH merger data, to test the validity of our assumption that it could be used to determine the parameters of the MBH mass function. Also, it may be a good idea to repeat our analysis, but instead of keeping the parameters governing the MBH mass function constant (and hence assuming that we know them perfectly), we could introduce them as parameters with prior densities reflecting our previous estimates, along with the respective uncertainties. It would be interesting to see if the parameters in the EMRI rate function could still be constrained.

Also, there might be another way to break the degeneracy between some of our parameters. In Babak et al. [2], there is another factor κ in the model for the EMRI rate, which we did not incorporate here. That factor also depends on $R_0(\vec{\lambda}|\vec{\theta})$ (see Equation 1.15), and $\Gamma(\vec{\lambda}|\vec{\theta})$ (given in Equation 1.16). Hence, it depends on p_1 and N_p; however, it does not depend on A or α. Therefore, it brings additional information about p_1 and N_p, which might help break their respective degeneracies with α and A.

Additionally, it would be useful to introduce a more detailed discussion of convergence, such as calculating the autocorrelation of the MCMC chains. Right now, it is easy to check if the chain truly has converged, because we know the true values of our parameters. In the future, however, when actual physical data will be analysed, these issues will become much more important.

Bibliography

[1] Abbott, B. P. et al. (2016) Observation of Gravitational Waves from a Binary Black Hole Merger. *Phys. Rev. Lett.* **116**, 061102.

[2] Babak, S. et al. (2017) Science with the space-based interferometer LISA. V: Extreme mass-ratio inspirals. *Phys. Rev. D* **95**, 103012.

[3] Abbott, B. P. et al. (2017) GW170817: Observation of Gravitational Waves from a Binary Neutron Star Inspiral. *Phys. Rev. Lett.* **119**, 161101.

[4] Gair, J. R., Tang, C. and Volonteri, M. (2010) LISA extreme-mass-ratio inspiral events as probes of the black hole mass function. *Phys. Rev. D* **81**, 104014.

[5] Gair, J. et al. (2004) Event rate estimates for LISA extreme mass ratio capture sources. *Class. Quantum Grav.* **21**, S1595-S1606.

[6] Gregory, P. (2005) *Bayesian Logical Data Analysis for the Physical Sciences.* Cambridge: Cambridge University Press.

[7] Gair, J. (2009) Probing black holes at low redshift using LISA EMRI observations. *Class. Quantum Grav.* **26**, 094034.

[8] Hogg, D. (2000) Distance measures in cosmology. *E-print arXiv*, 9905116.

[9] The AstroPy collaboration. (2018) The Astropy Project: Building an inclusive, open-science project and status of the v2.0 core package. *E-print arXiv*, 1801.02634.

[10] Bar-Or, B. and Alexander, T. (2016) Steady-state Relativistic Stellar Dynamics Around a Massive Black Hole. *The Astrophysical Journal* **820**, 2.

[11] Amaro-Seoane, P. et al. (2017) Laser Interferometer Space Antenna. *E-print arXiv*, 1702.00786.

[12] Foreman-Mackey, D. et al. (2013) emcee: The MCMC Hammer. *Publications of the Astronomical Society of the Pacific*, **125**, 925.

[13] Goodman, J. and Weare, J. (2010) Ensemble samplers with affine invariance. *Communications in Applied Mathematics and Computational Science* **5**, 1.

[14] Abbott, B. P. et al. (2016) Properties of the Binary Black Hole Merger GW150914. *Phys. Rev. Lett.* **116**, 241102.

Appendix A: More results

Results of estimating the simple power law mass function

Below are some more results from the analysis done in Chapter 3, using different true parameters. Sampling was done in $\ln A$ and α. In all the plots the number of walkers is 100 and the number of MCMC timesteps is 600.

Dataset 1

True parameters $A = 0.002$, $\alpha = -0.3$. The total number of events was 229.

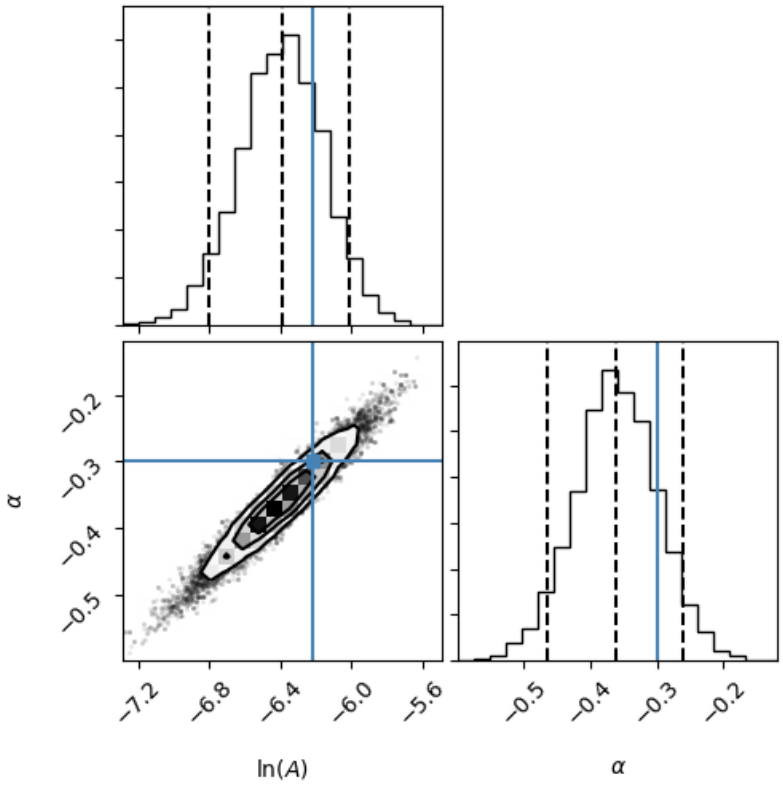


Figure 4.12: The corner plot of the MCMC sample from the posterior.

