Time Delay Estimation in Gravitationally Lensed Photon Stream Pairs

Michał Staniaszek
Supervisor: Peter Tiňo

March 27, 2013

Abstract

In this report, we present a system for estimating the time delay Δ between multiple realisations of a Poisson process with the underlying function $\lambda(t)$, with particular application to gravitationally lensed photon streams. We develop a linear estimator based on weighted least squares, and a kernel density estimator which we use to estimate $\lambda(t)$. We then introduce two methods for estimating the value of Δ using the function estimates, one using inter-function area, and another using probability density functions. Finally, we compare the performance of the two function estimation methods and time delay estimation methods on simulated data, and show that there is not a significant difference between the approaches.

Keywords: Nonhomogeneous Poisson process, gravitational lensing, machine learning, linear regression

Typeset in Linux Libertine using XƎLaTeX.

Contents

1 Introduction

With continued advances in computing and sensing technologies, the amount of data that can be gathered from both everyday objects and scientific experiments has increased rapidly. However, more data is not always a blessing—it must be stored and analysed for it to have any use, and this is not an easy task when one has terabytes of data to deal with. The Large Hadron Collider at CERN is one perhaps extreme example, producing on the order of five terabytes of data each second. Storing this amount of data, let along analysing it is impossible, and so multiple stages of intelligent filtering are applied, reducing the throughput to 100 gigabytes per second, and then further to around 200 megabytes per second, where it is finally stored, producing almost two CDs each second [22]. This project focuses on creating the foundations for a system to do such intelligent filtering, but in the context of astronomical data. The volume of data produced by modern telescopes, while not on the same scale as the LHC, is nonetheless overwhelming. Image sizes of one to two gigabytes are not uncommon, and deciding what data is actually relevant is not a trivial task [26]. Using intelligent filtering algorithms, it should be possible to flag up interesting-looking data for further study. While there are many areas in which such capabilities would be useful, we are particularly interested in finding candidates for images of gravitationally lensed objects. In order to do this, it is necessary to find pairs of observations of photon flux which appear to have the same underlying function. More precisely, given a set of data containing the time of arrival of photons from a particular source, henceforth called a *stream*, we wish to find another stream which, when shifted in time by some value Δ, has similar numbers of photons arriving in a given interval as the first stream. We call Δ the *delay* between the two streams. In this project, we develop a system which can generate simulated photon streams using Poisson processes, use linear regression to estimate the underlying function of a given stream, and, given the function estimates of two streams, estimate the time delay between them. Knowing the value of the time delay has many applications in astrophysics, and with more precise estimates, more accurate calculations can be made to increase our understanding of the universe we live in.

- strong lensing has delays on the order of hundreds of days, but weak lensing is more like on the order of hours - no longer sufficient to calculate flux for a single day, must do it in a different way, by measuring individual photon arrival times.

In section 2 we discuss the concepts underpinning the project in more detail, with a more in-depth explanation of the issues surrounding the calculation of the time delay and its uses. In section 5 we introduce our method of generating photon streams from Poisson processes. Section 2.3 shows our approach to estimating the underlying function of a given stream of photons. Our methods of calculating the time delays between multiple photon streams are explained in section 7. Section 3 gives detailed information on the design and development of the system, including the software and project management aspects. Finally, in section 8 we present experimental data from both simulated and real data and discuss the relative effectiveness of our methods.

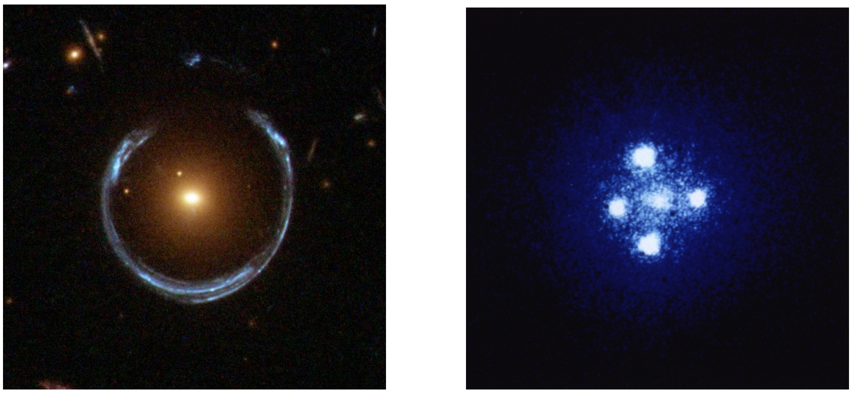
2 Background

2.1 Gravitational Lensing

In an eight-year period starting in 1907 and ending in 1915 with the publication of a paper on field equations of gravitation [8], Albert Einstein wrote many papers developing a new theory of gravitation, his general theory of relativity. This generalisation of special relativity and Newton's law of universal gravitation led to a revolution in the field of physics, and remains one of the most important scientific discoveries to date. The theory describes how spacetime is affected by the presence of matter and radiation, and this idea has many important consequences, but one of the effects in particular is important in the context of this report.

According to the theory, objects with mass, or massive objects, cause spacetime to curve around them. A simple way to visualise this effect is to imagine dropping a ball onto a sheet of cloth which has been pulled taut. The ball will eventually come to a stop in the centre of the cloth, and cause it to sag. Here, the sheet represents spacetime, and the ball represents anything from planets, to stars, or even entire galaxies. Depending on the weight of the ball, the shape of the cloth will be affected to different degrees—a ping pong ball will have hardly any effect at all, but if we drop a bowling ball onto the sheet, the effect will be significant. In a similar way, the amount that spacetime curves around a massive object depends on its mass. An object with high mass will cause a large amount of curvature, whereas a lower mass object will cause less. If a second ball, lighter than the first, is introduced to the system, what happens? With no initial velocity, it will roll in a straight line towards the first ball sitting at the centre of the sheet. This is one way of thinking about gravity and its relationship with spacetime—an object's gravitational attraction is a result of its mass curving spacetime, and the strength of the attraction is proportional to the mass. While objects with no mass, such as photons, cannot be affected by gravity directly, they *are* affected by the curvature of spacetime. This bending of light rays is known as *gravitational lensing*.

The first person to study the effects of gravitational lensing was Orest Chvolson, publishing a short note to *Astronomische Nachrichten* in 1924 [4]. However, the concept was largely unknown until a short calculation by Einstein was published in *Science* in 1936 [9]. Interestingly, Chvolson's note appears directly above a note from Einstein[1], but there appears to be no evidence that Einstein had ever seen it [23]. The first gravitationally lensed object to be identified was the twin quasar SBS 0957+561, in 1979, and since then, over a hundred such objects have been discovered [27, 12]. The effect of gravitational lensing is, as the name sug-



(a) An Einstein ring

(b) Einstein's cross

Figure 1: Two examples of strong lensing effects. a) shows light from a distant blue galaxy being distorted by the central galaxy LRG 3-757 [6]. b) shows four images of a distant quasar being lensed by a foreground galaxy [15].

gests, similar to that of a lens, such as that of a camera. Unlike a camera lens, however, gravitational lenses do not have a focal point, but instead a focal line, resulting in images such as that shown in Figure 1a if the source (the object being lensed), the lensing object (the massive object around which the light is being bent) and the observer lie on a straight line. This effect is relatively rare, however, and in general rather than a ring, multiple images of the source can be observed. In these so called *strong* lensing effects, the distortion is very clearly visible. However, two other classes of lensing exist—*weak lensing* and *microlensing*. The effects of weak lensing cannot easily be observed visually, but statistical techniques can show the distortion produced. Microlensing works on even smaller scales than the other two classes, and can be used to detect planets and stars. It has also been proposed as a method to find objects such as black holes and brown dwarfs, which are otherwise difficult to detect [25].

2.1.1 Importance of the Time Delay

In gravitationally lensed systems, there is a delay between photon streams coming from images of the source due to the bending of light. Light from one source may have had to travel a slightly longer distance than that from the other, and while photons travel extremely fast, over astronomical distances the delay can become quite large.

- strong lensing p86
- Talk generally about the problem of time delay estimation
- refer to physics papers attempting to make estimates of the delay

- talk about time delay estimation in particular, refer to kundic et al, many others
- talk about how better estimates benefit the scientific community
- refer to peter's paper about the efficacy of kernel regression
- better estimators are necessary to increase the accuracy of estimates
- this is an experiment to see whether this method has any use
- build on technique introduced in massey et al

2.2 Poisson Processes

In certain situations, there are many benefits of having good models of the numbers of events that occur in a given period. For example, being able to estimate the number of incoming requests to a server, the number of calls made to emergency services, and the rate of radioactive decay at any given time are all useful in different applications. Poisson processes are *stochastic processes* that can be used to do just that. A stochastic process is a way of representing the evolution of a random value or system over time by using collections of random variables. Most such processes do not evolve in a *deterministic* way. That is, the way they change as time passes is not predictable.

A Poisson process is one such process which counts the number of events and the time at which they occur in a given time interval, and have been used to model all of the above examples [13, 3, 2]. In their basic form, Poisson processes have the following important properties [24]:

1. $N(0) = 0$.
    - $N(t)$ represents the total number of events that occurred up until time t. Thus, if $N(0) = 0$, it follows that the process begins at $t = 0$.
2. The numbers of events occurring in disjoint time intervals are independent.
    - The *independent increment* assumption. This states that $N(t)$, the number of events that occur up to time t is *independent* of the number $N(t + s) - N(t)$, i.e. the number of events in the time interval between t and s. In other words, the number of events that occur in one interval does not have an effect on the number of events in any other time interval.
3. The probability distribution of the number of events that occur in a given interval is dependent only on the length of the interval.

    - The *stationary increment* assumption. The implication of this is that the probability distribution of $N(t + s) - N(t)$ is the same for all values of t. That is, the likelihood of a number of events n occurring in the above time interval does not change, regardless of the value of t.

4. No counted occurrences are simultaneous.

    - For all events that occur in the duration of the process, no two events will occur at the same time.

The most important thing about Poisson processes is the *rate parameter*, λ. This value represents the number of events that occur in each time interval. As we are counting events, it is clear that the rate parameter can never go below zero—there cannot be a negative number of occurrences in a given time interval. There are two types of Poisson processes, *homogeneous* and *non-homogeneous*. In a homogeneous Poisson process (HPP), the rate parameter is constant throughout the running of the process. This means that in every interval, the same number of events are likely to occur. In contrast, a non-homogeneous Poisson process (NHPP) has a rate parameter which varies. This means that the rate at which events occur varies during the running time of the process.

2.3 Function Estimation

2.3.1 Regression

Regression is a statistical technique used to fit lines or curves to data points in order to find some sort of relationship between them. The number of variables in the data is important. One of the variables is called a *dependent* variable. We want to find the relationship between this variable and the other variables, called *independent* variables. What makes one variable dependent and another independent? Consider the expression $y = f(x)$. If $f(x)$ is some function of the variable x, then we know that the value of y depends on the value of x. This is where the names come from. In this simple example, x is the independent variable, and y is the dependent variable. There can be multiple independent variables.

Linear regression is used in many different fields to find the trend between variables. It is heavily used in economics to make predictions about what happens in many economical situations. Finding trends in data is useful to many people in different ways.

- example of residuals. Show two linear estimates with the residuals sketched, one with a better fit than the other.

2.3.2 Kernel Density Estimation

This is another method which can be used to estimate functions, but which applies specifically to the probability density function of random variables. This technique uses *kernels* to estimate the function densities. A kernel is a function which has some parameters. To estimate functions, kernels are centred at certain points along the axis which is being estimated. The spread can be either at uniform intervals, each sample value, etc. Kernels may have a weight assigned to them. Varying the parameters of the kernels results in different properties of the estimate. There are many different kernels that can be used. Different kernels are used in different applications.

- Show some examples of different kernels

In this section, we present an overview of the structure of the system and the development process. More detailed notes on the implementation of a particular subsystem can be found in the section dedicated to that specific part. Here, we describe parts of the system which do not have a dedicated section, and also provide a general idea of the interactions between the parts of the system.

3 System

3.1 Design

3.2 Overall Structure

On the most basic level, the system is split up into two different modules; the estimators and the generators. These two parts of the system do not interact at all, and can be used as completely independent systems. What does connect them is a set of shared libraries, which provide mathematical functions, data structures and other non-specific utility functions. Figure 2 gives an overview of the structure of the system.

3.3 Libraries

Each part of the system makes extensive use of custom libraries. Each library consists of a header file which contains the function prototypes and include information, along with a separate file for the functions, which are compiled by `libtool` into a convenience library. The advantage of using `libtool` over other ways of constructing libraries is that it can create both shared and static libraries. This means that if the library needs to be re-used elsewhere it is simple to take the shared object file created and compile the program including the library by passing the standard `-l[libname]` syntax to `gcc`. Since the libraries contain various functions which are useful not only for this system, this is a useful feature to have as the libraries can be easily re-used. Due to some interdependencies, between the lower level convenience libraries, they are merged into one main library, again functionality provided by `libtool`. The main function of the libraries is to abstract out code which may be used multiple times in order to reduce clutter, and avoid code duplication and the increased likelihood of errors associated with it.

