University of Auckland

Department of Engineering Science

Part IV Project Report

Statistical Gravitational Microlensing Models

Author
Alex Kennedy

Supervisors
Dr. Andreas Kempa-Liehr
Dr. Nicholas Rattenbury

Project Partner
Gemma Nash

September 27, 2019

Abstract

Gravitational microlensing events have led to the discovery of more than 80 planets. In anticipation of the launch of the NASA telescope WFIRST, observations of gravitational microlensing events are expected to become much more numerous and current manual techniques for their analysis will become insufficient. We present a workflow for developing methods to estimate system parameters from microlensing observations. We based this method on time-series feature analysis of simulated light curves and validated it for a parameter space reduced to one dimension. We determined a set of seven features for making accurate predictions of the separation parameter. We present a Random Forest model which is robust to the noise and data outages common to current microlensing data. Furthermore, we present an implementation of Bayesian Linear Regression on polynomial combinations of these features, which produces probability distributions for estimated parameters. Both models have an out-of-sample explained variance of 99.99%. The probability distributions of the Bayesian model enable the implementation of an active learning procedure, whereby new and expensive microlensing simulations can be run in parts of parameter space where there is most uncertainty. This method represents a framework by which the large microlensing parameter space can be efficiently explored so that future observations from WFIRST can be swiftly characterised. Finally, we have written a well-documented Python package `galens` for reproducing and extending our research.

Acknowledgements

Thanks to Nick for the questions.

Thanks to Andreas for the answers.

Thanks to Gemma for the journey.

Declaration of Contribution

Gemma and I worked very closely throughout the course of this project. We have each had significant influence in each component, so a clear split is challenging to define. With this caveat in mind, I will attempt to declare contribution as best I can. I was more involved with the simulation and Python package creation, while Gemma was more involved with the implementation of the Bayesian model. I created each of the plots in this report, bar Fig. 8b and Fig. 10. Experimentation was done iteratively in short cycles, so cannot be easily attributed to either of us.

Contents

1 Introduction

One method of discovery of planets outside our solar system (exoplanets) is gravitational microlensing [29]. This method has been deployed successfully using ground-based surveys to detect and characterise more than 80 exoplanets [8]. The NASA flagship mission WFIRST, a space-based telescope currently in development by NASA, promises a renaissance in gravitational microlensing observations [37], necessitating new and improved modelling approaches to accurately and quickly process new data. Observations of gravitational microlensing events, termed light curves, are time-indexed measurements of stellar brightness. They are complex and non-linear functions of time [29], and it is of interest to develop a method to assist in their analysis automatically. Current methods rely on human intervention or significant assumptions to simplify the problem [29, 18]. We aim to develop a methodology to recover microlensing parameters from their observations by determining a suitable set of time-series features for representing arbitrary microlensing light curves. Hence, we aim to develop a model to predict planetary parameters. Additionally, it is desirable to produce a model that can be used as part of an active learning algorithm, allowing expensive simulations to be run in regions of the parameter space where there is most uncertainty.

2 Literature Survey

2.1 Exoplanet Detection

At time of writing, over 4,000 exoplanets have been discovered and confirmed [8]. There are several methods by which exoplanets are detected, each most suited to the characterisation of parameters for different types of planets [29], with parameters including mass, radius, and orbital elements. Many confirmed planets differ wildly from any objects in our solar system, for example, 'hot Jupiters', which are large planets orbiting very close to their host stars and rocky, heavy 'super-Earths' [29]. Exoplanet discovery carries broad implications for the yet unsolved problem of the origin of our solar system, alongside the fundamental exploration of the universe that exoplanet research represents. Access to an extensive, increasingly representative database of planetary systems beyond our own provides evidence for evaluating these theories of solar system development [23]. Furthermore, the rate of exoplanet detection suggests there are likely many billions of planets in our galaxy alone [29], carrying potentially enormous implications regarding the search for extraterrestrial life [34, 10].