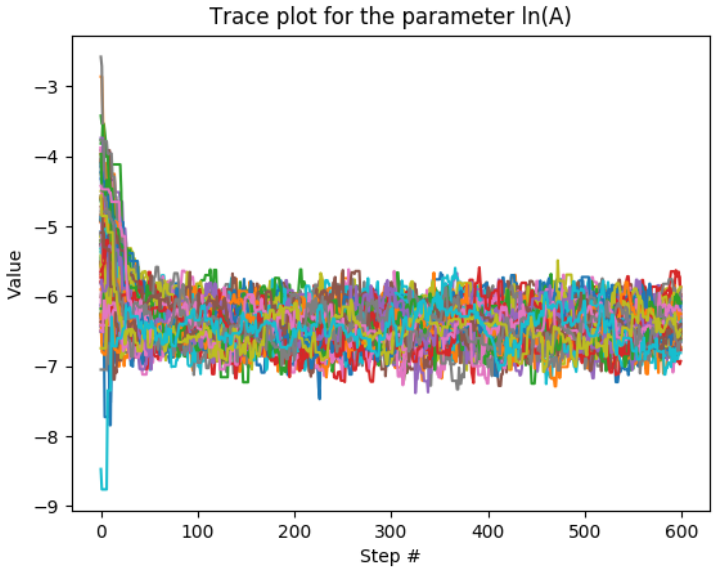


Figure 4.13: Trace plot of all the walkers in the $\ln A$ dimension of the posterior.

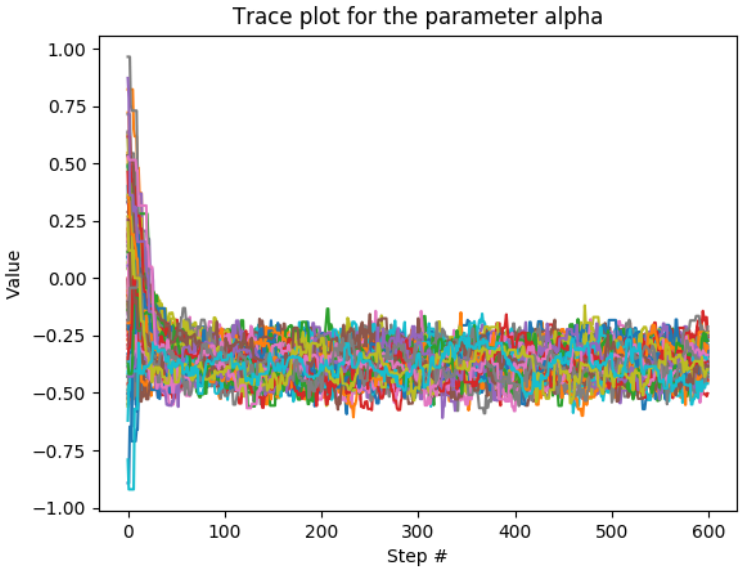


Figure 4.14: Trace plot of all the walkers in the α dimension of the posterior.

Dataset 2

True parameters $A = 0.01$, $\alpha = 0.3$. The total number of EMRI events was 121.

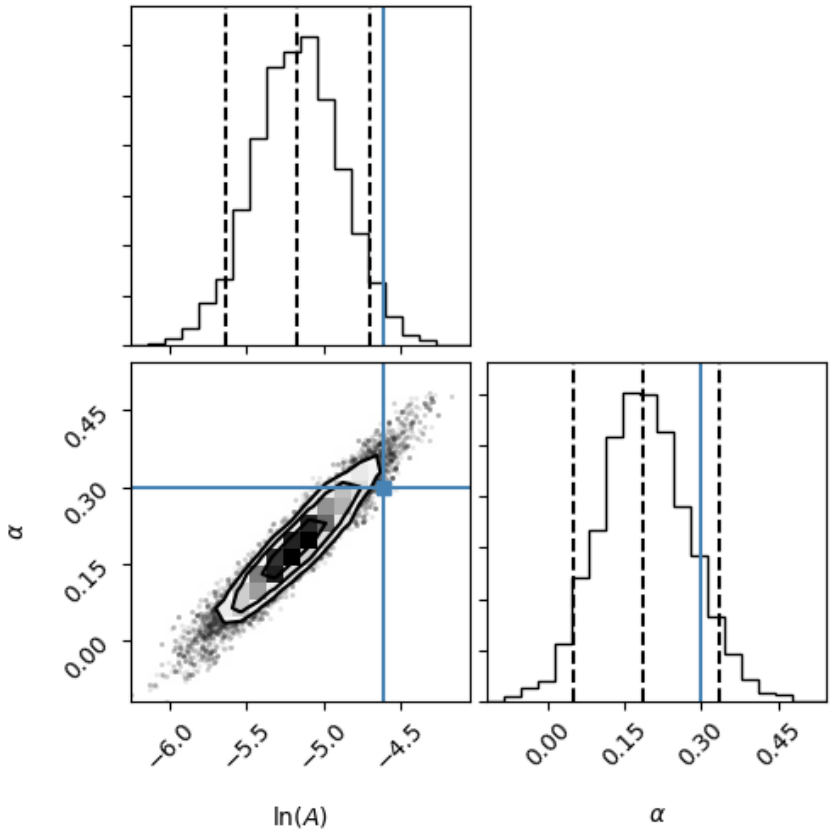


Figure 4.15: The corner plot of the MCMC sample from the posterior.

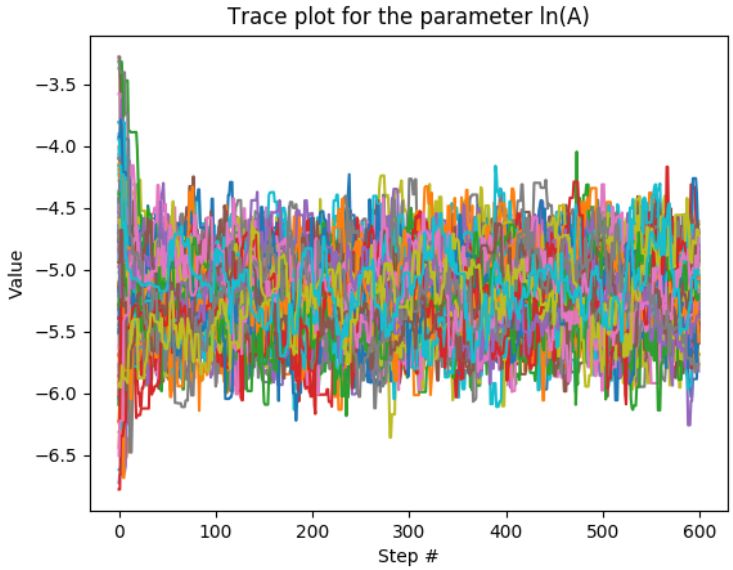


Figure 4.16: Trace plot of all the walkers in the $\ln A$ dimension of the posterior.