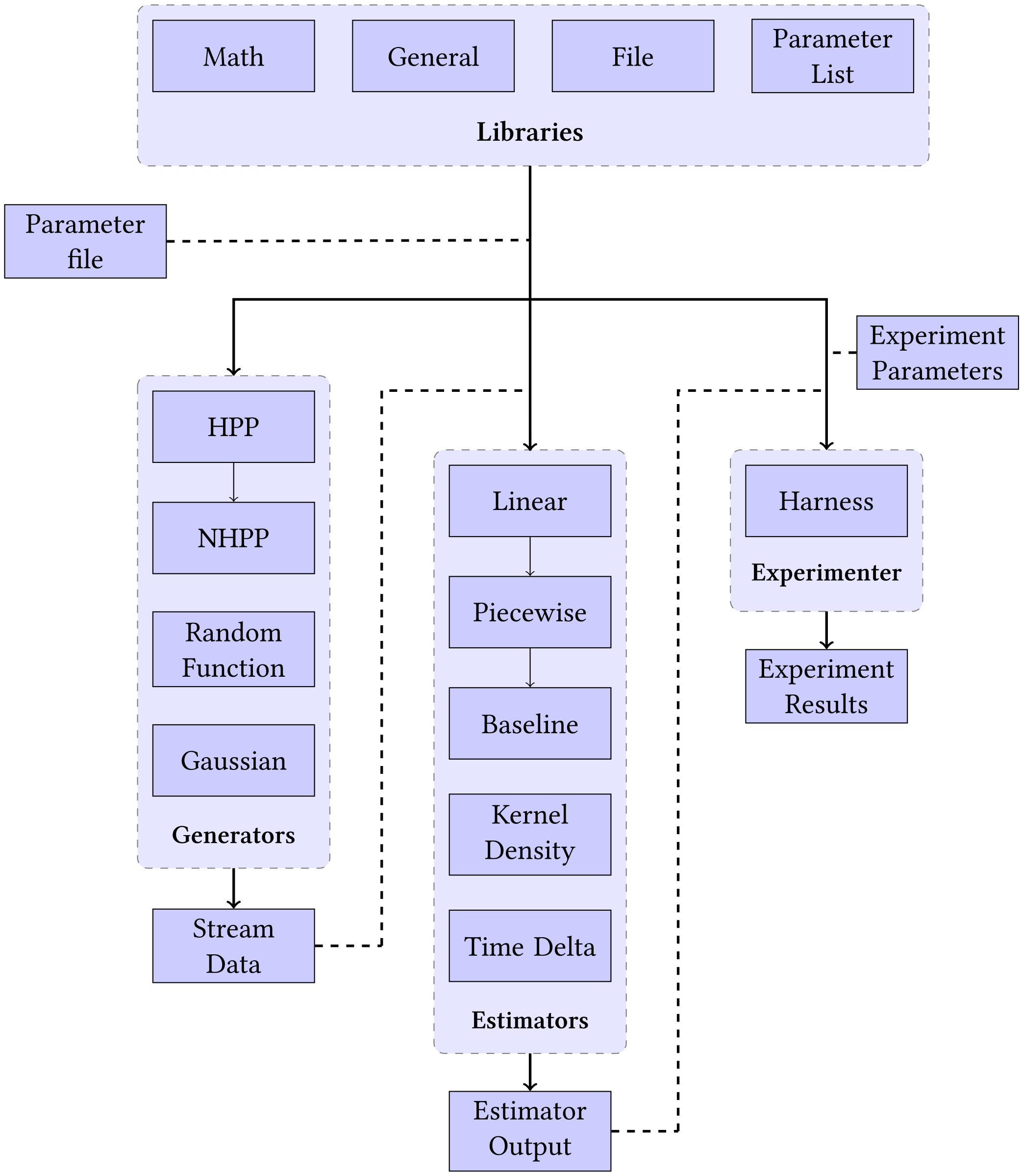


Figure 2: System structure

3.3.1 Parameter List

The parameter list library defines a singly-linked list which is used to store data parsed from the parameter files which contain parameters that modify the behaviour of the system. The functionality implemented by this library is very important, as almost all functions outside the libraries require one of these lists to perform their tasks. The library provides the functions for adding elements to the list and finding its length. A function for removal of elements is not provided, as there is no situation which should necessitate the removal of elements from the list. There is also functionality for checking whether a parameter with a given name exists, retrieving the value of a parameter, and setting the value of a parameter.

There are multiple retrieval functions, each of which retrieves values of different types. The parameter list is constructed in such a way that all values in it are stored as character arrays. This means that if a parameter value is required by some function, it must be converted into the type which that function requires. Since it is known inside the function which type is required, the relevant function can be called. Functions to read `double` and `int` types are provided, along with a function to retrieve the character array. In addition, some of the parameters in the files are comma-separated lists of integers or doubles, which must be parsed into arrays before they can be used. The parsing also applies to the `double` and `int` types, which must be converted from string form into the correct type. In order to reduce code duplication, the conversion of the variables is done inside the retrieval function.

The issue of whether the parameters should be parsed into the correct types upon reading of the file may be raised, but we believe that this solution is more appropriate. Firstly, it is often the case that not all parameters present in the file are required when any given task is executed by the system. Therefore, parsing all the parameters upon read may be a waste of processing time. In addition, having to parse parameters requires knowledge about what types they should be. This can be deduced within the code, but would require more complex logic. The storage would also be complicated by this, as the list would be required to store variables of multiple types. As mentioned above, parsing parameter values only when they are needed, and in contexts where their type is known means that the current structure performs its role well.

3.3.2 Mathematics

As the name implies, the mathematics library provides the mathematical functions required by the system which are not provided by the standard C library. Some of the library functions are based on functionality provided by the GNU Scientific Library [11], particularly those which calculate probability density functions or require random number generators. The most important part of the library is the functionality it provides for computations to do with Gaussians, in particular the discrete Gaussian transform. It also provides some basic useful functions, such as finding the minimum and maximum values in arrays, averaging, summing, adding to or multiplying arrays, and some implementations of statistical functions such as the root mean square error, standard deviation and the like. Functions which are critical to the operation of the system will be described in the implementation section of the relevant subsystems.

The most challenging part of the implementation of the library was to get around the issues caused by doubles. Functions which deal with calculations based on timings require a certain precision on the start and end times of intervals to work correctly. Due to the nature of double precision values, calculations with them often do not have the precise value of the calculation. Particular problems were encountered when incrementing a value by a floating point number and comparing it to another. The floating point increments caused the value to be slightly (on the order of 1.0×10^{-20}) below the actual value, and this caused calculations to be incorrect and resulted in a cascade of erroneous calculations. To deal with this problem, functions for comparing double precision values to a specific precision were implemented.

3.3.3 Input/Output Utilities

This library implements functionality for reading from and outputting to files, as well as for checking the state of files and directories on the system. This library is also used to parse the parameter file into the system, and as such defines the syntax that the parameter file must follow. We attempted to find a library equivalent to the Java Properties class, which allows the structured reading and storage of parameters. We were, however, unable to find such a library and therefore had to implement our own version. The syntax is very simple; the # symbol defines a comment, and that line will be ignored when parsing. A parameter is defined as an string of ASCII characters followed by a single space, followed by more ASCII characters. This library also reads in event data files, which are needed as input to the estimators, and can retrieve either all events, or data in a specific interval.

As well as reading in data, the library also serves to output data from various data structures used within the system. This ranges from simple arrays to more complex data structures used to store representations of Gaussians or function estimates.

3.3.4 General Utilities

The final library is for functions which do not fit in anywhere else, such as memory allocation and freeing, printing structs and some error checking functions. There are also functions for generating default parameter files. The functionality of this library is relatively limited, but it means that the rest of the code in the system is much cleaner, as memory allocation and freeing of relatively complex structs can be done with a single function call. This abstraction of the memory management also makes dealing with the memory much easier—it is more difficult to make a mistake when freeing something, as the functions are already defined.

3.4 Experimenter

- explanation of the various bits to do with stuttering and so on
- How data is parsed in and stuff
- how the experiment numbers are incremented

3.5 Launcher

- what thing it uses getopt
- user-friendliness (help output)
- struct usage to improve ease of modification

3.6 Scripts

3.7 Interface

Usage instructions for the system can be found in Appendix B

4 Development

4.1 Development Process

- first draft up code in notebook to get down the concept
- write a basic code skeleton and add tests to make sure that it works as intended - particularly for mathematics and the like
- flesh out the code and integrate it with the system
- make code as modular as possible to make it easy to add stuff in later

4.2 Development methodologies

- could be merged with previous section, but mention some of the unix rules of thumb, like the rule of least surprise and so on.

4.3 Testing

- talk about check, which functions were tested (mostly library functions, hard to test estimators in a reliable way)
- give examples of the tests

4.4 Version Control

- branching strategy
- commit frequency
- using issues on github
- storing backups of tags on svn

4.5 Project Management

- keep changelog
- show examples of changelogs and commit messages from the same time period
- writing up and planning layout in notebook

5 Simulation of Photon Streams

The first step in building the system was the development of a photon stream simulator. The ability to simulate photon streams means that the system can be tested on many different stream types, so that we are able to determine where its strengths and weaknesses lie. While many simulation tools are very complex, our system does not require simulation of the source objects or the movement of photons, as we are only interested in their arrival time. A source can be represented by some random variable X, which indicates the variability of the source with time. Different types of sources will have different types of characteristic functions—the variation in a quasar will be very different to that of an individual star, for example. A NHPP is an ideal way to represent this type of system. The function $\lambda(t)$ will represent the random variable, and the values output from the process will represent the arrival times of the photons. $\lambda(t)$ describes the variability of the source in time. In other words, it provides a rate parameter at each time t for the duration of the simulation. To be able to simulate a wide variety of functions, it is necessary to have the capacity to generate functions with different characteristics.

5.1 Function Generation

To evaluate the performance of the function and time delay estimators, it will be necessary to test the accuracy of the estimates on different types of functions. To this end, the capability of generating random functions will be very useful. To generate random functions, we make use of Gaussian kernels or just Gaussians?. The generation process involves four simple steps:

1. Pick some value Δt which represents the distance between the mean μ of successive Gaussians.

2. Define some value α, where the standard deviation σ of each Gaussian is determined by $\alpha \cdot \Delta t$.

3. For each Gaussian, choose some weight w_i, from a uniform distribution between -1 and 1.

4. Using some step s, sum all the Gaussians at each point on the x-axis which we get from these s values.

The first step defines how spread out the Gaussians should be in the interval $[t_0, T]$ in which the function is to be generated. If the spread is large, then depending on the standard deviation of the Gaussians there will be many points on the interval where the value of the function is zero. On the other hand, with a low value of Δt, most points on the line should have some non-zero value.

The α parameter determines the standard deviation σ of all the Gaussians used to generate the function. The value of σ is the one that affects the final function the most. Low values will result in each Gaussian covering only a small interval, so if the Gaussians are sufficiently spread out, the variation in the function will be much larger than if higher values of σ are used.

With just the above two steps, the functions generated would be very homogeneous, because each Gaussian has the same weight. With uniform Gaussians, there would be hills at each point where a Gaussian is centred, and very little to speak of in between, and the height of the function would never exceed a certain value. To introduce more variation, a weight w_i is sampled uniformly from ($w_i \sim U(-1, 1)$). Uniform sampling simply means that each value between -1 and 1 has an equal probability of being chosen. To further increase the variation in the functions that can be generated, some multiplier can be used, which scales the values of the weights, meaning that the function will have larger values over the whole interval.

The final step is to calculate the values which will make up the function. Starting at the beginning of the interval t_0, we sum the values of all the Gaussians at points along the line until the end of the interval T is reached. The points that are sampled are defined by $t_i = t_{i-1} + s$, where s is some sample step. The use of smaller sample steps results in a higher resolution. The sum of the Gaussians at time t can be calculated by

$$f(t) = \sum_{g \in G} w_g \cdot e^{-(t-\mu_g)^2/2\sigma_g^2} \tag{1}$$

Where G is the set of Gaussians which make up the function, and w_g, μ_g and σ_g are the weight, mean and standard deviation respectively of the current Gaussian being processed.

- random functions using gaussian sums
- explain how the gaussians are placed, how the standard deviation is calculated, what effect this has on the function shape ($\sigma = \alpha \cdot \Delta t$)
- examples at various alpha and delta t values?
- $w_t \cdot e^{\frac{x-\mu^2}{2\sigma^2}}$

In addition to the random function generation, it may sometimes be useful to generate a function from a known expression, and the system includes this functionality as well, which will be described below.

5.2 Generating Streams from Functions

Once the function has been generated, we can use it to generate values for the random variable X which governs a NHPP. To generate a NHPP, it is necessary to build on the generation of a HPP. It is well known that probability of an event occurring follows an exponential distribution. The rate parameter λ determines how many events occur in a given time interval. Knowing this, we can calculate the time of the next event by sampling from this distribution. Generate a random value $U \sim U(0, 1)$. The time t to the next event is defined by

$$t = \frac{1}{\lambda} \log(U) \tag{2}$$

Using this calculation, it is possible to generate a realisation of a HPP for any length of time. This provides a base which can be extended to generate events from NHPPs. To generate events from the NHPP, we use a technique called thinning. The basic concept behind thinning is to generate a large amount of values, and then remove them based on some method. In the case of the NHPP, we generate events with a rate parameter λ, where $\lambda > \lambda(t)$ for $0 <= t <= T$. In other words, the homogeneous lambda value must be larger than the value of the function we are generating from at any point. First, two random values are independently sampled from a uniform distribution between 0 and 1, $U_1, U_2 \sim U(0, 1)$. U_1 is used in (2) to find the next event time from the homogeneous process governed by λ. Using the time t generated from that, the value of $\lambda(t)$ is found. Depending on the ratio between $\lambda(t)$ and λ, the event is kept or discarded. When the value of $\lambda(t)$ is close to that of λ, more events are kept because $U_2 \leq \frac{\lambda(t)}{\lambda}$

will be true more of the time. The variation of $\lambda(t)$ in time means that events are generated proportional to the value of lambda.