2.2 Physics of Gravitational Microlensing

Gravitational microlensing is the method of exoplanet detection which we consider. Of some 4,000 confirmed exoplanets, over 80 have been discovered by this method. Microlensing is a discovery technique of research interest because it is capable of detecting planets which are difficult to detect via other methods [29]. Notably, this includes lower-mass planets orbiting at approximately a few AU (astronomical units, the radius of Earth-Sun orbit). While other detection mechanisms rely on observations of the host star (the star about which the planet orbits), this is not a requirement for microlensing, as observations are based on a background star. This characteristic allows the detection of free-floating planets (planets not orbiting a host star) [25]. Furthermore, gravitational microlensing occurs by a single snap-shot of a source system. It does not require repitition of slow period signal, as is a requirement for radial velocity and transit techniques Perryman [29]. In particular, the WFIRST telescope, discussed in Sec. 2.3, is expected to find planets with orbital radii of 1–10AU and masses 10% of Earth upwards [28]. The exoplanet database is biased towards planets that are easiest to detect by other methods such as radial velocity and transits. The WFIRST mission will attempt to fill the gaps of exoplanet knowledge [28].

Gravitational microlensing events are one time occurrences which happen when light

from a background star (the source) is deflected by the gravity of a foreground star (the lens) [32]. When the lens passes directly between an observer and the source, this deflection causes a characteristic change in the light observed from the source. The lens has the effect of briefly focussing light from the background star onto the observer due to the bending of light from the source by the gravity of the lens. These events typically last on the order of hours or days. When planets are present in the lens system, they may also deflect light from the source, causing their own change to the observed light curve. This change can be significant, increasing the brightness of the source by several orders of magnitude. Planets in the lens system can be discovered within these light curves.

Microlensing events are the result of general relativity and suffer from multiple degeneracies [32], where different lens systems can produce indistinguishable observations. These properties make it difficult to automatically or quickly characterise the lens system from the observations. Simulation of microlensing events is also challenging and is addressed in Sec. 3.1. To make accurate estimations of parameters from light curves, comparison with an extensive suite of these expensive simulations across a large parameter space is needed.

2.3 NASA WFIRST

The Wide Field Infrared Survey Telescope (WFIRST) is a major NASA space telescope mission, currently scheduled for launch in the mid-2020s. It features a 2.4m telescope, and one of its primary missions will be to observe gravitational microlensing events. It will observe an area of sky 100 times larger than the Hubble Space Telescope [37].

The WFIRST mission represents the first space-based microlensing survey, and is expected to observe vastly more microlensing events than its ground-based counterparts [28] (such as OGLE [35], MOA [22], and KMTNet [19]). The relatively small number of microlensing observations in the past, in addition to their complexity, has meant much analysis is done manually, or by other closely supervised techniques. As a greater number of microlensing events are observed, this approach grows untenable, and the advent of WFIRST compels the production of more automated methods. Some automated approaches have been considered and are discussed in Sec. 2.5.

Our techniques will, where possible, make use of the latest design parameters of this telescope. For example, microlensing events are expected to be recorded with a cadence of 15 minutes [28]. There are further nuances to data expected from WFIRST, which may need to be considered as our models and techniques mature.

2.4 Recent Work

Other methods are being explored to approximate the parameters of microlensing light curves automatically. Khakpash et al. [18] have produced an algorithm to approximately recover planetary parameters from microlensing light curves, developing an idea proposed by Gaudi & Gould in 1997 [13]. The approach involves fitting the parameters for the main microlensing event (the magnification from the star), then fitting smaller planetary perturbations on top of the primary curve. This approach is limited to binary lens systems. Furthermore, if the light curve has multiple peaks, which is possible in binary systems, then their algorithm does not perform well.

Part of the challenge of working with gravitational microlensing observations is that they can be difficult to simulate and take significant computation. This makes producing training data expensive. Efficient and optimised algorithms exist for certain classes of microlensing events, such as binary systems [5]. However, different systems require expensive numerical calculations. Perryman [29] adumbrates the many difficulties in light curve modelling and fitting. Under some conditions, the magnification of the source star can be formally infinite. In these regions, higher order effects become important, yet inclusion of these higher order effects for the duration of the entire simulation are prohibitively expensive for large numbers of simulations. The standard model fitting

techniques described by Perryman require many simulations across a large parameter grid. Because of this, it is of interest to minimise the number of simulations needed for a model.