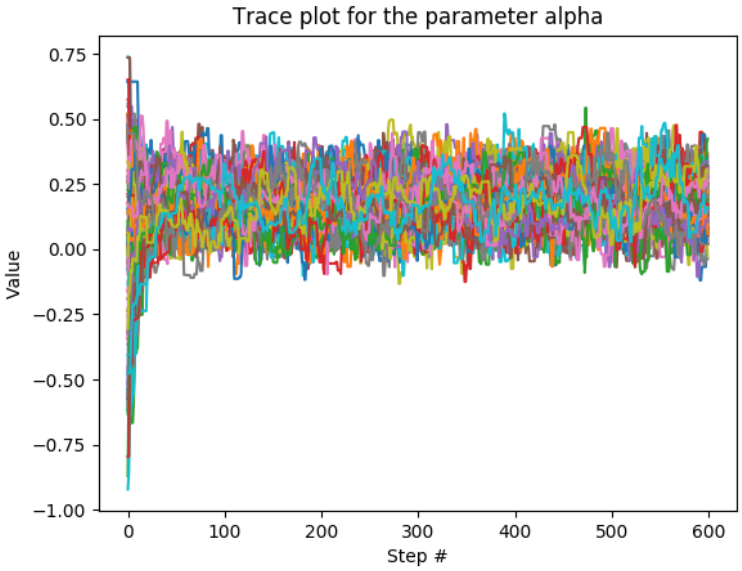


Figure 4.17: Trace plot of all the walkers in the α dimension of the posterior.

Dataset 3

True parameters $A = 0.0005$, $\alpha = -0.3$. The total number of EMRI events was 61.

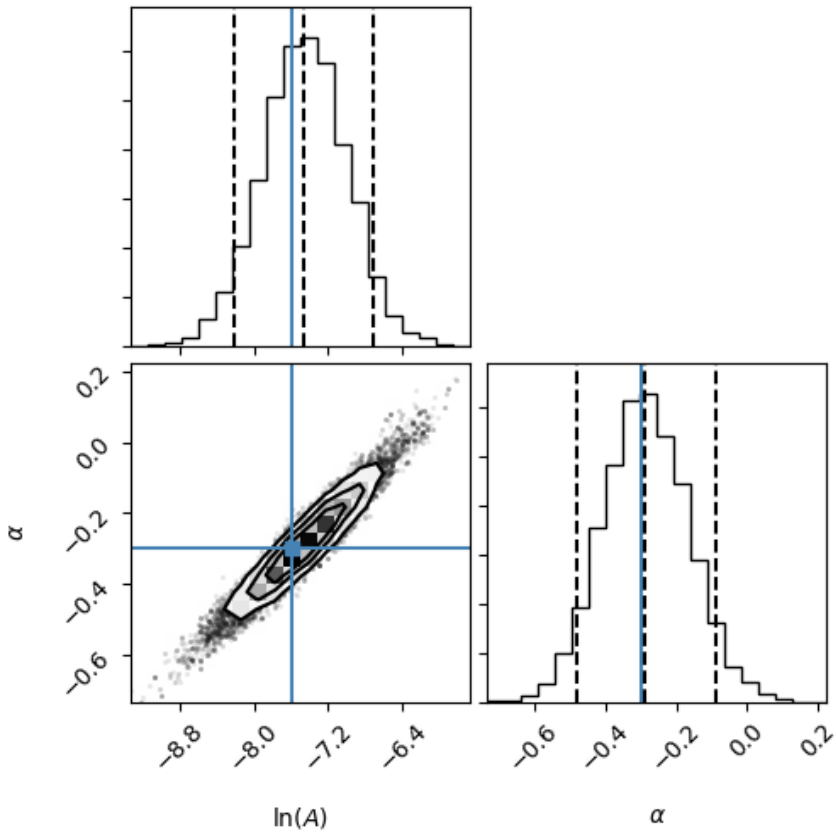


Figure 4.18: The corner plot of the MCMC sample from the posterior.

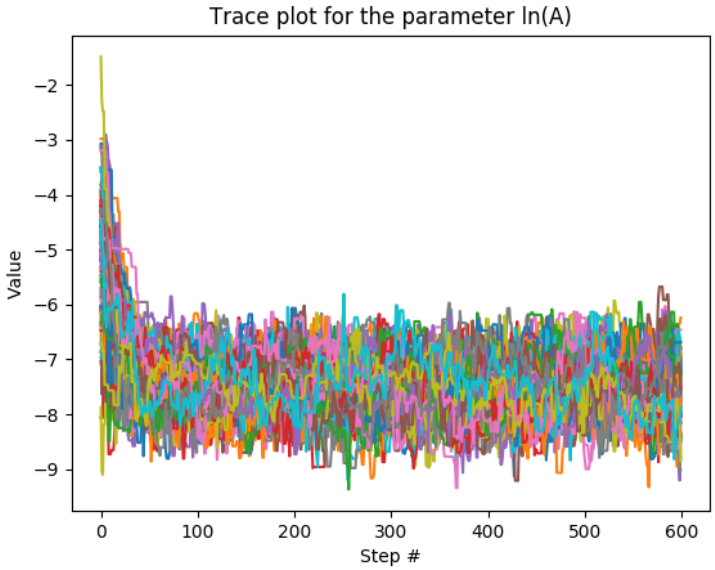


Figure 4.19: Trace plot of all the walkers in the $\ln A$ dimension of the posterior.