- Need to generate event times - use Poisson process
- start with homogeneous
- extend homogeneous to non-homogeneous (explain math)
- Issues with the implementation - must have $\lambda > \lambda(t)$ for all $0 <= t <= T$.
- Diagram showing HPP and NHPP
- http://preshing.com/20111007/how-to-generate-random-timings-for-a-poisson-process

5.3 Implementation

The implementation of the random and expression based function generators form the first part of the *generator* sub-system. This part of the system deals with the generation of functions, and the generation of photon streams from these functions.

- gaussians as structs
- gauss vector structs
- functions into math library
- separation of functions so that they can be called externally and internally
- what can be generated
- How expr and gauss version differ

5.3.1 Generating from Expressions

- muparser
- enter expression and define variables in parameter file
- parsed in and calculated automatically

5.3.2 Generating from Random Functions

-

6 Function Estimation

6.1 Baseline Estimation

In this section, we present the baseline method for function estimation. The core of the estimator is based on the iterative weighted least squares estimator derived by Massey et.al [19], and in the next two sections we attempt to explain its derivation in simpler terms than in the original paper. The subsequent sections detail our additions to the simple estimators in order to form the final baseline estimator which will be used as the first of our function estimation methods.

6.1.1 Ordinary Least Squares

The ordinary least squares (OLS) estimator forms the core of the baseline estimator. This estimator will form an estimate by minimising the sum of squared residuals. It is important to note the difference between errors and residuals. In statistical terms, an *error* is "The difference between the observed value of an index and its "true" value" [20], and a *residual* is "The difference between the observed value of a response variable and the value predicted by some model of interest" [10]. The "true" in the definition of error is in inverted commas due to the fact that the true value of the function is unobservable—it is only possible to obtain a statistical sample. The residual, on the other hand, is the difference of the observation from some *estimate* of the function. This first estimator estimates a linear function of the form $y = ax + b$, or a straight line. While this is not directly useful for estimating characteristic functions, it was developed in order to gain a deeper understanding of the ideas behind regression, and in order to construct a simple estimator which could then be extended.

In order to estimate the function, the stream of events must first be converted into a form which is suitable for processing. To do this, we first pick a time interval $(0, T]$, and divide it into N sub-intervals, or *bins*. According to [19], the kth bin I_k is calculated by

$$I_k = \left(\frac{(k-1)T}{N}, \frac{kT}{N} \right], \; 1 \leq k \leq N \tag{3}$$

and the midpoint x_k of each bin is

$$x_k = \left(k - \frac{1}{2} \right) \frac{T}{N}, \; 1 \leq k \leq N \tag{4}$$

Due to the independent increments property of Poisson processes, splitting the interval leaves us with N bins, each of is defined by an independent

Poisson random variable [19] Y_k with mean

$$\lambda_k = \frac{T}{N}(a + bx_k) \tag{5}$$

where T/N is used to normalise the value of λ_k. The value of Y_k in our case is the number of photon arrival times for each bin. In order to perform regression on the data, we need a model of the data. At this stage, we make the assumption that the data is represented by a linear function, so the model is linear. The model is used to connect the random variables and the parameters, and describes how they are related. Our model becomes $Y = \alpha + \beta x + \epsilon$, or $Y_k = \alpha + \beta x_k + \epsilon_k$ [19]. The values α and β are the two regression parameters which we use to estimate the values of a and b of the characteristic function. What is a regression parameter? ϵ represents the Poisson error that is present in the data that we are trying to model. As mentioned before, this technique works by minimising the sum of squared residuals. The value of a residual can be computed by [16]

$$d_k = Y_k - (a + bx_k) \tag{6}$$

However, since we cannot know the real values of a and b, we must instead use the regression parameters α and β. Substituting these into (6) we get

$$d_k = Y_k - (\alpha + \beta x_k) \tag{7}$$

With this, we can calculate residuals for our function estimate. This calculation by itself is not sufficient, though, as summing the residuals necessarily results in a value of zero **FIND A CITATION FOR THIS!**. To get a useful value from the residuals, we square the value of each residual.

$$d_k^2 = (Y_k - [\alpha + \beta x_k])^2 \tag{8}$$

Until now, we have been ignoring ϵ_k, the Poisson noise associated with the random variable. In order to compensate for this, it is necessary to introduce a weight w_k for each interval, initialised to 1 as we are using the OLS technique [19]. Introducing this weight into (8) and summing over all bins, we have

$$\sum_{k=1}^{N} w_k(Y_k - [\alpha + \beta x_k])^2 \tag{9}$$

the residual sum of squares (RSS). We want to find the values of α and β for which the RSS is minimised, and so the final expression becomes

$$\min_{\alpha,\beta} \sum_{k=1}^{N} w_k(Y_k - [\alpha + \beta x_k])^2 \tag{10}$$

which is known as the *objective function*. Once the objective function is known, we can define estimators $\hat{\alpha}$ and $\hat{\beta}$, which we will use to estimate values of α and β to find the minimum [19].

$$\hat{\beta} = \frac{\sum_{k=1}^{N} w_k(x_k - \bar{x})(Y_k - \bar{Y})}{\sum_{k=1}^{N} w_k(x_k - \bar{x})^2} = \frac{\sum_{k=1}^{N} w_k(x_k - \bar{x})Y_k}{\sum_{k=1}^{N} w_k(x_k - \bar{x})^2} \tag{11}$$

$$\hat{\alpha} = \bar{Y} - \hat{\beta}\bar{x} \tag{12}$$

$$\text{where} \quad \bar{x} = \frac{1}{N}\sum_{k=1}^{N} w_k x_k \quad \text{and} \quad \bar{Y} = \frac{1}{N}\sum_{k=1}^{N} w_k Y_k \tag{13}$$

If we ignore the effect of adding the weight w_k for the time being, the calculation of $\hat{\beta}$ is equivalent to dividing the covariance of x with Y by the variance of x [17]. This should allow us to understand more about how the value of the estimate changes depending on what the values of x and Y look like. The covariance is a measure of the strength of the correlation between two or more random variables [28]. If high values of one variable occur when the other variable also has high values, then the covariance is positive. If high values of one variable occur when the other has low values, then it is negative. The variance, on the other hand, is a measure of the variation in values of a random variable. If all values are close to the mean, then the variance is small, and if there are large deviations from the mean value, then the variance is large. If the covariance is positive, then the values of Y increase as x increases. The variance of x depends only on the length of the interval—short intervals have low variance, and long intervals high variance. This is because the calculation of the variance is done by finding the distance to the midpoints of bins from the value of $\bar{x}$, which is the midpoint of the interval. It is clear that the sign of $\hat{\beta}$ depends on whether the covariance is positive or negative, and this in turn defines the sign of the gradient. The steepness of the gradient is defined by the magnitude of the covariance. Since the value of the variance is constant, the larger the magnitude of the covariance, the steeper the gradient. Once we know the gradient of the line, the calculation of the intercept is simple, so long as we know the value of a point on the line, and the point $(\bar{x}, \bar{Y})$ is one such point. We rearrange the equation $\bar{Y} = \hat{\alpha} + \hat{\beta}\bar{x}$ to solve for $\hat{\alpha}$. Notice that since the values of $\bar{x}$ and $\bar{Y}$ do not change, the point defined by the mean values becomes a pivot for the line estimate. The addition of the weights adds bias into the calculation, taking into consideration the variation of those points which have a smaller estimated value of λ. The weight update calculation is discussed in the next section.

We normalise the values of $\hat{\alpha}$ and $\hat{\beta}$ by multiplying the resulting estimate by the number of bins over the interval length. The fewer bins used in the estimate, the larger the bin count will be for each bin, and consequently the larger the estimated values will be. To return the estimate to the same scale,

$$\hat{a} = \frac{N}{T}\hat{\alpha} \quad \text{and} \quad \hat{b} = \frac{N}{T}\hat{\beta} \tag{14}$$

- show non-normalised estimate vs normalised estimate

As we are dealing with a Poisson process with a rate function, it is natural to impose a constraint on the values of $\hat{a}$ and $\hat{b}$ which states that the rate function must be non-negative throughout the entire interval $[0, T]$, since it is not possible to have a negative rate [19].

$$\hat{a} \geq 0 \quad \text{and} \quad \hat{b} \geq -\hat{a}/T \tag{15}$$

There are two cases in which this constraint can be violated; when $a < 0$ or $b < -\hat{a}/T$ [19]. In the first case, we set

$$\begin{aligned} \hat{a} &= 0 \\ \hat{b} &= \frac{N}{T}\frac{\sum_{k=1}^{N} w_k x_k Y_k}{\sum_{k=1}^{N} w_k x_k^2} \end{aligned} \tag{16}$$

and in the second,

$$\begin{aligned} \hat{a} &= -\hat{b}T \\ \hat{b} &= -\frac{N}{T}\frac{\sum_{k=1}^{N} (T - x_k) Y_k}{\sum_{k=1}^{N} w_k (T - x_k)^2} \end{aligned} \tag{17}$$

Is this sufficient? Applying these adjustments in the cases in which the constraints are violated ensures that the final estimate is always within the required constraints. However, the adjustments introduce some non-linearity into the system [19]. With this set of equations, the fundamental structure of the OLS estimator is complete. In the next section, we discuss the addition of weight update rules and finding estimates of λ.

- Example of residuals
- Example function estimate on a linear function, use Poisson to generate
- Talk about this being the first step, both to learn about regression and to get a simple estimator working before moving on to more complex ones

6.1.2 Iterative Weighted Least Squares

The iterative weighted least squares (IWLS) estimator builds upon the OLS estimator. As the name suggests, the extension is to include an iterative part. The OLS estimator performs a single estimate of the function and leaves it at that. The IWLS estimator, on the other hand, repeats the process multiple times, updating its estimates. Perhaps the most important update to the estimator is the use of unequal weights, which change depending on the variances of the random variable which defines the bin which the weight is being applied to. A Poisson random variable has a variance that is equal to its mean—this means that a higher value of λ_k results in a larger variance. To compensate for this, we give higher weights to bins which have lower values of λ, as the variances will be lower. As shown in equation (5), the value of λ is easy to calculate, but the values of a and b must be known. In order to modify weights appropriately, we must be able to obtain estimates of λ, which can be done using [19]

$$\hat{\lambda}_k = \frac{T}{N}(\hat{a} + \hat{b}x_k) \tag{18}$$

The weights can then be updated using

$$\hat{w}_k = \frac{\frac{N}{\hat{\lambda}_k}}{\sum_{k-1}^{N}\left(\frac{1}{\hat{\lambda}_k}\right)} \tag{19}$$

which has some desirable properties [19]. **Minimum variance estimator among linear functions of the observations Y_k that are unbiased**. Each iteration of the estimator updates these estimates of λ and the weight for each bin, and the process is stopped when the change in the estimates becomes negligible, which consistently happens in between two and five iterations [19]. With this estimator, we have something which can improve upon the estimates from OLS with only a small amount of additional calculation. However, for our purposes this is not sufficient. The characteristic function of stellar objects are not linear functions, so we must extend this linear approach to give us some reasonable estimates of functions which are not straight lines.

6.1.3 Piecewise Iterative Weighted Least Squares

It is clear that the IWLS estimator alone is not sufficient to complete our task. In order to have a reasonable estimate of the characteristic function, we need to be able to estimate a function which is not a straight line. During the development process, we considered the possibility of approximating a function by multiple straight-line estimates. This type of function is known as a piecewise linear function. Extending the approach presented in the previous two sections, we take the interval $[0, T]$, and split it into several sub-intervals. Then, the function underlying each of these sub-intervals is estimated using IWLS. We also add some minor extensions in an attempt to improve the quality of the estimates. Sub-intervals are estimated starting from the first, and moving to the next once the process is complete. However, since the number of sub-intervals that the interval is split into is somewhat arbitrary, we implemented an estimate extension strategy. When the estimate is completed, a short interval after the sub-interval being estimated is checked to see how well the estimate matches it. The check is performed using probability density functions (PDF). The extension interval is split into several bins, and the likelihood of obtaining the bin counts of those bins given the function estimate is checked. We use a simple threshold function which only permits the extension of the estimate if for each bin the PDF calculated does not fall below a certain value. The calculation of PDFs depends on the type of probability distribution being used to check the data. In our case, this is a Poisson distribution, and we use

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!} \tag{20}$$

to calculate the probability of getting a value k for the bin count with a rate parameter $lambda$. While this technique is an improvement on using straight lines to estimate functions which are curves, it is still not sufficient, as the resulting function estimate is piecewise disjoint—the estimate for each interval does not connect smoothly into the next. There are jumps between intervals.

- explain intuition behind the technique. Split the whole interval into some finite number of sub-intervals and estimate the function of each interval in turn using IWLS.

- give reasoning behind moving to this technique. Some parts of functions look like they are pretty much linear - maybe it is a nice way to solve them. mention that this was developed on my own interest in seeing how it worked

- Explain the not-so-good parts - each subsection estimate is disjoint from the next, but the stream must be a continuous function.
- Talk about line extension and the minimum length issue

Initially, we thought that it may be possible to decide whether to extend the line or not based on the difference in slope between the estimate from the previous time interval and the estimate of the next. If the previous estimate was positive, and the next negative, and vice versa, clearly the line should not be continued. The intercept parameter was considered to be much less important. However, this assumption was highly flawed. Due to the nature of poisson processes, it is perfectly possible that although the function has changed significantly after the end of the previous interval, the estimate for the interval that we are trying to extend the line into could return very similar values to that of the previous interval. Because of this, we extend the line when we should not be doing so. There are several potential solutions to this problem. First, rather than forming a new estimate, we extend the line and then check how much the error has increased. If it goes over a certain threshold, then we reject the extension attempt and try again, this time with a shorter extension. Another potential way of improving the piecewise estimation in general would be to require the estimate for the next time period to start from the end point of the last time period. This would prevent the intercept parameter from changing, and would force the estimator to find the best estimate given a specific starting point, rather than giving it free reign to find the estimate which actually minimises the error.