2.5 Statistical Methods in Physics

Success has been achieved using similar methods to the ones we propose in other areas of Physics. Doctor et al. [9] proposed a Gaussian process regression method to produce reduced order models of gravitational waveforms created by compact binary coalescences, such as black hole mergers. They used an active learning algorithm to produce new, expensive general relativity simulations at positions in parameter space where uncertainty was highest. Using this approach, they were able to produce high-quality approximations of simulations. We propose the exploration of a similar approach for microlensing simulations. Their approach, and that of the FRESH algorithm differ in that their transformation was invertible; an approximation of original light curve could be recovered from their representation. Our features, in general, cannot be used for this purpose, but we discuss methods to bypass this issue in Sec. 2.6.

Other machine learning approaches have been developed for physics applications. Feindt [11] presented a neural network where the outputs of the model are probability distributions. This model was presented alongside one of its first applications in high-energy physics. Baehr et al. [1] discuss the use of this algorithm for online analysis of the high bitrate data stream from the Belle-II pixel detector, a particle physics experiment.

2.6 Bayesian Statistics

In Sec. 3, we address the goal to identify probability distributions of estimated parameters, not just point estimates. For this purpose, we turn to the Bayesian view of statistics. Ghahramani [14] discussed the idea that a probabilistic approach is central to designing state-of-the-art, successful machine learning workflows. One of their messages was that of the value of Bayesian optimisation, the concept of identifying where a model or function representation is most uncertain in order to optimise where to simulate or evaluate next. It is this general approach which we wish to apply.

The further strength of Bayesian Statistics, in general, is that it lends itself well to an intuitive understanding of problems. The idea of reassigning probability of an event in light of new evidence is the natural way in which we think as humans, as described by Kruschke et al. [21]. The resultant probability distributions of estimated parameters provides an implicit understanding of the confidence in estimations. Furthermore, analytic expressions for Bayesian Linear Regression are discussed by Bishop [3]. Also discussed by Bishop, is that Bayesian statistics can be used for model comparison. When the methodology identified in this study is extended beyond the limited parameter space, microlensing light curves are likely to be analysed under a range of different assumptions.

3 Statement of Research Intent

The goal of this study is to design a methodology for predicting the parameters of gravitational microlensing observations. We will seek to identify and develop this methodology for a suitable reduction of the large parameter space of general microlensing events. Furthermore, probability distributions of predictions is sought because such an output could be applied to an active learning algorithm, a method to improve the efficiency of the general problem of broad parameter range simulations. The research should pertain to observations expected from the planned WFIRST mission. Furthermore, we will adhere to best practice data science to ensure that subsequent researchers can reproduce all results. We will develop a well-documented codebase for our methodology in order that other researchers can re-use the code.

3.1 Simulation of Microlensing Events

3.1.1 Event Simulation

In order to approximate the parameters of microlensing observations, it is necessary to simulate observations of these events. The simulations form the basis of training and test data in the absence of large amounts of real data. For the task of simulation, we have elected to use the software package `MulensModel` [30]. This package implements several algorithms for simulating microlensing events such as VBBL [5]. However, it is restricted to working with binary lens systems. `MulensModel` implements state-of-the-art algorithms under this constraint. It includes treatment of higher-order effects such as limb darkening and runs quickly, making it acceptable for our goals.

In the first instance, `MulensModel` will be suitable for our needs. Methods to generate light curves for more complex lens systems are not as well developed or accessible. The efficient and accurate VBBL algorithm was published in 2018 [5], and further research into simulation is ongoing. Use or discussion of more general methods of simulation, such as ray tracing methods, is outside the scope of this study. The methodology we intend to implement can stand under these constraints because they are the assumptions under which we operate.

3.1.2 Error Simulation

Observations of microlensing events can be noisy, and the treatment of this error is critical to producing a useful solution. It may be possible, with significant effort, to simulate the expected noise of WFIRST under the most recent design parameters [28]. However, given the associated difficulties, we will make use of noise reported in observations of the OLGE project [35]. OGLE is a gravitational lensing survey using a 1.3m telescope at Las Campanas Observatory, Chile.

The noise associated with WFIRST observations is expected to be significantly lower than that of any ground-based survey [28]. However, the mapping of OGLE uncertainty to WFIRST is outside the scope of this project.