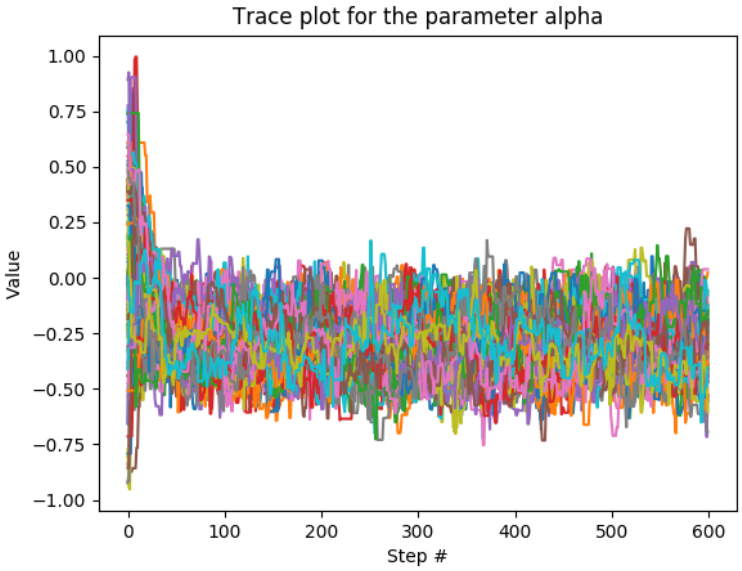


Figure 4.20: Trace plot of all the walkers in the α dimension of the posterior.

Results of estimating the properties of stellar clusters

Below are some more results from the analysis done in Section 4.2, using different true parameters. Sampling was done in p_1 and $\ln N_p$. In all the plots the number of walkers is 100 and the number of MCMC timesteps is 600. Also, $A = 0.01$ and $\alpha = -0.3$.

Dataset 1

True parameters $p_1 = 0.0$, $\ln N_p = 10.0$. The total number of events was 53.

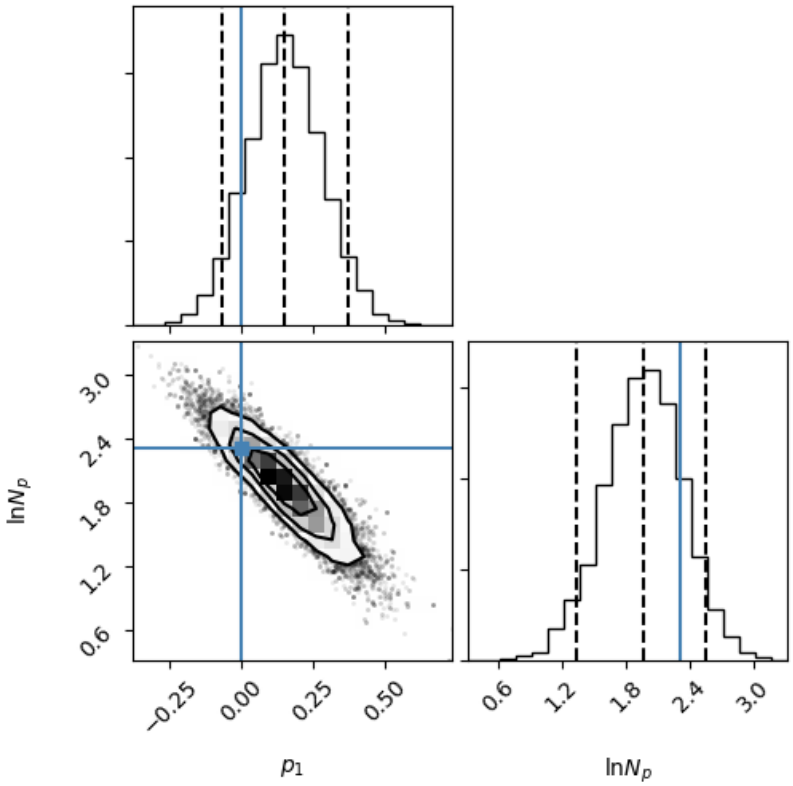


Figure 4.21: The corner plot of the MCMC sample from the posterior.

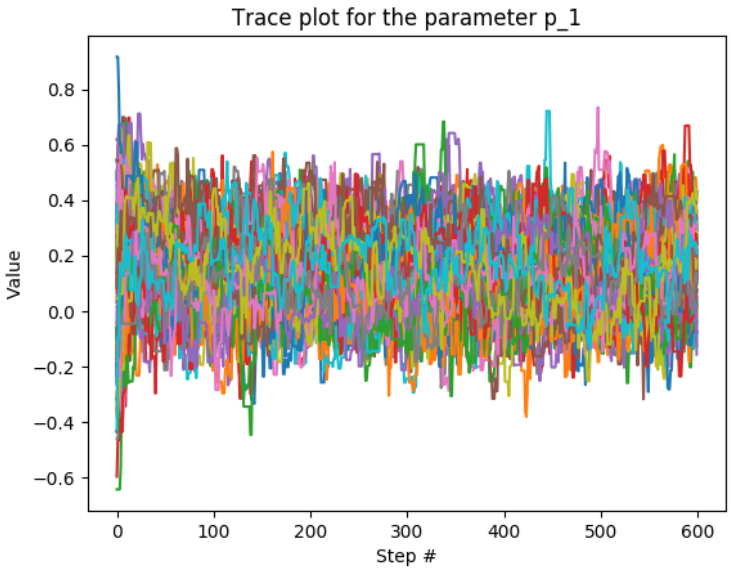


Figure 4.22: Trace plot of all the walkers in the p_1 dimension of the posterior.

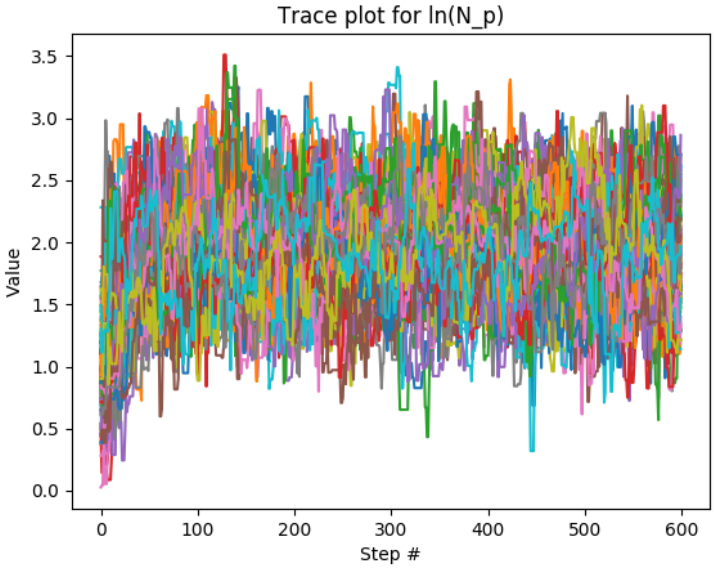


Figure 4.23: Trace plot of all the walkers in the $\ln N_p$ dimension of the posterior.