- coding issues
  what to do with the issue of minimum length of intervals? Sometimes not extending the original gives a better estimate of the line than re-estimating the interval extended, or adding the short interval onto the end of the previous one and using its estimate. See data in the $\text{min}_{\text{intervallength}}$ folder in data. The better fitting line is the baseline estimate of that with no minimum, and the other set is the estimate with minimum interval length applied

6.1.4 Baseline

In the previous section, we introduced a piecewise method for function estimation. In this section we present the final modification to that estimator which completes the baseline estimator. As mentioned, the piecewise IWLS estimator gives us a piecewise disjoint estimate of the function, but we would like one which is piecewise continuous. In order to do this, the end of each interval estimate must meet the start of the next. To do so, we calculate the midpoint between the start and end of the estimates at each breakpoint, and then modify the estimates to make the functions meet at that point. This leaves us with a continuous function that forms our estimate of the function.

- improvement on the piecewise method by making sure that the function is continuous, i.e. the start of the function at each interval is the end of the one in the previous
- How we calculate the points at which to join the functions - do it at the midpoint on y between the start and end
- show a baseline and piecewise estimate on the same function

6.2 Kernel Density Estimation

For comparison to the baseline estimator, a kernel density estimator was also implemented. A kernel density estimator works by estimating the function using multiple functions called kernels. We use a gaussian kernel

$$K(\mu, \sigma) = e^{-(t-\mu_g)^2/2\sigma_g^2} \tag{21}$$

to estimate the function. There are methods which involve centering the kernels at uniform intervals along the x-axis, but for our purposes centering kernels at each arrival time is the best technique [5]. Once a kernel has been centred on each arrival time, the values of the kernels are summed at given points along the x-axis to form a function estimate. However, this is not the final step in the process. Depending on the standard deviation of the kernels used, the function estimate produced will not match the actual function. As with the normalisation constants used in the OLS estimator, we must divide the resulting values by some normalisation constant. The calculation for the kernel density estimate is slightly more complex than that of the OLS estimator. The normalisation constant is estimated by using the Poisson PDF. A range of possible normalisation constants is checked, and the one chosen is the point at which

$$\sum_{n=1}^{N} \frac{\lambda(t)^k e^{-\lambda(t)}}{k!} \tag{22}$$

is minimised. Dividing through all the values by the normalisation constant gives us the normalised estimate of the function.

- explain how kernels are calculated, and how they are centred

- explain the effect of the standard deviation on the estimate
- needs normalisation to get the proper estimate
- how the normalisation constant is found, using pmf stuff

6.3 Implementation

7 Time Delay Estimation

Once we are able to estimate the characteristic function of photon streams, we can use these estimates to attempt to provide an estimate of the time delay between two streams. If the two streams come from the same source, then they should have the same characteristic function. Our estimates of the characteristic function will differ for both streams due to the fact that the number of photon arrivals will be different for each bin, but each should look relatively similar. In this section we present two methods for estimating the time delay between a pair of photon streams.

- basic explanation of what we want to do with this part of the system, referring to the introduction a little?

7.1 Area Method

The first of the two methods uses a very simple metric to estimate the time delay. By taking the two function estimates, we can attempt to match up the two functions so that they "fit together" best. This goodness of fit can be determined by the area between the two functions. The point at which the area between the two is lowest is the natural point at which the two functions should match. Using the first estimate as a base, with its time delay set to zero, we guess at values of Δ, and shift the second function by that value. Then, we calculate the inter-function area using

$$\begin{aligned} d(\hat{\lambda}_1, \hat{\lambda}_2) &= \int (\hat{\lambda}_1(t) - \hat{\lambda}_2(t))^2 \, dt \\ &\approx \frac{1}{N} \sum_{i=1}^{N} (\hat{\lambda}_1(t) - \hat{\lambda}_2(t))^2 \end{aligned} \tag{23}$$

We make our estimate of Δ where the value of $d(\hat{\lambda}_1, \hat{\lambda}_2)$ is minimised. You will notice that to calculate the area between curves we must use an integral. In our case, an exact calculation is not necessary, and so we approximate it by a discrete calculation instead, which gives us a value which is good enough for our purposes.

- use the area of the space between two functions, find the time delay which minimises the value
- show integral formula and then show the simplified discrete formula

7.2 Probability Density Function Method

The second method of estimation is using probability density functions. As before, we guess a value of Δ between $-\Delta_{\text{max}}$ and $+\Delta_{\text{max}}$ and shift the second stream by that amount. However, we know that there must be a single characteristic function, and we want to see how well our estimate of that matches the bin counts in each stream. From the two stream estimates we have, $\hat{\lambda}_1$ and $\hat{\lambda}_2$ (which is shifted by Δ), we make an "average" function $\bar{\lambda}$ by combining the two.

$$\bar{\lambda}(t) = \frac{\hat{\lambda}_1(t) + \hat{\lambda}_2(t + \Delta)}{2} \tag{24}$$

The point on $\bar{\lambda}$ at time t is the midpoint between the values of the two estimates at that time. Once we have $\bar{\lambda}$, we can assign some score to the current estimate of the value of Δ.

$$\begin{aligned}\log P(S_A, S_B \mid \bar{\lambda}(t)) = \sum_{t=\Delta_{\text{max}}}^{T-\Delta_{\text{max}}} & \log P(S_A(t) \mid \bar{\lambda}(t)) \\ & + \log P(S_B(t + \Delta) \mid \bar{\lambda}(t))\end{aligned} \tag{25}$$

Here, we calculate the probability that the function $\bar{\lambda}$ is the characteristic function of the two streams S_A and S_B. The streams are split into bins, and the probability of the number of events in each bin given the value of λ calculated for that bin is computed.

Perhaps this should go in the previous section It is important to note that the value of Δ_{max} defines the interval in which the probabilities are summed. The need for calculation only in some specific interval should be clear—if one function is shifted, and both functions have the same time interval, then there will be an interval of Δ on either end of the range in which only one of the functions has a value. As such, the functions are combined only in the interval in which both functions have values. In addition to this, since the value of Δ changes, the intervals in which there is an overlap between the two functions changes. Setting $\Delta = 0$ minimises the value, and $\Delta = \pm\Delta_{\text{max}}$ maximises it. To be able to compare the scores of different values of Δ, we must perform calculations on the interval in which the two functions have values for all possible values of Δ. If the calculations were to be performed on different intervals or interval lengths each time, it would be necessary to scale the scores for the longer intervals to the shorter intervals, and this scaling would likely not result in an accurate representation of the actual score. Imposing this constraint on the intervals we can work with has an additional effect; the value of Δ_{max} can never exceed the interval length T in which we are performing the estimate. We are left with the constraints $T_{\text{est}} = [t_0 + \Delta_{\text{max}}, T - \Delta_{\text{max}}]$, $\Delta_{\text{max}} < T$ on the interval and the maximum value of Δ.

The calculation of λ is slightly more complicated than just taking the value of lambda at the midpoint. Since we are considering a number of events occurring in a given time span, we must consider the value of lambda in that entire time interval. In order to do this, we integrate the value of lambda over the interval

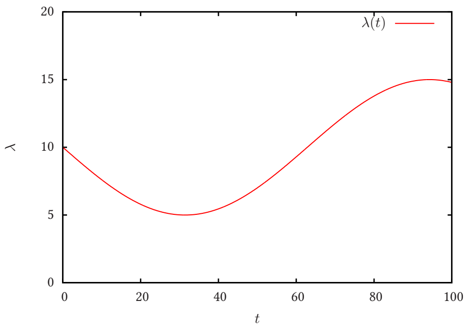
$$\lambda_{a,b} = \int_a^b \lambda(t)\, dt \tag{26}$$

However, as with the calculation of the area between curves, we do not need an exact value, only a good approximation, and so we use a discrete version of this equation where the value of t is incremented by some finite step for each successive value. The smaller the value of the step the more accurate the approximation of $\lambda_{a,b}$ becomes. As with the previous estimator, the estimate is made in two stages, first with a coarse pass over the values of delta to compute an initial estimate, and then a finer second pass around the first estimated value in order to refine the estimate.

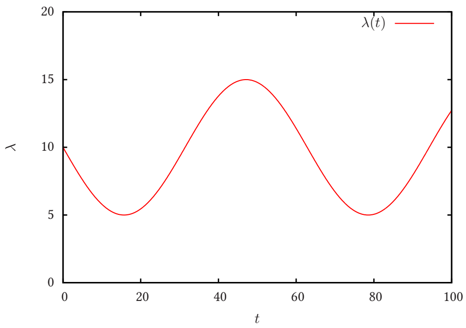
8 Evaluation

The experiments done have two stages. First, the optimum parameter set for each function that is being experimented on is found using model selection. Model selection involves withholding some of the data from the estimator so that we can see how well a parameter set generalises if not all the data is available. Withholding data is done by removing the event data from intervals uniformly distributed across the interval that is being estimated. Each function is estimated, and the value of the function in the regions where data was removed is compared to the value that would be expected had all the data been present. This is done using log probabilities, taking the Poisson PDF at each point. The sum of these log probabilities gives the parameter set its score for that function. The optimum parameter set for that function is the set which maximises the sum.

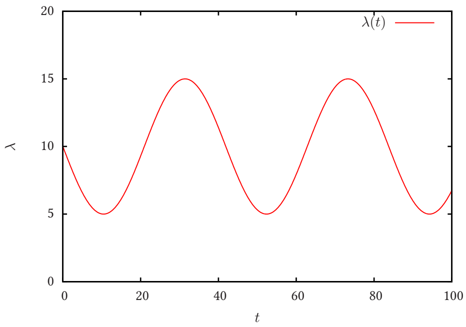
Once the optimum parameter set has been found, the time delay for the pair of streams is estimated, using all the data that is available. From this we receive estimates of the time delay on which it is possible to perform statistical analysis. The mean, standard deviation and error for each



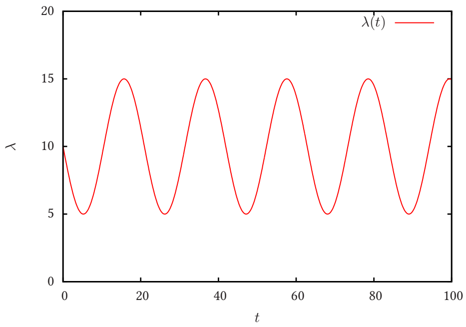
(a) $\alpha = 0.005$



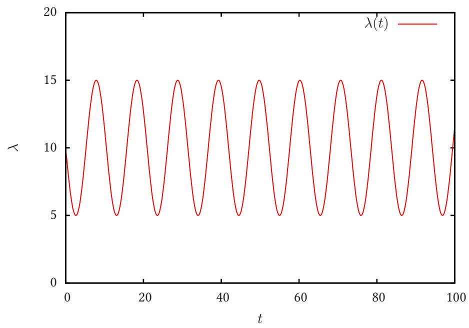
(b) $\alpha = 0.01$



(c) $\alpha = 0.015$



(d) $\alpha = 0.03$



(e) $\alpha = 0.06$

Figure 3: Functions used for preliminary experiments on sine functions, showing the different α values used.

estimate on each function is calculated, and from this we can examine the effectiveness of the estimates. The aim of the experimentation is to compare the effectiveness of the time delay estimation with four combinations of estimators: gaussian area, gaussian pdf, baseline area and baseline pdf. For the full set of experimental data, see Appendix C.

We assume that the distribution of the samples is Gaussian, but this may not be the case. However, full non-parametric testing is out of the scope of this project.

8.1 Sine Functions

The first experiment performed was on sinusoidal functions of the form $y = a - b\sin(\alpha t)$. An increase in the value of α increases the oscillation frequency, and a decrease reduces it. The value of a indicates how much the wave is shifted along the y-axis, and b determines the amplitude of the wave.

8.1.1 Preliminary Experiments

In the first set of experiments, we investigate the performance of the estimators on five values of α: 0.05, 0.1, 0.15, 0.3 and 0.6. Figure 3 gives an indication of what the functions look like. For each value of α, 25 pairs of streams were independently generated, each with an interval of 100 time units and a time delay of 10 time steps between the two streams. The Gaussian estimator was set to sample the kernels at a resolution of 0.3 time steps, and the standard deviation of the kernels was varied. The baseline estimator was set to use 3 iterations of the IWLS estimator, and four other parameters were experimented on.

IWLS sub-intervals 2, 4, 6, 8, 10

PDF threshold 0.01 to 0.15 with a step of 0.01

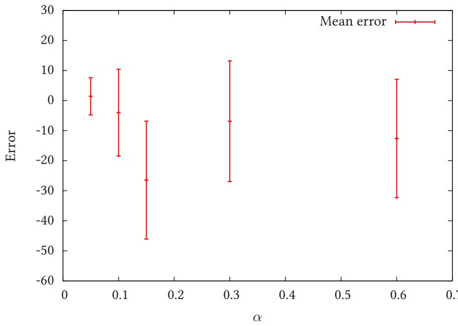
Maximum extension 5, 7, 9, 11, 13, 15, 17, 19, 20

Maximum breakpoints As above

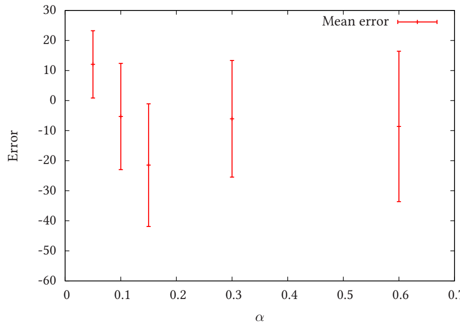
Gaussian standard deviation 0.5 to 20 with a step of 0.5

The parameters were co-varied, meaning that each value for one of the parameter settings was tested with all possible values of the other parameters, for a total of 6115 possible combinations, resulting in 152,875 runs of the function estimator for each value of α.