3.2 Light Curve Representation

We aim to determine a suitable set of time-series features that can be used to represent the light curve of a microlensing event. The features will be used to form an algorithm for estimation of parameters. The parameter space of microlensing events has a high dimensionality and suffers from multiple degeneracies [29], where multiple sets of parameters can produce the same light curve. We aim to identify a set of features of time-series which represent a mapping to a reduced parameter space and can be used to predict parameters of the system swiftly. In addition, the reduced size of the feature space makes a comprehensive exploration of the space more feasible. The challenge lies in determining a set of features which sufficiently capture the subtleties of microlensing light curves, while not being too large. A determination of light curve features for successful

prediction would represent a discovery in itself: that it is possible to make predictions about microlensing events using time-series features.

We intend to use the machine learning library `tsfresh` [6], which implements the FRESH algorithm [7]. We will do this by extracting large numbers of features from simulated microlensing events. `tsfresh`, implemented in Python, allows the extraction of nearly 700 features, collected from various research domains. Next, we perform hypothesis tests and explore other feature selection techniques to determine the features most useful for predicting simulation parameters. These features will then inform an algorithm to make predictions of parameter distributions for the simulated microlensing events.

A benefit `tsfresh` is that its set of features includes those identified by Dr Nick Rattenbury, the domain expert for this project, as promising avenues of exploration. The primary example of this is wavelet decomposition of the observations, which are like microlensing observations in that they are non-periodic and localised.

3.3 Prediction Algorithms

The prediction algorithms used in this project are Random Forest and Bayesian Linear Regression. Random Forest is a general, non-linear estimator with a well-tested implementation in scikit-learn [27], the Python package for our machine learning efforts. It is easy to parameterise, yet powerful. The downsides of this algorithm are that it can be challenging to interpret, and provides no estimate of uncertainty. Secondly, we turn to Bayesian Linear Regression. This is like simple linear regression but also yields probability distributions for parameters and distributions. We turn to Bishop [3] for a presentation of this algorithm. The disadvantages of this algorithm is that we are limited to a linear treatment of inputs.

3.4 Active Learning Algorithm

We aim to determine a suitably performing prediction algorithm which includes a presentation of uncertainty. This algorithm will then be used in future research to perform expensive microlensing event simulations efficiently by running microlensing simulations at places in the parameter space with the highest uncertainty. In this way, the algorithm should spend more time resolving the most complex changes in the microlensing light curves. When real data is then introduced, we aim to be able to make quick predictions about the system those data represent with associated uncertainty.

3.5 Methodology

Most importantly, we seek to develop, implement, and validate a methodology for recovering parameters of arbitrary microlensing events from their observations. By developing this method using best practices of reproducible data science, and validating it for a reduced parameter space, we open the door for future researchers to pick up and extend this research.

4 Methods

4.1 Project organisation

This project uses standard Data Science best practices, as described by Wilson et al. [38]. In particular, special care is taken to ensure that all data and results are reproducible and saved clearly. All the steps to produce data, are retained, as well as all raw data. This data was then synced continuously to Google Drive. All code, exploratory notebooks, documentation, logs and computing environment specifications are version controlled in Git. Git stores the complete history of changes to our project, with unique strings (Git hashes) identifying the state of our repository at any time. The hash is all that is required to revert our code and environment to precisely what it was at any time. All results are then reproducible, allowing easy error correction and for us to return to any good models identified. Such practices also enable effective group collaboration and serves as a self-documenting log of work done.

4.1.1 Version Control

This project makes comprehensive use of the version control technology, Git, hosted by GitHub[1]. Version control confers a range of benefits. Firstly, using Git allows easy collaboration, with each team member working on a complete copy of the project. The complete project is also always available in the cloud. Version control of all work, code, and parameterisations allows for complete reproducibility of our work, which is critical when undertaking a research project. Wilson et al. [38] discuss the need to keep track of all changes made by a human immediately, to share changes frequently, and to mirror the project on the machines of all collaborators. We have conformed to each of these standards, committing all changes at least daily.

Git ensures a complete history of code and work done is stored, as well as its author in the form of a commit log. This commit log forms an automatic, dated, and descriptive journal of work done on the project. By making use of descriptive commit messages, we ensure that we keep a journal of progress done on this project and by whom, in addition to manual notes and journalling. This project has over 180 commits recorded.