Dataset 2

True parameters $p_1 = 3/8$, $\ln N_p = 10.0$. The total number of events was 21.

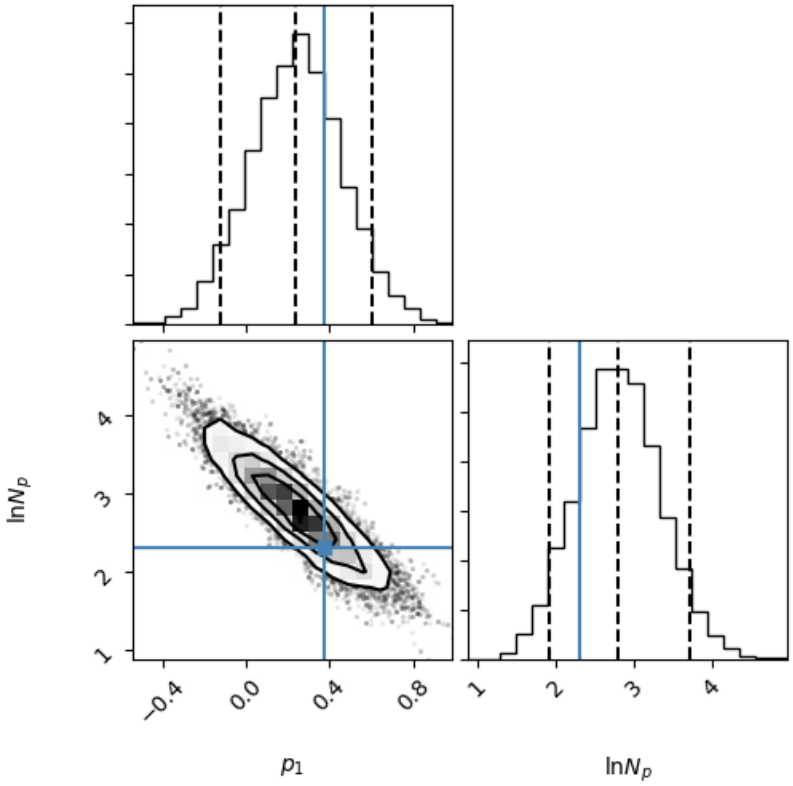


Figure 4.24: The corner plot of the MCMC sample from the posterior.

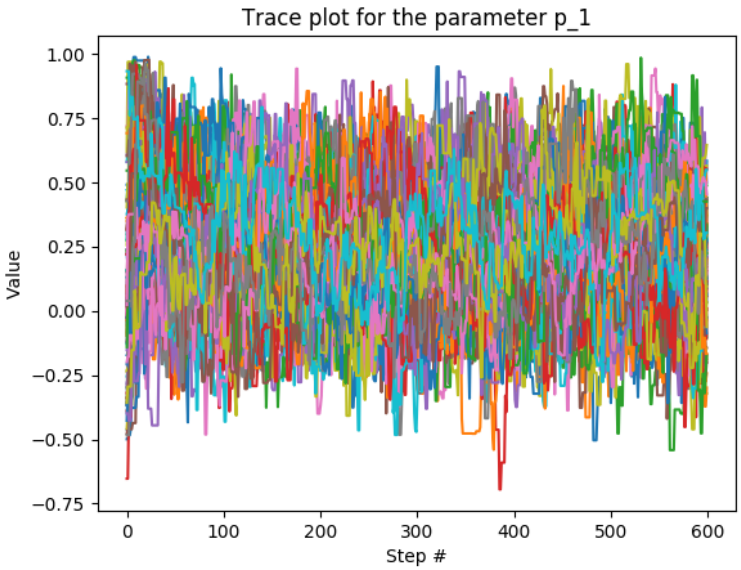


Figure 4.25: Trace plot of all the walkers in the p_1 dimension of the posterior.

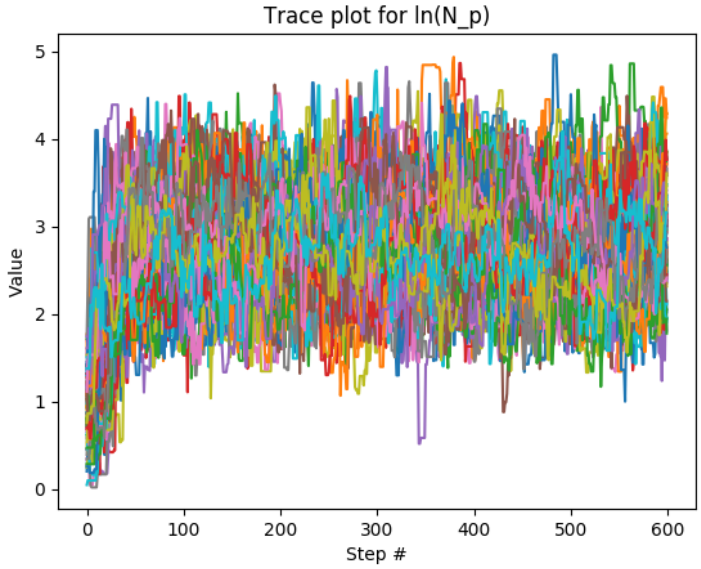


Figure 4.26: Trace plot of all the walkers in the $\ln N_p$ dimension of the posterior.

Dataset 3

True parameters $p_1 = -0.19$, $\ln N_p = 50.0$. The total number of events was 17.