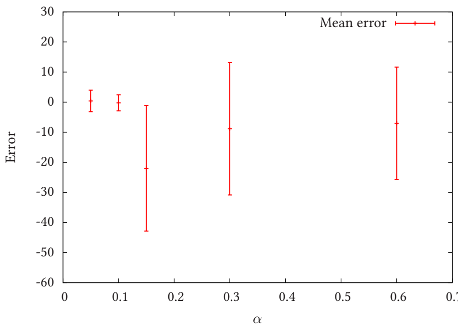
Estimates appear to be reasonably accurate until α exceeds 0.1, after which errors become much greater, and standard deviation increases. The area time delay estimator is significantly better than the PDF for both of the function estimators, with p-values of 0.00017 and 0.0000074 for the baseline and Gaussian method respectively at $\alpha = 0.05$. The difference between



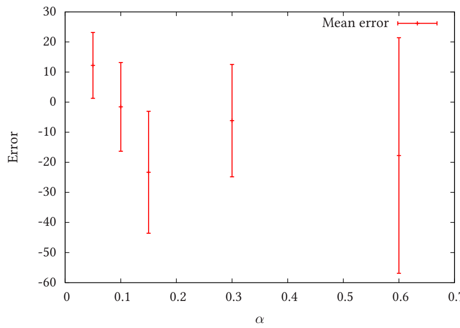
(a) Baseline area



(b) Baseline PDF



(c) Gaussian area



(d) Gaussian PDF

Figure 4: Error on the preliminary experiments. Error bars show standard deviation of error. Performance appears to deteriorate when $\alpha > 0.1$.

the two function estimation methods was not significant, with p-values in excess of 0.4 for comparisons between the baseline and Gaussian estimators for the same time delay estimators at $\alpha = 0.05$. Results from $\alpha > 0.005$ show no statistical significance in the difference between the various estimators, so although the p-values at α =0.05 are significant, they are not sufficient to say that the area estimator is always better. Figure 4 shows the error of the various estimator combination at each value of alpha.

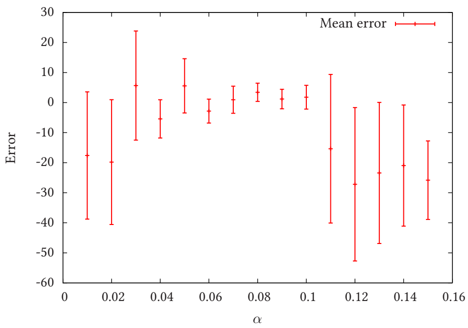
	Gaussian	Baseline
Area	10.39 ± 3.60	11.43 ± 6.18
PDF	22.20 ± 10.94	22.06 ± 11.20

Table 1: Experimental results for $\alpha = 0.05$. Actual time delay is 10. ($\mu \pm \sigma$)

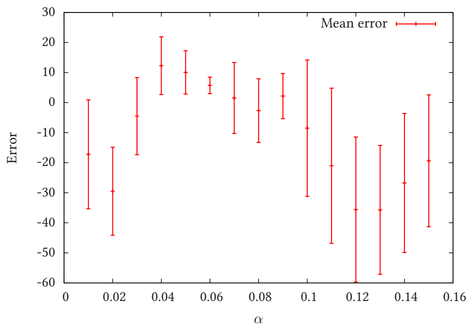
8.1.2 Refined experiments

Although the previous set of experiments provide some indication as to the performance of the estimators, we investigated their effectiveness on a smaller range of α values. In this set of experiments, we used the same parameters, but generated a new set of functions for value of α from 0.01 to 0.15, with a step of 0.01 between each successive set of stream pairs. For each value of α, 10 pairs of streams were generated. The time delay was set to 15 time steps, and the experiments were run with the same set of experimental parameters as the previous experiments.

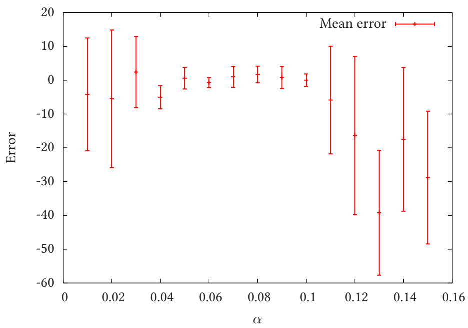
The result of this second set of experiments uncovered an interesting pattern in the performance of the estimators. Figure 5 shows the error for each combination of estimators for different values of α. It is clear to see from the graphs that there is a window of optimum performance where α is between 0.04 and 0.1. As with the previous set of experiments, the area estimator again outperforms the PDF estimator, which is visible in the graphs. Within this window, the area method is significantly better than the PDF estimator in some cases, but this significance varies greatly as α varies, and we therefore cannot conclude that there is a definite increase in accuracy using the area method. As before, the Gaussian and baseline methods do not differ significantly in performance, but on average the Gaussian method performs slightly better, having smaller standard deviations than the area method.



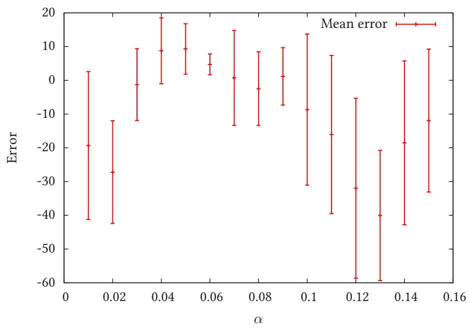
(a) Baseline area



(b) Baseline PDF



(c) Gaussian area



(d) Gaussian PDF

Figure 5: Error on the second set of experiments. Error bars show standard deviation of error. Peak performance is in the window $0.04 \leq \alpha \leq 0.1$

8.2 Random Functions

The experiments on sine functions have not yielded any definitive result as to which methods are more effective, and so we also performed a series of experiments using random functions rather than sine curves. Evaluating the performance of the estimator on these functions is important, since functions from real lensed objects will be very unlikely to follow a perfect sine curve, instead fluctuating somewhat randomly. In order to test a variety of different functions, vary the α parameter in the equation $\sigma = \alpha \cdot \Delta t$, where σ is the standard deviation of the Gaussians used to generate the random function. The weight of each Gaussian was set to 3, to give a larger range of shapes that the function could take on.

8.2.1 Preliminary Experiment

For the preliminary experiment, we chose to use five different values of α, 0.4, 0.8, 1, 2 and 3. While increasing the α parameter in the previous set of experiments would make the functions more difficult to estimate, in this case the opposite is true; larger values are easier to estimate, whereas smaller values are more difficult. This is due to the relationship of α and the standard deviation of the Gaussians used to generate the functions. For the preliminary experiments we set the value of Δt to be 10, resulting in 11 Gaussians being spread uniformly across the 100 time unit interval. Given that α ranges from 0.4 to 3, the value of σ will be between 4 and 30 time units. Lower values of σ result in each Gaussian being spread over a smaller interval, which in turn means that when the Gaussians are summed to construct the function it will have more variation than with large values. We generated 5 different functions for each value of α, and from each of these generated 5 pairs of photon streams. In these initial experiments, we wish to discover where the point of deterioration is, so that we can look at the region close to this in more detail in a subsequent set of experiments. We use the same set of experimental parameters as in the previous set of experiments:

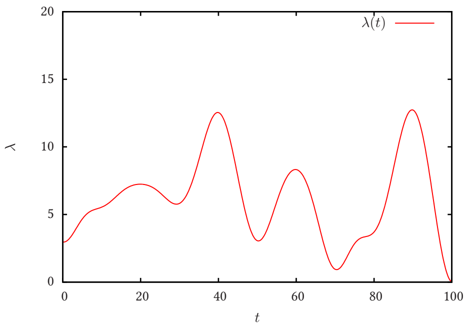
IWLS sub-intervals 2, 4, 6, 8, 10

PDF threshold 0.01 to 0.15 with a step of 0.01

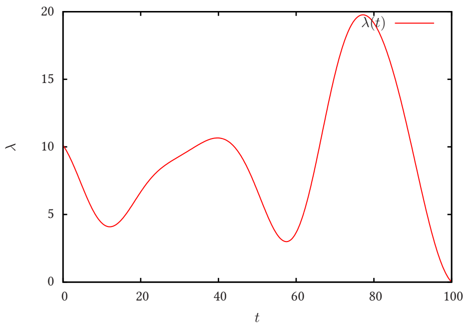
Maximum extension 5, 7, 9, 11, 13, 15, 17, 19, 20

Maximum breakpoints As above

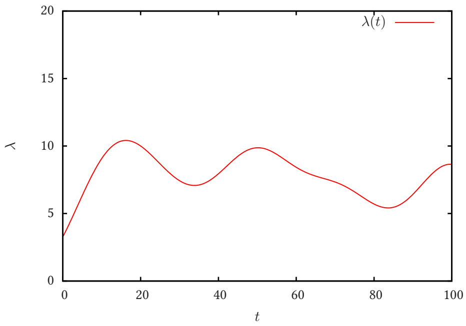
Gaussian standard deviation 0.5 to 20 with a step of 0.5



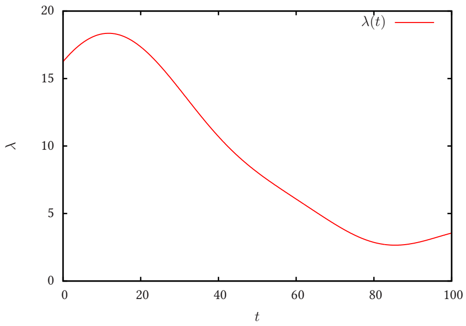
(a) $\alpha = 0.4$



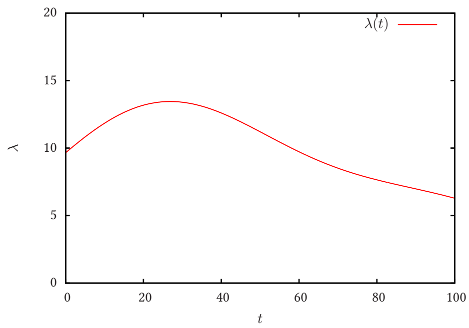
(b) $\alpha = 0.8$



(c) $\alpha = 1$

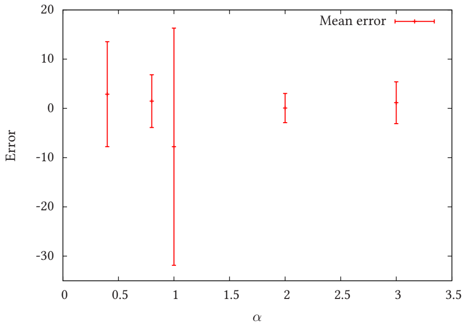


(d) $\alpha = 2$

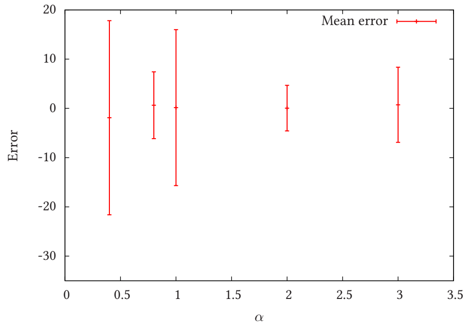


(e) $\alpha = 3$

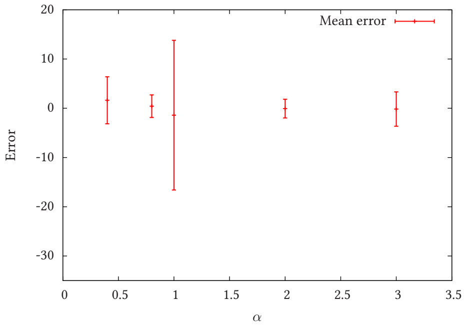
Figure 6: Examples of random functions generated by different values of α. Oscillation of the functions decreases as α increases.



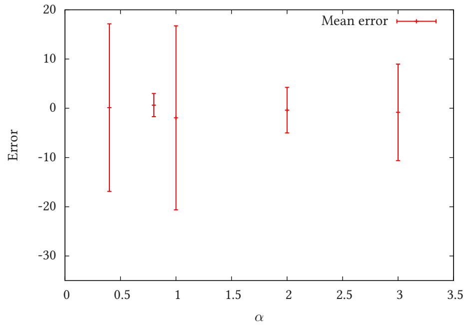
(a) Baseline area



(b) Baseline PDF



(c) Gaussian area



(d) Gaussian PDF

Figure 7: Mean error for each value of α for the preliminary random function experiments for each method combination.

The results from the experiment were very enlightening. Figure 7 shows the error of the estimators over all α values. The estimators performed with much smaller error values on average, leading us to believe that the large errors in the previous experiments were due to the shape of the functions. The methods that we use seem to be ineffective on functions which have a symmetrical shape, which sine functions are. The estimators appear to be much more stable, with the mean error deviating relatively little from zero, in comparison to the wild variation in the sine function experiments. While the performance of the estimators was better, the difference between method combinations is still not significant. The large error at α is 1 is due to very large errors occurring in estimates of two functions in that data set. This indicates that while on average the estimators perform well, on functions with certain characteristics there are large differences in the performance. Both time delay estimation methods perform worse when α is 0.4, but the estimate from the area method is clearly less affected.