4.1.2 Folder structure

We have maintained a file folder structure also suggested by Wilson [38]. The most important thing is that we agreed on an organisation structure, and this was one of the first things decided in this project. By making use of, and agreeing on, a tested, commonly used structure early, we avoided any unreasonable messiness or inefficiency. An outline of the file folder structure is shown in Fig. 1.

The `galens` folder, our project's Python package, contains our production style code (Sec. 4.1.5). The `docs` folder contains manually created documentation, including meeting minutes, code descriptions, and discussions of important changes and results (Sec. 4.2). The `experiments` folder contains specific simulation parameterisations, as well as Jupyter notebooks containing all analyses and results obtained (Sec. 4.1.6). The `conf` folder contains parameters for retrieving data from external sources. Each of these folders are version controlled. Finally, the `data` folder contains raw external data, processed outputs, and simulation results. This folder was backed up continuously to Google Drive. It is not version controlled as it contains too much data. Despite this, each file is entirely reproducible by running the code (set to the appropriate version) used to create or retrieve the data in the first place.

4.1.3 Python

The primary coding language for this project is Python 3.7, the latest version of Python at time of writing. We selected Python for its strong scripting capability and

[1] https://github.com/alex-kennedy/microlensing. Access can be provided on request.

because it is easy to write and debug. Python is extremely well maintained, and its use is ubiquitous in Physics and Data Science applications. As a widely-used programming language, there are many well written, fast packages for our applications, such as NumPy, Pandas, Scikit-learn. Furthermore, excellent domain-specific packages are implemented in Python, granting easy access to these packages. Most importantly `tsfresh` [6] for feature extraction and `MulensModel` [30] for simulation of gravitational microlensing events. While faster programming languages exist, Python's ability to interface with faster, compiled languages such as C++ go a significant way to mitigating this disadvantage. The other advantage to Python in this context is that it is dynamically typed, making prototyping swift in contrast to statically typed languages.

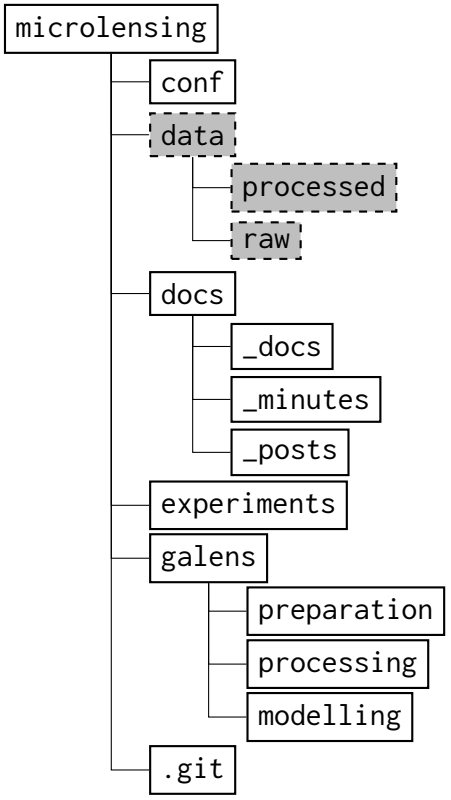


Figure 1: Project folder structure. Unshaded folders are version controlled in Git. The grey boxes are not version controlled, but are backed-up in Google Drive.

Python is free, open-source, and platform-independent, meaning our results are reproducible by other researchers and free for others to implement. A vital goal of this project is to design a procedure for others to use and extend, and this was a key consideration in our decision.

4.1.4 Anaconda Package Management

It is critically important that we run our code in a known environment, that is to say, with known versions of all packages and languages used. It is essential for collaboration, to ensure the same code produces the same outputs for different platforms, people, and machines, and to ensure reproducibility. We frequently found that different versions of packages would produce different outputs when running the same code. These problems are most common in less well-maintained scientific packages, such as `MulensModel`. These problems can undermine the veracity of reported results, so care was taken to record the correct version.

In order to maintain a consistent environment, we elected to use the Anaconda package management platform for Python. Anaconda is a well maintained, free platform designed for this purpose. To use this, we created an environment configuration document listing all packages and their version numbers. This configuration file was version controlled to ensure we all had the same version, and we could restore correct versions of the environment if we returned to a previous state of the repository. This provides a cheap way to store environment specifications with every code revision.