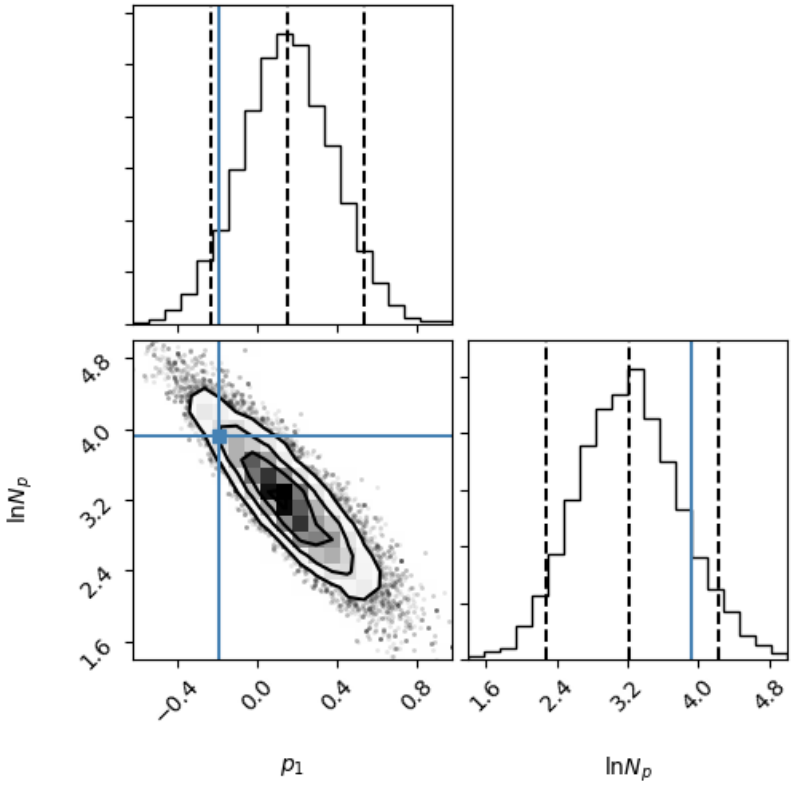


Figure 4.27: The corner plot of the MCMC sample from the posterior.

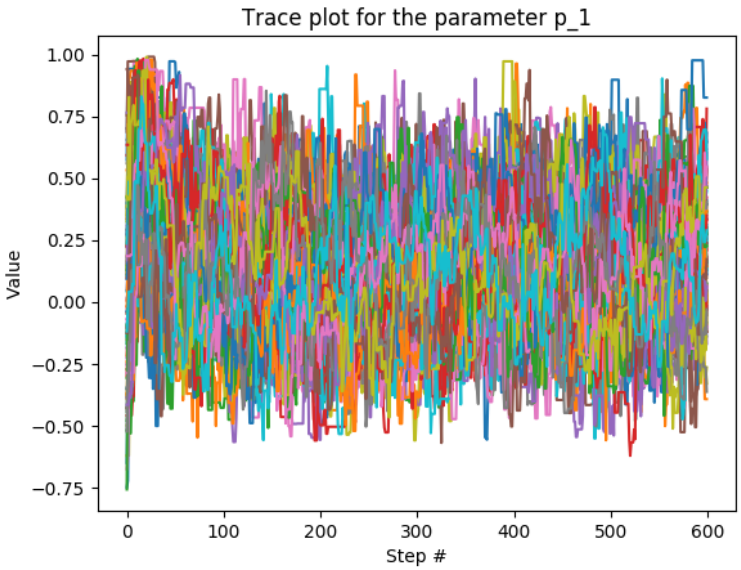


Figure 4.28: Trace plot of all the walkers in the p_1 dimension of the posterior.

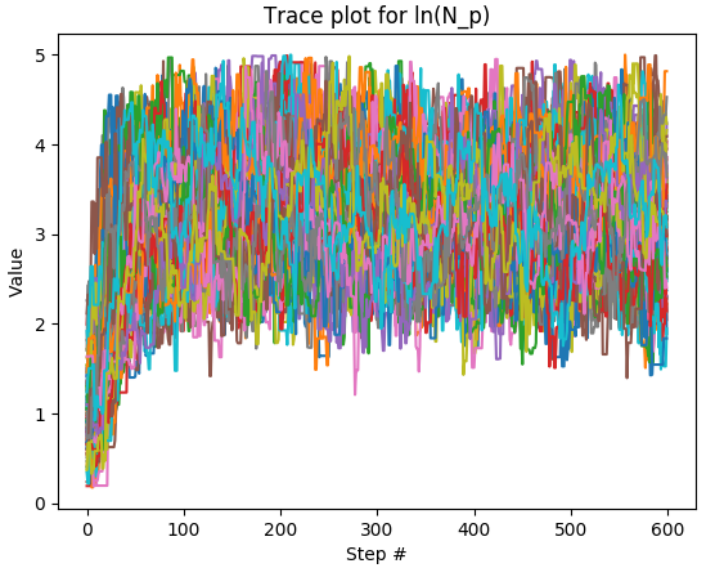


Figure 4.29: Trace plot of all the walkers in the $\ln N_p$ dimension of the posterior.

Appendix B: Computer code

These are the code files for the implementation of the Affine Invariant Ensemble Sampler for estimating the MBH mass function, which is described in Chapter 3.

The following is the code file `shared_stuff.py`, which contains all the basic variables and functions that are used in both data generation and analysis. All of these will be imported into the other code files at the beginning.

```
# This file contains the key variables and functions used in all code files.

# This file shouldn't contain anything besides the most important stuff that is
# used by most other code files, since every variable, function, etc in this
# file will be imported into all the other code files, and we don't want to have
# accidental name clashes.

import numpy as np
import math as m
from astropy.cosmology import Planck15

# Number of parameters to be estimated:
ndim = 2

# Number of datasets to be analysed:
n_data = 15

# Fix p_1:
p019 = -0.15

# Mass will be expressed in terms of 3 * 10^6 solar masses - define the scaling factor:
mass_scale_f = 3e6

# Let's set the parameter ranges, bin widths, etc.
lnM_lower = m.log(1e4/mass_scale_f) # approx 9.3
lnM_upper = m.log(1e7/mass_scale_f) # approx 16.1
lnM_nbins = 20
dlnM = (lnM_upper - lnM_lower)/lnM_nbins # bin width

z_lower = 0.05
z_upper = 1.2
z_nbins = 20
dz = (z_upper - z_lower)/z_nbins # bin width