8.2.2 Second Experiment

In order to investigate the estimator performance further, we performed an additional experiment on a finer set of data, varying α from 0.1 to 1.5, with steps of 0.1. Going down to such a low value of α results in functions which have very large variations, with impulse-like peaks and troughs, an example of which can be seen in Figure 8. The parameter ranges used were the same as in the previous experiment on random functions.

This experiment confirms our observation from the previous experiment that the Gaussian area method combination is the one which should be used to get the best estimates with the smallest errors, which is clear to see in Figure 9. Again, there was no pattern in the p-values that could be said to indicate that one method is significantly better than another, so we can not conclude with certainty that the Gaussian area method is indeed better than the others. However, this and previous experiments have shown that the size of the error from estimates with that combination is in the vast majority of cases smaller than that of other combinations. Errors appearing in combinations using the PDF method increase as values of α drop below 0.4, indicating that the method is more error-prone when functions have large variations. The PDF method in general has larger standard deviations on the error than the area method.

9 Conclusion

In this report, we have presented our system for estimating the time delay in gravitationally lensed photon stream pairs. We showed two methods for estimating the characteristic function of the stream; the baseline method, which is build upon the iterative weighted least squares method, and the Gaussian kernel density estimation method. In addition, we presented two methods for time delay estimation, one using inter-function area, and another using probability density functions.

We performed two experiments on sine functions with different oscillation frequencies. The first showed that there appeared to be a point of deterioration at which the estimators' performance experienced a large decrease. In the second, we investigated the performance on a finer level, and noted that there was a window in which estimators performed well, whereas in other areas there were large errors. From the first to experiments, the area method for time delay estimation appeared to be slightly

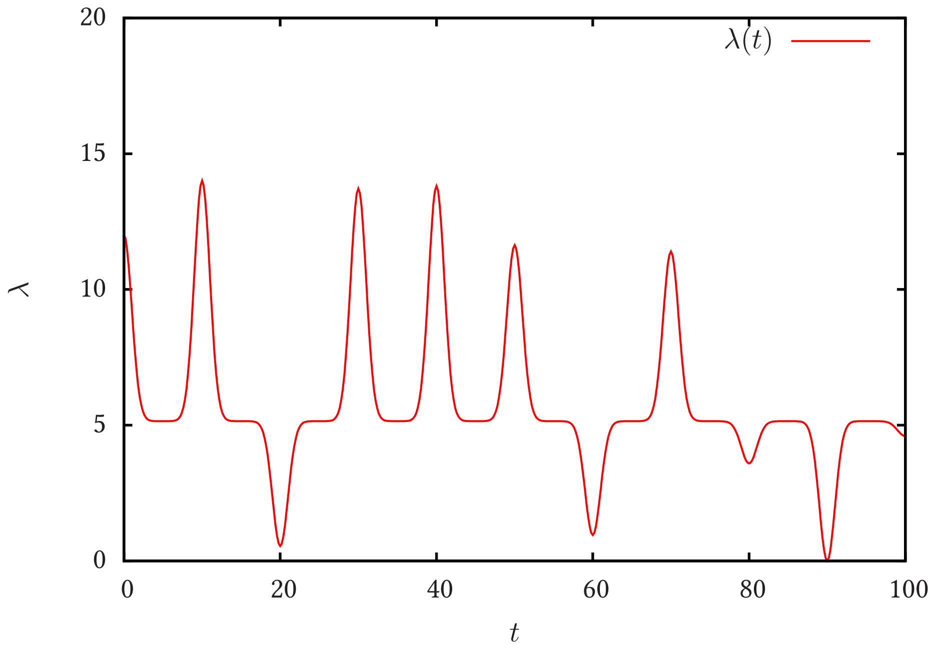
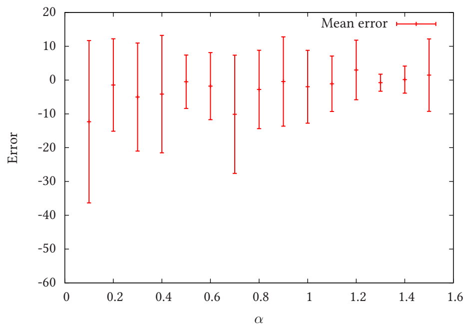


Figure 8: Example of a function generated with α set to 0.1. The straight line at $\lambda = 5$ is a result of the function being shifted to make all values ≥ 0.

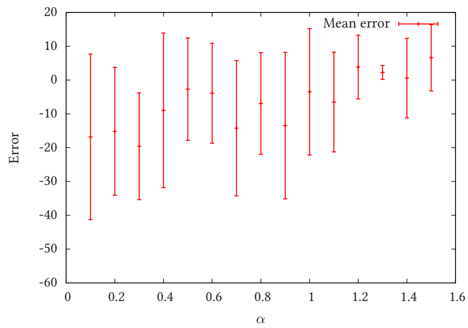
better than the PDF method, and the Gaussian method for function estimation slightly better than the baseline method. However, the differences between the methods were shown to be insignificant.

In the second set of experiments, we used randomly generated functions to gauge performance of the estimators a second time, in order to more accurately represent their performance on data which resembles real data. These experiments indicated that the bad performance in the sine function experiments may have been caused by the characteristics of the functions. The error for all methods was much smaller on average, and apart from a few cases the standard deviation was also much better. We also noted that for some of the functions in the random data set the errors were 5–10 times greater than the average. We believe that this indicates that our methods are not suitable for use on some functions. Looking at the results of the sine function experiments, the likelihood is that the functions which cause trouble are those with a somewhat symmetrical shape, or recurring pattern in them. We were unable to demonstrate a significant difference between any combination of methods, but it appears that the Gaussian kernel density function estimation method combined with the area time delay estimation method produce the best results.

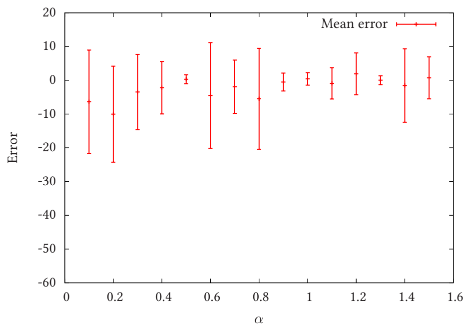
We have achieved all that we had set out to do. We have a method of



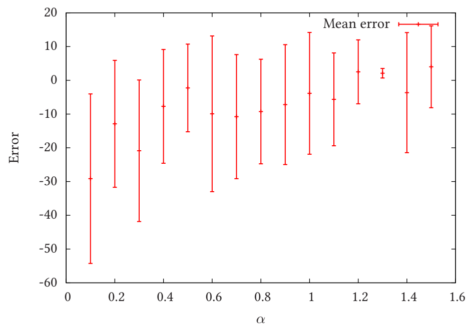
(a) Baseline area



(b) Baseline PDF



(c) Gaussian area



(d) Gaussian PDF

Figure 9: Mean error for each value of α for the second set of random function experiments for each method combination.

estimating functions and time delay, as well as a way to simulate photon streams. In the next section we discuss ways in which the system could be improved, and possible future work.

9.1 Improvements and Future Work

The first improvement is in the simulation of photon streams. Currently, the λ parameter provided to the generator must be larger than the value of the function $\lambda(t)$, $0 \leq t \leq T$. This means that the maximum value of the function must be calculated before the program is run, or a value of λ must be chosen such that the function is unlikely to exceed it. In most cases this does not pose a real issue, and large values of λ can be chosen to no negative effect—the generation of data is still very fast. However, for the sake of completeness and convenience, implementing the generation in such a way that the extra λ value is not necessary may have some benefits. Apart from the thinning method that we have used, there are many other methods of generating nonhomogeneous Poisson processes [21, 14, 18] which could be implemented to improve this aspect of the system.

There is also the potential for improvements to the baseline estimator. Currently, at each breakpoint only the midpoint is considered. To improve the estimates received, using a hierarchical search could be beneficial. Instead of only a single point being used, a search could be done along the line between the points to find the point at which the probability density function was maximised. If this was done for each breakpoint, then it should be possible to find a function which provides an improved estimate compared to the current naive approach.

From our experiments, we discovered that the time delay estimators developed appear to be unsuitable for estimating functions with certain characteristics, namely those which have some sort of periodicity—a good example of this is the sine function. In general, the estimators will struggle to correctly estimate the time delay for functions which have repeating patterns in them due to the way that the methods are implemented. A simple addition to the system which could provide additional information is to provide a confidence value for the time delay estimate calculation. Also, currently only the very highest scoring value of Δ is reported. In addition to this, reporting other peaks in the score may provide more information about the estimate.

Although we have performed several experiments, we were unable to obtain real photon stream data on which to test our estimators. To find out whether our system would be useful in real applications, testing it on actual data would be beneficial.

As mentioned in the introduction, this system is intended to form a base for a system which can automatically identify potential gravitationally lensed objects. We believe that the current system provides a good foundation for such a system. However, given that its accuracy is limited, the idea case is for this system to provide some sort of initial estimate, and then hand over to another system which is able to make more accurate estimates. We have identified three features that could be added as an extension to this system, or as separate systems:

1. Pull stream data from a database or some other form of storage
2. Compute likelihood of a pair of images coming from the same object based on estimates from our system
3. Keep track of which data has been processed and the confidence values of the estimates associated with that data

The combination of our system with a system or systems with these features would potentially create a system that could reasonably be applied to real-world problems.

9.2 Individual Comments

Although I have been required to work on several reasonably large projects during my time at university, this is the largest by far. Other projects of comparable size have been team projects, and as such I did not have to deal with the whole of the code base or management of the project. I believe that working alone on this project (other than weekly supervision meetings) has improved my abilities in many areas. First and foremost, working on a project in a field which I have relatively little experience is quite a daunting task. Before starting I had some interest in astronomy and machine learning, but my knowledge of problems and approaches to solving them in those areas was minimal. Although a deep understanding of astronomy was not required to complete the project, at least some understanding of the natural phenomena was necessary in order to progress. Developing the function estimators was particularly challenging, with literature on the subject being quite heavy on mathematics with which I was unfamiliar. I had to study the papers on which the function estimators are based for quite a long time before I felt confident that I understood the important points. I have come to understand the techniques much better than I did initially, but there is still much to learn. Statistical testing was also a challenging part of the project, requiring me to understand how various statistical techniques work, and which approaches are valid for what data. Processing and analysing the results of the experiments was also new to me, but ended up being a good learning experience which will be useful for any scientific projects I may encounter in the future.

In addition to being in an unfamiliar field with new mathematical concepts, I also chose to write the project in C, a language which I had studied for only a short time before starting the project. Attempting to implement a complex system in a language which one is new to is difficult, and it took a few months before I was able to add new features and modify old code with the confidence and speed with which I can do so in other languages. C has a rather small set of standard libraries, and so I had to implement many features that are commonly available in the standard libraries of Java and Python. For more complex functionality, in order to save time I had to find libraries to use, and work out how to use a system with relatively sparse documentation and information available. I think that forcing myself into an uncomfortable situation in terms of unknown environments has paid off, as I am now confident in the use of C.

During the course of the project, I had to make several decisions about the structure of the code, and make sweeping changes to the code base. One example is the point at which I made the switch from the use of pointer arrays to store estimate data to using structs. This required the modification of some of the fundamentals of the system and required a large amount of care to implement without breaking the functionality. While in team environments it is possible to discuss structural changes and how to go about implementing a new feature, I had to rely on my own judgement to do both, which required a lot of time considering the benefits of one particular approach.

Throughout the project I have attempted to

References

[1] SAO/NASA Astrophysics Data System (ADS). *Über eine mögliche Form fiktiver Doppelsterne.* URL: http://articles.adsabs.harvard.edu/full/1924AN....221..329C.

[2] Martin F Arlitt and Carey L Williamson. "Internet web servers: Workload characterization and performance implications". In: *IEEE/ACM Transactions on Networking (ToN)* 5.5 (1997), pp. 631–645.

[3] F Cannizzaro et al. "Results of the measurements carried out in order to verify the validity of the poisson-exponential distribution in radioactive decay events". In: *The International Journal of Applied Radiation and Isotopes* 29.11 (1978), 649–IN1.

[4] Orest Chwolson. "Über eine mögliche Form fiktiver Doppelsterne". In: *Astronomische Nachrichten* 221 (June 1924), p. 329.

[5] Juan C. Cuevas-Tello, Peter Tiňo, and Somak Raychaudhury. "How accurate are the time delay estimates in gravitational lensing?" In: *Astronomy and Astrophysics* 454.3 (2006), pp. 695–706.

[6] NASA Astronomy Picture of the Day. *A Horseshoe Einstein Ring from Hubble.* URL: http://apod.nasa.gov/apod/ap111221.html.

[7] John W. Eaton. *GNU Octave.* URL: http://www.gnu.org/software/octave/.

[8] Albert Einstein. "Die Feldgleichungen der Gravitation". In: *Preußische Akademie der Wissenschaften, Sitzungsberichte.* 1915.

[9] Albert Einstein. "Lens-Like Action of a Star by the Deviation of Light in the Gravitational Field". In: *Science* 84 (Dec. 1936), pp. 506–507.

[10] B.S. Everitt and A. Skrondal. *The Cambridge Dictionary of Statistics.* Cambridge University Press, 2010.

[11] Free Software Foundation. *GSL - GNU Scientific Library.* URL: http://www.gnu.org/software/gsl/.

[12] CFA-Arizona Space Telescope Lens Survey of Gravitational Lenses. *CASTLeS Survey.* URL: http://www.cfa.harvard.edu/castles/.

[13] Amir Hajjam, J-C Creput, and Abder Koukam. "Approach for optimized and dynamic medical emergency management". In: *Service Systems and Service Management, 2008 International Conference on.* IEEE. 2008, pp. 1–6.