```

```
35 # Total number of bins:
36 n_bins = lnM_nbins*z_nbins
37
38
39 # The main EMRI rate function.
40 # --Input--
41 # muu - vector of parameters
42 # lnM, z - lower ends of a particular bin
43 # --Output--
44 # The expected EMRI rate for that bin
45 def rate(muu, lnM, z):
46   return r0(muu, lnM+dlnM/2)*muu[0]*m.exp(muu[1]*(lnM+dlnM/2))*dlnM*
       dz
47   # Note that we add half of the bin width to lnM to calculate the
       rate for the midpoint of the bin
48   # NB! We will multiply this rate vector with the comoving volume
       and observable lifetime separately (see volume_obslife_factors
       below). That is done to reduce the computational cost of each
       MCMC timestep, since the observable lifetimes and comoving
       volumes don't explicitly depend on the model parameters (the '
       rate' function will be called 100 times per MCMC timestep, so we
       don't want to include unnecessary computation).
49
50 # The EMRI rate of an individual MBH.
51 # --Input--
52 # muuu - vector of parameters
53 # lnM - log mass
54 # --Output--
55 # the EMRi rate of an individual MBH with log mass ln M (in Gyr^(-1)
       )
56 def r0(muuu, lnM):
57   return 400*(3**p019)*m.exp(p019*lnM)
58   # here, we should divide by 10e9 to get the rate in y^(-1) instead
       of Gyr^(-1);
59   # however, we divide the differential comoving volume by that
       instead (and get it in Gpc instead of Mpc),
60   # so it cancels out!
61
62 # The observable lifetime function - to be multiplied with the EMRI
       rate separately (see volume_obslife_factors)
63 # --Input--
64 # lnM, z - log mass, redshift
65 # --Output--
66 # The observable lifetime (in years)
67 def ObsLifeSpin4(lnM, z):
68   zlist =
       [0.05,0.1,0.15,0.2,0.25,0.3,0.35,0.4,0.45,0.5,0.55,0.6,0.65, ...]
69   Mlist = [10000,12996.6,16891.1,21952.6,28530.8,37080.3,48191.6,
       ...]
70   obslife =
       [[23.8889,12.5758,6.66667,3.68056,1.93333,0.769231,0,0,0,0,0,0,
       0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
       0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
       [22.1429,12.8788,7.3913,4.51389,2.73333,1.53846,0.740741,
       0.0595238,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
       0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
       ...]
```

```
    # these arrays define the observable lifetime for each M, z value
    # note: we had to shorten them to fit them in the report

  NM=30
  Nz=60
  M = mass_scale_f * m.exp(lnM)

  i=0
  if M < Mlist[0]:
    i = 1
    print "Warning - mass below minimal mass in function!"
  elif M > Mlist[NM-1]:
    i = NM - 1
    print "Warning - mass above maximal mass in function!"
  else:
    while Mlist[i] <= M and i < NM-1:
      i += 1
  j = 0
  if z < zlist[0]:
    j = 1
    print "Warning - redshift below minimal redshift in function!"
  elif z > zlist[Nz-1]:
    j = Nz - 1
    print "Warning - redshift above maximal redshift in function!"
  else:
    while zlist[j] <= z and j < Nz-1:
      j += 1

  life = obslife[i-1][j-1] * (Mlist[i] - M) * (zlist[j] - z)
  life += obslife[i][j-1] * (M-Mlist[i-1]) * (zlist[j] - z)
  life += obslife[i-1][j] * (Mlist[i]-M) * (z - zlist[j-1])
  life += obslife[i][j] * (M-Mlist[i-1]) * (z - zlist[j-1])
  life /= (Mlist[i] - Mlist[i-1]) * (zlist[j] - zlist[j-1])
  return life

# Generate a vector of values which will be multiplied with the corresponding EMRI rate in each bin.
# These factors represent the differential comoving volumes and observable lifetimes.
volume_obslife_factors = list()
for k in range(lnM_nbins):
  for j in range(z_nbins):
    lnM_eff = lnM_lower + k*dlnM + dlnM/2 # add half of the bin width to get the midpoint
    z_eff = z_lower + j*dz + dz/2
    volume_obslife_factors.append(ObsLifeSpin4(lnM_eff, z_eff) * Planck15.differential_comoving_volume(z_eff).value / 10e9 * 4 * m.pi)
```

Data generation

This is the algorithm that was used for data generation.

```
from shared_stuff import *
```

```
2
3 # Let's define our model parameters A and alpha:
4 mu = [0.002, 0.0]
5
6 parameters = [mu]*n_data
7
8 # Now, we're going to generate n_data datafiles into the folder /
    data.
9 for i in range(n_data):
10   # Calculate the expected EMRI rates for each bin.
11   # Bins will be arranged in a list in the following way: the first
      z_nbins elements will have lnM=lnM_lower with z starting from
      z_lower and increasing all the way to z_upper-dz. The next
      z_nbins elements will have lnM=lnM_lower+dlnM with z increasing,
      etc.
12   rates_temp = list()
13   for k in range(lnM_nbins):
14     for j in range(z_nbins):
15       rates_temp.append(rate(parameters[i], lnM_lower+k*dlnM,
     z_lower+j*dz))
16   rates = np.multiply(rates_temp, volume_obslife_factors)
17   samples = np.random.poisson(rates)
18
19   datafile = open("data/data%d.txt" % i, 'w+')
20   datafile.write("The first ndim lines (after this one) contain the
      true parameters (A, alpha). (p019 is fixed to be "+str(p019)+".)
      The rest contain the rate of each bin and the corresponding
      sampled value (on the same line). Total number of events: "+str(
     sum(samples))+".\n")
21   for k in range(ndim):
22     datafile.write(str(parameters[i][k])+"\n") # write the
      parameters used to generate the data into the file
23
24   for j in range(n_bins):
25     datafile.write(str(rates[j])+" "+str(samples[j])+"\n")
26   datafile.close()
```