[14] Martin Haugh. *Generating Random Variables and Stochastic Processes.* 2004. URL: http://www.columbia.edu/~mh2078/MCS04/MCS_generate_rv.pdf.

[15] HubbleSite. *First ESA Faint Object Camera Science Images the Gravitational Lens G2237 + 0305.* URL: http://hubblesite.org/newscenter/archive/releases/1990/20/image/a/.

[16] J.F. Kenney. *Mathematics of statistics*. v. 1. Van Nostrand, 1939. Chap. 7, pp. 134–156.

[17] J.F. Kenney and E.S. Keeping. *Mathematics of Statistics, Pt. 1, 3rd ed.* Van Nostrand, 1962. Chap. 15, pp. 252–285.

[18] P.A.W. Lewis and G.S. Shedler. "Simulation of nonhomogeneous Poisson processes with log linear rate function". In: *Biometrika* 63.3 (1976), pp. 501–505.

[19] William A Massey, Geraldine A Parker, and Ward Whitt. "Estimating the parameters of a nonhomogeneous Poisson process with linear rate". In: *Telecommunication Systems* 5.2 (1996), pp. 361–388.

[20] *OECD Glossary of Statistical Terms*. OECD Publishing, 2008.

[21] Raghu Pasupathy. "Generating Nonhomogenous Poisson Processes". In: *Wiley Encyclopedia of Operations Research and Management Science* (2011).

[22] Worldwide LHC Computing Grid Project. *What is WLCG: Data Processing*. 2011. URL: http://lcg-archive.web.cern.ch/lcg-archive/public/data-processing.htm.

[23] Jürgen Renn and Tilman A Sauer. *Eclipses of the stars*. Max-Planck-Inst. für Wissenschaftsgeschichte, 2000.

[24] Sheldon M Ross. *Simulation*. Academic Press, 1997.

[25] Peter Schneider, Christopher Kochanek, and Joachim Wambsganss. *Gravitational Lensing: Strong, Weak and Micro: Saas-Fee Advanced Course 33*. Vol. 33. Springer, 2006.

[26] Jean-Luc Starck and Fionn Murtagh. *Handbook of Astronomical Data Analysis*. 2002.

[27] D Walsh, RF Carswell, and RJ Weymann. "0957+ 561 A, B- Twin quasistellar objects or gravitational lens?" In: *Nature* 279.5712 (1979), pp. 381–384.

[28] Eric W. Weisstein. *Covariance*. URL: http://mathworld.wolfram.com/Covariance.html.

Appendices

Appendix A Installation

A.1 MuParser

download package
run `./configure --prefix=/usr`, followed by `make && make install` (may require sudo) this installs muparser so that headers can be found in `/usr/include`

```
sudo apt-get install libgsl0-dev check
```

Appendix B Usage

B.1 General Usage

B.1.1 Generating Functions

B.1.2 Estimating Functions

B.2 Running Experiments

B.2.1 Creating functions for experimentation

Using the `genfunc_rand.sh` script found in the `scripts` directory, random functions can be generated, conforming to certain parameters. In this file, we specify the directory to which to output by modifying the `OUTPUT_DIR` parameter. The `LAUNCHER_LOC` parameter specifies the location of the `deltastream` executable used to run the program. The `PARAM_FILE` parameter defines the location of the parameter file to use to generate the functions.

Once these have been set, we specify the values to use to generate the function. The values in the the `AVALS` parameter define what values of α will be used to generate the functions. The `DIVISOR` parameter specifies what to divide the values in `AVALS` by when modifying the α parameter in the parameter file. This can be set to 1 to just use the values inside the array. The values in the `AVALS` array are also used to create directories, so the divisor is also used to prevent creation of directories such as `alpha_0.3`. The `NFUNCS` parameter defines how many different functions to generate. `NPAIRS` defines the number of pairs of streams that will be generated from each function. Streams generated will be copies of the function. For example, when `NPAIRS` is set to 5, a function $f(a)$ is generated, along with two streams. Then, four more streams are generated from the same function $f(a)$. This allows for multiple trials on similar data. The FPREF and APREF define the text that is prepended to the directories. Setting FPREF to function_ and APREF to alpha_ will put each set of functions in a directory structure like alpha_1/function_1.

B.2.2 Generating Model Selection Data

Next, we use the stutter_batch.sh script to generate streams with data removed in certain intervals to use for model selection. Here, we set the INDIR parameter to the directory which we set as the output directory in the previous script, and make sure to set the AVALS, NFUNCS and NPAIRS parameters to the same values. We must also define the EXP_PFILE parameter, which tells the script where to look for the experimental parameters. In this file, we must set up which data should be removed. Modifying values in the setup section of the experiment parameter file will allow the choosing of various intervals. To generate a default experiment parameter file, use deltastream -d [filename] -x a. Once this is set up, we run the script, and it generates a new set of files in the same location as the original data which has data in some intervals removed, with names something like random_function_0_output_stream_0_stuttered.dat.

B.2.3 Experiment Parameter Setup

Now, we set up the experiments that we wish to perform on the data. In the experiment parameter file, there are various options which control how the experiments are run. The most important is the experiment_names parameter, which defines the names of the experiments that you wish to run. Once the names are set, we must define four parameters that are used to run the experiment.

```
experiment_names exp_1,exp_2 // Name the experiments
// These parameters will be varied during the experiments
exp_1_params base_max_breakpoints,base_max_extension
exp_2_params gauss_est_stdev
test_exp_1 yes // We want to experiment on this
test_exp_2 no // This will not be experimented on
// Set the estimator to use for the experiment
exp_1_estimator base
ext_2_estimator gauss
// Estimate the function or the time delay
exp_1_type function
exp_2_type delay

// Set the parameter values for experiments
base_max_extension 3,6,...,11
base_max_breakpoints 4,5,...,10
```

```
gauss_est_stdev 1,2,3,4

// This is important! Set the time delay between streams
// Used later to analyse the results
timedelta 0,15
```

When setting the parameter values, `...` can be used to specify a range. In the example, the `base_max_extension` parameters would be 3, 6, 9 and 11. The `timedelta` parameter is important as well—it provides the program with the actual value of the time delay between streams, which is used to determine the score of certain parameter settings. Information about the parameters used to generate streams can be found in the output directories in the `gen_params.txt` file.

B.2.4 Running Model Selection

Once the parameters are set up, we run model selection on the generated streams using the `runexp_batch.sh` script. Here we again set the various parameters needed, and specify a new output directory into which the experiment data is output. Depending on the number of experiments being run, the data can take up a lot of space (on the order of gigabytes), so choose a disk with plenty of free space. It is also a good idea to run a small subset of the experiments before running them all, just to make sure that you are outputting to the correct directory—**data in the output directories from previous experiments is overwritten**. Once you are sure that everything is good to go, run the script. Time taken depends on the number of parameter combinations and number of functions you are running the experiments on. A reasonably large set of data (approximately 151,000 experiments) took approximately two hours on an Intel i5 processor.

B.2.5 Time Delay Calculation

Once the experiments have completed, we use the best parameter settings from the model selection stage to run time delay estimators on the data again, this time with all data available to the estimators. First, we use the `get_goodness.sh` script to extract the experiment numbers of the highest scoring parameter settings. Inside the `runtd_exp.sh` file, we modify the relevant parameters, setting the parameter files to read from, the directory from which to read the parameter data—the directory set as the output directory for the model selection, the location in which the files output from the `get_goodness.sh` script, and the place where we wish to put the files produced by this stage of the process. When the script is run, it performs a time delay estimation on the streams with the best parameters for each function and α value. Inside each directory, a file `results.txt` is produced, which contains the some data about the performance of the estimators with that combination of methods on the given α value for that function. In the next step, we extract this data into a more usable form.

B.2.6 Extracting Result Data

In the `extract_results.sh` script, we set up the parameters so that `INDIR` is set to read from the top level of the time delay results directory, and `OUTDIR` is set to the location to which we wish to output the aggregated results. There are three different flags that can be set to produce data in different forms for processing. The `TT` flag makes the script output error data in a form in which it can be processed by other scripts to run t-tests. The `DV` flag outputs data which can be used to calculate the mean value of the time delay estimate across all functions. Usually, the means are calculated on a per-function basis, but setting this flag outputs data in a form which groups data from all the functions for one value of α into one set which can then be easily processed as a single set of estimates. The `EV` flag does a similar thing to the `DV` flag, but for error data. The error values are grouped by α value, and the resulting files can be used to find the aggregate error for each value of α for a specific method combination. Running the `extract_results.sh` script will output the data. Next, we will explain how to process the resulting files.

B.2.7 Processing Result Data

Inside the results directory, the top level contains files which detail the mean estimate, standard deviation and mean error for each function for each value of α. The `results` directory contains directories with files which are used to produce different data. The `data` directory contains copies of all results files, with the filenames showing what experiment the file was taken from.

T-tests To create data for t-tests, we use the files in the `alpha_errors` directory. With this data we will be able to compare the errors of one combination of method to another. The `ttest_columnate_agg.sh` and `ttest_columnate_individual.sh` scripts are used to process the data further into files readable by the `ttest.m` script. The first script groups data so that when the t-tests are run, results from all functions for one value of α for one method are compared to the same set of functions for the same value of α, but with a different method combination. The second script processes data so that results for individual functions are compared, rather than an aggregate set of data. T-test data will be output to a directory `ttest` in the directory specified in the script. In each file, there will be columns of data used for the t-test, as well as some information about where the data was taken from.

Using the `ttest.m` script, we can run t-tests on the data. The script was written using GNU Octave [7], but should also be compatible with Matlab. The `read_start_x`, `read_start_y`, `read_end_x` and `read_end_y` must be modified to match the data before the script is run. These values specify the range used by the `dlmread` command to parse in data from the files. In the case of 4 columns with 25 lines each, the values are set to

```
read_start_x=0
read_start_y=0
```

```
read_end_x=24
read_end_y=3
```

When run, the script produces a set of t-tests from the data. The `paired_tests` matrix contains the results of two-tailed paired t-tests on the data, and the `single_sample` matrix contains the results of single sample t-tests on the error values calculated by subtracting one set of data from the other. The comparisons array indicates which columns were compared to produce each column of the matrix. In general, 1 refers to the baseline area method, 2 to the baseline PDF method, 3 to the Gaussian area method, and 4 to the Gaussian PDF method.

Mean and Standard Deviation of Estimates Using the `multifunc_mean.sh` script, the mean and standard deviation of estimates from different combinations of methods can be generated. Setting the `INDIR` variable to point to the `results/estimates` directory will perform the computations using a short Octave script, and output the results to a file, which will additionally contain tables for use in Emacs' `org-mode`. Tables 2 and 5 are examples of these tables converted into LaTeXusing the export functionality built into `org-mode`.

Error of Estimates Being able to display the error of combinations of methods, such as the graphs in Figures 4 and 5 is also useful, and data to do this can be produced by the `multifunc_errmean.sh` script. The script will produce files for each combination of methods, which can then be plotted with a program such as gnuplot. One way to plot the data using gnuplot is

```
plot "baseline_area_err.txt" using 1:2:3 with errorbars
```

Appendix C Experimental Data

In this appendix we present the full set of experimental data. BA=baseline area, GA=Gaussian area, BP=baseline PDF, GP=Gaussian PDF. BA/GA indicates a test comparing the Gaussian area method against the baseline area method.

α	Baseline area	Gaussian area	Baseline PDF	Gaussian PDF
0.05	11.432 $\pm$ 6.18	10.388 $\pm$ 3.60	22.064 $\pm$ 11.20	22.20 $\pm$ 10.94
0.10	6.008 $\pm$ 14.46	9.76 $\pm$ 2.67	4.712 $\pm$ 17.68	8.42 $\pm$ 14.73
0.15	-16.44 $\pm$ 19.62	-12.024 $\pm$ 20.85	-11.472 $\pm$ 20.41	-13.308 $\pm$ 20.26
0.30	3.152 $\pm$ 20.09	1.14 $\pm$ 22.01	3.94 $\pm$ 19.40	3.84 $\pm$ 18.67
0.60	-2.62 $\pm$ 19.71	3.00 $\pm$ 18.65	1.404 $\pm$ 25.02	-7.744 $\pm$ 39.14

Table 2: Table of mean estimate and standard deviation for combinations of methods on the first set of sine experiments ($\mu \pm \sigma$, $n = 25$). The actual time delay is 10.