The MCMC algorithm

This is the author's implementation of the MCMC algorithm.

```
1 import matplotlib.pyplot as pl
2 import emcee
3 import corner
4
5 from shared_stuff import *
6
7 # Let's define the log prior:
8 def lnprior(p):
9   if p[0] > 0 and p[0] < 0.05 and p[1] > -1 and p[1] < 1:
10          return -m.log(p[0])
11        else:
12                return -np.inf
13
```

```
14 # The likelihood of "data" (a vector of values assumed to be from a
       Poisson distribution with means encoded in "means") with length
       n_bins):
15 def lnlike(means, data):
16   summa = 0.0
17   for i in range(n_bins):
18     if means[i] > 0:
19       summa += -means[i] + data[i]*m.log(means[i])
20   return summa
21
22 # Define the log posterior:
23 def lnprob(params, data):
24   lnp = lnprior(params)
25         if lnp == -np.inf:
26                 return -np.inf
27         else:
28     means_temp = list()
29     for i in range(lnM_nbins):
30       for j in range(z_nbins):
31         means_temp.append(rate(params, lnM_lower+i*dlnM, z_lower+j*
       dz))
32     means = np.multiply(means_temp, volume_obslife_factors)
33                 return lnp + lnlike(means, data)
34
35 # Set the number of "walkers" (different Markov Chains that run in
       parallel)
36 nwalkers = 100
37 # Number of runs:
38 n_runs = 600
39 # The length of the burn-in:
40 burn_in_len = 400
41
42 # Set a very crude initial guess for the parameters (used in
       generating the initial position of the walkers)
43 guess = [0.5, 0]
44
45 for i in range(n_data):
46   print("Starting run %d" % i)
47   datafile = open("data/data%d.txt" % i, 'r')
48   datafile.readline()
49   params = list()
50   for j in range(ndim):
51     params.append(float(datafile.readline().strip()))
52   rates = list()
53   samples = list()
54   for k in range(n_bins):
55     rate_str, sample_str = datafile.readline().split()
56     rates.append(float(rate_str))
57     samples.append(float(sample_str))
58   datafile.close()
59
60   # Let's initialize the walkers around our crude guess
61   p0 = np.zeros([nwalkers, ndim])
62   for j in range(len(p0)):
63     p0[j] = np.random.multivariate_normal(guess, [[0.1,0], [0,0.2]])
64     while lnprior(p0[j]) == -np.inf:
```

```
65      # This step is taken so that the walkers wouldn't be
        initialised outside the "uniform prior zone" where the posterior
        is zero
66      p0[j] = np.random.multivariate_normal(guess, [[0.1,0],
        [0,0.2]])
67
68  # Now, let's create an EnsembleSampler object and sample with it
69  sampler = emcee.EnsembleSampler(nwalkers, ndim, lnprob, args = [
        samples])
70  print "Starting MCMC"
71  pos, prob, state = sampler.run_mcmc(p0, n_runs)
72  print "Finished MCMC"
73
74  # Let's omit the first burn_in_len samples and make a flat Numpy
        array out of the rest of them
75  post_samples = sampler.chain[:, burn_in_len:, :].reshape((-1, ndim
        ))
76
77  # Now, let's find median estimates and construct the credible
        intervals for each dimension individually.
78  # Let's stick them in one big Numpy array called CI (in each row
        we have the lower end of the CI, the
79  # median and the higher end of the CI of each dimension):
80  CI = np.empty([ndim, 3])
81  for j in range(ndim):
82    CI[j] = np.percentile(post_samples[:,j], [5, 50, 95])
83
84  print("The median estimate for A is "+str(CI[0,1])+" and the 90%
        symmetric credible interval: ("+str(CI[0,0])+", "+str(CI[0,2])+"
        )")
85  print("The median estimate for alpha is "+str(CI[1,1])+" and the
        90% symmetric credible interval: ("+str(CI[1,0])+", "+str(CI
        [1,2])+")")
86
87  acceptance_frac = np.mean(sampler.acceptance_fraction)
88  print "Mean acceptance fraction: "+str(acceptance_frac)
89
90  # Let's write the results into a file:
91  datafile = open("results/%d_estimate.txt" % i, 'w+')
92  datafile.write("The next ndim lines contain the true parameters (A
        , alpha). (p019 is fixed to be "+str(p019)+")\n")
93  for k in range(ndim):
94    datafile.write(str(params[k])+"\n") # write the true parameters
        into the file
95  datafile.write("The next ndim lines contain the median estimates
        in each dimension:\n")
96  for j in range(ndim):
97    datafile.write(str(CI[j, 1])+"\n")
98  datafile.write("The next ndim lines contain the 90% symmetric
        credible intervals in each dimension (each line contains the
        interval in one dimension: the lower and higher ends separated by
        a comma):\n")
99  for j in range(ndim):
100   datafile.write(str(CI[j, 0])+", "+str(CI[j, 2])+"\n")
101 datafile.write("\n")
102 datafile.write("Number of walkers: "+str(nwalkers)+"\n")
```

```
103   datafile.write("Number of runs: "+str(n_runs)+"\n")
104   datafile.write("Burn-in length: "+str(burn_in_len)+"\n")
105   datafile.write("Mean acceptance fraction: "+str(acceptance_frac)+"
        \n")
106   datafile.close()
107
108   # Let's generate a corner plot. Note that some of the first
        iterations are not shown on the histograms
109   # (we're essentially discarding the burn-in, making it pretty long
        to play it safe)
110   fig_corner = corner.corner(post_samples, labels = [r"$A$", r"$\
        alpha$"], quantiles = [0.05, 0.5, 0.95], truths = params)
111   fig_corner.savefig("results/%d_triangle_qs.png" % i)
112   pl.close(fig_corner)
113
114   # and a version without quantiles shown on the 1D histograms:
115   fig_corner2 = corner.corner(post_samples, labels = [r"$A$", r"$\
        alpha$"], truths = params)
116   fig_corner2.savefig("results/%d_triangle.png" % i)
117   pl.close(fig_corner2)
118
119   # Let's generate the trace plots
120   trace_A = pl.figure()
121   for j in range(nwalkers):
122     pl.plot(sampler.chain[j, :, 0])
123   pl.xlabel('Step #')
124   pl.ylabel('Value')
125   pl.title("Trace plot for the parameter A")
126   trace_A.savefig("results/%d_traceplot_A.png" % i)
127   pl.close(trace_A)
128
129   trace_alpha = pl.figure()
130   for j in range(nwalkers):
131     pl.plot(sampler.chain[j, :, 1])
132   pl.xlabel('Step #')
133   pl.ylabel('Value')
134   pl.title("Trace plot for the parameter alpha")
135   trace_alpha.savefig("results/%d_traceplot_alpha.png" % i)
136   pl.close(trace_alpha)
```