α	BA/GA	BA/BP	BA/GP	GA/BP	GA/GP	BP/GP
0.05	0.47815	0.00017369	0.00011531	1.2894×10^{-5}	7.3895×10^{-6}	0.96623
0.10	0.21721	0.78218	0.56963	0.17294	0.66295	0.43371
0.15	0.45352	0.39424	0.58888	0.92654	0.82960	0.75580
0.30	0.74226	0.89066	0.90272	0.64223	0.64879	0.98556
0.60	0.31540	0.53891	0.56946	0.80325	0.23073	0.33954

Table 3: Table of paired t-test p-values for preliminary sine experiments

α	BA/GA	BA/BP	BA/GP	GA/BP	GA/GP	BP/GP
0.05	0.40134	3.4334×10^{-5}	2.7901×10^{-5}	3.7237×10^{-6}	4.5120×10^{-6}	0.87340
0.10	0.19016	0.77691	0.58516	0.16046	0.65594	0.35306
0.15	0.44337	0.39762	0.54968	0.92600	0.84756	0.72240
0.30	0.77304	0.90010	0.89556	0.60185	0.54686	0.98399
0.60	0.25471	0.49996	0.56251	0.75755	0.22830	0.27032

Table 4: Table of p-values for preliminary sine experiments for one sample t-test performed on error values

α	Baseline area	Gaussian area	Baseline PDF	Gaussian PDF
0.01	-2.60 $\pm$ 21.17	10.81 $\pm$ 16.69	-2.21 $\pm$ 18.15	-4.32 $\pm$ 21.91
0.02	-4.80 $\pm$ 20.77	9.49 $\pm$ 20.36	-14.51 $\pm$ 14.65	-12.22 $\pm$ 15.20
0.03	20.67 $\pm$ 18.17	17.41 $\pm$ 10.52	10.49 $\pm$ 12.84	13.72 $\pm$ 10.65
0.04	9.58 $\pm$ 6.36	9.97 $\pm$ 3.44	27.3 $\pm$ 9.61	23.75 $\pm$ 9.77
0.05	20.58 $\pm$ 9.03	15.61 $\pm$ 3.21	25.06 $\pm$ 7.20	24.30 $\pm$ 7.48
0.06	12.17 $\pm$ 3.97	14.30 $\pm$ 1.48	20.72 $\pm$ 2.74	19.71 $\pm$ 3.09
0.07	15.95 $\pm$ 4.51	15.99 $\pm$ 3.10	16.53 $\pm$ 11.80	15.72 $\pm$ 14.06
0.08	18.42 $\pm$ 3.03	16.70 $\pm$ 2.46	12.35 $\pm$ 10.60	12.52 $\pm$ 10.93
0.09	16.19 $\pm$ 3.24	15.83 $\pm$ 3.25	17.16 $\pm$ 7.50	16.17 $\pm$ 8.51
0.10	16.79 $\pm$ 3.95	15.01 $\pm$ 1.82	6.48 $\pm$ 22.69	6.31 $\pm$ 22.38
0.11	-0.36 $\pm$ 24.73	9.13 $\pm$ 15.92	-6.03 $\pm$ 25.82	-1.06 $\pm$ 23.42
0.12	-12.19 $\pm$ 25.52	-1.36 $\pm$ 23.43	-20.62 $\pm$ 24.16	-16.97 $\pm$ 26.66
0.13	-8.42 $\pm$ 23.48	-24.21 $\pm$ 18.47	-20.71 $\pm$ 21.45	-25.04 $\pm$ 19.28
0.14	-5.96 $\pm$ 20.16	-2.49 $\pm$ 21.27	-11.75 $\pm$ 23.12	-3.53 $\pm$ 24.27
0.15	-10.83 $\pm$ 13.07	-13.80 $\pm$ 19.64	-4.36 $\pm$ 21.96	3.07 $\pm$ 21.17

Table 5: Table of mean estimate and standard deviation for combinations of methods on the second set of sine experiments ($\mu \pm \sigma$, $n = 10$). The actual time delay is 15.

α	BA/GA	BA/BP	BA/GP	GA/BP	GA/GP	BP/GP
0.01	0.15295	0.96699	0.86740	0.13054	0.11670	0.82642
0.02	0.157811	0.266797	0.398533	0.010178	0.019558	0.748621
0.03	0.64686	0.18670	0.33527	0.22703	0.46908	0.56859
0.04	0.87326	0.0002154	0.0018482	7.5319×10^{-5}	0.00085570	0.44706
0.05	0.1371851	0.2597158	0.3539836	0.0020559	0.0049413	0.8286646
0.06	0.14919	4.719×10^{-5}	0.00027971	7.8065×10^{-6}	0.00016456	0.47247
0.07	0.98273	0.89192	0.96325	0.89579	0.95576	0.89615
0.08	0.20268	0.11576	0.13599	0.24583	0.27772	0.97364
0.09	0.81655	0.72573	0.99482	0.63121	0.91209	0.79640
0.10	0.23567	0.19600	0.18339	0.27572	0.26019	0.98741
0.11	0.34587	0.63994	0.95152	0.15109	0.29466	0.67393
0.12	0.36082	0.48103	0.70215	0.10321	0.20355	0.76434
0.13	0.13020	0.26141	0.11813	0.71497	0.92673	0.65779
0.14	0.72652	0.57817	0.81986	0.38812	0.92404	0.47133
0.15	0.710077	0.457267	0.110994	0.349136	0.096720	0.474311

Table 6: Table of paired t-test p-values for second set of sine experiments.

α	BA/GA	BA/BP	BA/GP	GA/BP	GA/GP	BP/GP
0.01	0.238363	0.963181	0.834164	0.109943	0.083662	0.718245
0.02	0.012781	0.337071	0.318196	0.020011	0.011247	0.693379
0.03	0.70689	0.28411	0.42725	0.17888	0.27976	0.46476
0.04	0.85129	5.6159×10^{-5}	0.0016052	7.1618×10^{-5}	0.00051425	0.11740
0.05	0.13016	0.29164	0.38843	0.00076639	0.0017663	0.20065
0.06	0.12141	0.00027185	0.00054176	3.4371×10^{-5}	0.00034019	0.058177
0.07	0.96101	0.89812	0.96606	0.89112	0.95491	0.65016
0.08	0.093847	0.128048	0.147634	0.223662	0.252733	0.634744
0.09	0.71169	0.73272	0.99485	0.66951	0.91808	0.36155
0.10	0.28488	0.22685	0.21387	0.29118	0.27635	0.66831
0.11	0.23385	0.66956	0.95495	0.10105	0.18555	0.61964
0.12	0.42867	0.60176	0.75116	0.11260	0.20142	0.67466
0.13	0.16087	0.33354	0.14625	0.48521	0.91090	0.37568
0.14	0.74527	0.64935	0.84987	0.33823	0.91812	0.15872
0.15	0.74025	0.52892	0.17196	0.40520	0.18736	0.39848

Table 7: Table of p-values for second set of sine experiments for one-sample t-test performed on error values

α	Baseline area	Baseline PDF	Gaussian area	Gaussian PDF
0.4	17.884 ± 10.465	13.096 ± 19.351	16.62 ± 4.6861	15.132 ± 16.708
0.8	16.472 ± 5.2632	15.64 ± 6.6625	15.424 ± 2.259	15.644 ± 2.3131
1.0	7.216 ± 23.638	15.156 ± 15.542	13.6 ± 14.92	13.056 ± 18.349
2.0	15.068 ± 2.911	15.052 ± 4.5451	14.932 ± 1.8674	14.62 ± 4.5364
3.0	16.148 ± 4.246	15.724 ± 7.6279	14.844 ± 3.4838	14.176 ± 9.7994

Table 8: Results for preliminary random function experiments. Values shown are calculated by aggregating estimate data from 5 functions with 5 estimates for each α value. The actual time delay is 15. ($\mu \pm \sigma$, $n = 25$)

α	BA/GA	BA/BP	BA/GP	GA/BP	GA/GP	BP/GP
0.4	0.29071	0.59093	0.4965	0.38927	0.69758	0.67568
0.8	0.63269	0.37356	0.48299	0.88083	0.99779	0.73982
1	0.17468	0.26783	0.34286	0.7245	0.67007	0.91057
2	0.98845	0.84775	0.68509	0.90508	0.74266	0.75625
3	0.80916	0.24102	0.3605	0.60221	0.53605	0.74949

Table 9: Paired t-test results for preliminary random function experiments, calculated by aggregating results for each function for each method and comparing. $n = 25$

α	BA/GA	BA/BP	BA/GP	GA/BP	GA/GP	BP/GP
0.4	0.28246	0.6203	0.43163	0.39593	0.68339	0.69719
0.8	0.524	0.23804	0.44777	0.86289	0.99737	0.7035
1	0.15115	0.12471	0.24086	0.69366	0.48648	0.85042
2	0.988	0.84899	0.67334	0.89557	0.31721	0.71896
3	0.78673	0.13475	0.3649	0.5442	0.1848	0.70261

Table 10: One sample t-test results for preliminary random function experiments, calculated by aggregating results for each function for each method and comparing. $n = 25$

α	Baseline area	Baseline PDF	Gaussian area	Gaussian PDF
0.1	2.668 $\pm$ 24.011	-1.808 $\pm$ 24.508	8.656 $\pm$ 15.301	-14.152 $\pm$ 25.117
0.2	13.54 $\pm$ 13.682	-0.16 $\pm$ 18.927	4.964 $\pm$ 14.226	2.1 $\pm$ 18.803
0.3	9.976 $\pm$ 15.983	-4.576 $\pm$ 15.793	11.528 $\pm$ 11.16	-5.872 $\pm$ 20.965
0.4	10.848 $\pm$ 17.376	6.032 $\pm$ 22.883	12.808 $\pm$ 7.7668	7.284 $\pm$ 16.856
0.5	14.508 $\pm$ 7.9035	12.312 $\pm$ 15.138	15.316 $\pm$ 1.3203	12.732 $\pm$ 12.987
0.6	13.212 $\pm$ 9.9238	11.116 $\pm$ 14.794	10.524 $\pm$ 15.65	5.072 $\pm$ 23.078
0.7	4.88 $\pm$ 17.486	0.74 $\pm$ 20.025	13.096 $\pm$ 7.8901	4.236 $\pm$ 18.376
0.8	12.224 $\pm$ 11.602	8.076 $\pm$ 15.033	9.523 $\pm$ 14.948	5.748 $\pm$ 15.487
0.9	14.568 $\pm$ 13.218	1.524 $\pm$ 21.66	14.488 $\pm$ 2.6585	7.8 $\pm$ 17.765
1.0	13.024 $\pm$ 10.781	11.544 $\pm$ 18.706	15.42 $\pm$ 1.8552	11.148 $\pm$ 18.037
1.1	13.9 $\pm$ 8.1991	8.496 $\pm$ 14.743	14.1 $\pm$ 4.6522	9.348 $\pm$ 13.755
1.2	17.988 $\pm$ 8.8131	18.852 $\pm$ 9.433	16.912 $\pm$ 6.1955	17.512 $\pm$ 9.4673
1.3	14.228 $\pm$ 2.5246	17.264 $\pm$ 2.0459	15.028 $\pm$ 1.3014	17.096 $\pm$ 1.4202
1.4	15.128 $\pm$ 4.0093	15.588 $\pm$ 11.763	13.46 $\pm$ 10.885	11.348 $\pm$ 17.795
1.5	16.46 $\pm$ 10.726	21.592 $\pm$ 9.7962	15.724 $\pm$ 6.2363	19.004 $\pm$ 12.125

Table 11: Results for second set of random function experiments. Values shown are calculated by aggregating estimate data from 5 functions with 5 estimates for each α value. The actual time delay is 15. ($\mu \pm \sigma$, $n = 25$)

α	BA/GA	BA/BP	BA/GP	GA/BP	GA/GP	BP/GP
0.1	0.51733	0.29827	0.019342	0.076423	0.084991	0.00032037
0.2	0.0051315	0.034784	0.017553	0.28463	0.67378	0.54648
0.3	0.0021849	0.69233	0.0042038	0.00012929	0.80605	0.00062073
0.4	0.40614	0.60899	0.46525	0.16734	0.8266	0.14323
0.5	0.52329	0.61644	0.56189	0.32788	0.91659	0.32726
0.6	0.55908	0.47181	0.11175	0.89125	0.27578	0.33316
0.7	0.44001	0.037343	0.89952	0.0060815	0.52319	0.031529
0.8	0.28021	0.47903	0.10078	0.73421	0.59217	0.38478
0.9	0.013321	0.97645	0.13302	0.0046355	0.26822	0.068792
1.0	0.73328	0.27892	0.65733	0.30772	0.93958	0.24459
1.1	0.11578	0.91596	0.16168	0.076168	0.83356	0.10831
1.2	0.73935	0.61978	0.85478	0.39434	0.61843	0.79203
1.3	2.4441×10^{-5}	0.16549	9.5411×10^{-6}	2.9926×10^{-5}	0.73738	2.2806×10^{-6}
1.4	0.85395	0.47565	0.30533	0.50993	0.32528	0.61501
1.5	0.083675	0.76805	0.43587	0.014871	0.41057	0.23495

Table 12: Paired t-test results for second set of random function experiments, calculated by aggregating results for each function for each method and comparing. $n = 25$

α	BA/GA	BA/BP	BA/GP	GA/BP	GA/GP	BP/GP
0.1	0.53491	0.29591	0.023309	0.13562	0.015965	0.0024401
0.2	0.0091442	0.018017	0.0072404	0.22424	0.55365	0.5514
0.3	0.0033213	0.53154	0.0015812	0.00045591	0.80415	0.0012471
0.4	0.44784	0.61693	0.50809	0.16338	0.72314	0.15106
0.5	0.53795	0.5997	0.57453	0.34868	0.87381	0.33991
0.6	0.57406	0.47553	0.12949	0.89505	0.1207	0.15509
0.7	0.37818	0.020646	0.87337	0.0073325	0.11969	0.026935
0.8	0.18908	0.38286	0.056598	0.59594	0.23452	0.23195
0.9	0.004523	0.97677	0.04792	0.0070827	0.12395	0.075853
1.0	0.7347	0.25313	0.52376	0.30931	0.92957	0.22829
1.1	0.098855	0.89295	0.12298	0.081173	0.77935	0.10247
1.2	0.66062	0.52511	0.79827	0.31073	0.23649	0.75251
1.3	6.6396×10^{-7}	0.044196	4.2502×10^{-8}	1.978×10^{-8}	0.48723	6.8121×10^{-10}
1.4	0.8409	0.54461	0.28351	0.42718	0.14969	0.48467
1.5	0.098748	0.63584	0.33856	0.0080649	0.3598	0.21506

Table 13: Single sample t-test results for second set of random function experiments, calculated by aggregating results for each function for each method and comparing. $n = 25$