4.1.5 Code style

Code written for this project may be roughly divided into production and notebook code. Notebook code tends to be written swiftly, run few times, and run manually (see Sec. 4.1.6). Production code, however, tends to be run many times and is optimised. Our production code in this project has been organised as a Python package, `galens`. Production code in this project follows the PEP8 style for Python code [36]. We format all production code with `yapf` [39], a formatter for Python files created and maintained by

Google. We impose these standards on our project code because it improves readability, aids optimisation, and promotes consistency. These criteria are important for publishing code useful to others.

The most important aspects of style concern docstrings and commenting. A docstring is a Python inline comment, one is written for each class and function. They contain a one sentence description of the function, inputs with data types, and types of outputs. The docstring style we decided to use is the Google style docstring because it easy to read and write, and consistent with the Google Style Guide. This format is parseable by automated documentation generators, such as Sphinx or `pdoc3`. Complete documentation the `galens` package is available in Appendix A. An example docstring for a toy function is shown below.

```
"""Creates a list of light curves.

Args:
    number (int): Number of curves.

Returns:
    list: Light curves.
"""
```

Our dedication to a self-consistent code style helps with code maintenance and collaboration, critical of any reproducible research project.

4.1.6 Jupyter Notebooks

Most experimentation took place within Jupyter notebooks. All notebooks are then version controlled and ordered by date. Our experimentation follows the literate programming paradigm where code, explanations, parameterisations, and outputs are interspersed together. The code is executed within the notebook, and the outputs are retained. The notebooks are then version controlled. In this way, they provide a comprehensive log of experimentation done. Notebooks allow us to reproduce and understand any experiment run at any time by any team member. Not only are Jupyter Notebooks powerful logging and discussion tools in their own right, but the way we used them meant they formed a natural, free-flowing history of work, progress and details of critical results. The idea is to treat experimentation as works of literature and focus on explaining the thought process, as described by Donald Knuth when he first introduced Literate Programming in 1984 [20].

4.1.7 Infrastructure

Computation took place by remote (SSH) access to a 64GB, eight core Apple iMac Pro. Using this remote resource allowed us to start long-running jobs and offered high computational power. Gravitational microlensing event simulation and feature extraction requiring significant computation. We could then access outputs of simulations and code in a single location, adding another layer of care to ensure we work on the same data. It was data on this system which was continuously backed-up on Google Drive.

4.2 Documentation

Our work is well documented to ensure team members can check the most recent progress, and to ensure we accurately record the reasons for decisions. The most important part of the documentation process resides in the commit history, discussed in Sec. 4.1.1. However, there are also more explicit documentation steps taken. The documentation discussed in this section is in addition to in-code documentation, including our project website, log, and meeting minutes.

4.2.1 Website

Written documentation, including meeting minutes, was made available to team members via a website[2]. The code and documents for this website are version controlled in our central Git repository. Following every commit, the website is automatically built and updated (continuous integration), so everyone has the most recent version. The website is hosted using GitHub pages.

Note the website is not available publicly, requiring user authentication to access, but we can provide access on request. This use of the website made documentation highly accessible and easy to manage. New documentation can be written in simple Markdown from a standard text editor within the working copy of the repository. Markdown is very easy to write, and plain text is acceptable Markdown. It is also the same documenting language used in Jupyter Notebooks. For these reasons, documentation is easy to access and to write. Furthermore, our Markdown version provides functionality for LaTeX mathematics, so we can present the full range of discussions we have. This is an essential part of long-running and substantial projects to ensure both that documentation is read and written.

4.2.2 Log

The project log is a location on the website for manually created journal entries. We used this location to discuss significant additions or changes to the project and to provide descriptions of decisions made for our production style code. The log is in addition to the natural work log from Jupyter Notebooks and commit history.

Example articles from this section contain discussion on decisions such as units, notation used, and decisions made about how and why specific models are implemented.

4.2.3 Meeting minutes

We type and store all meeting minutes on our website. We do this to ensure that everyone had an understanding of the key takeaways from each meeting and any subsequent action items. They are also useful as a reference when undertaking long-running work, or referring back to decisions made and texts or articles discussed.

4.3 Notation

We will present a brief discussion of the physics of gravitational microlensing events and the notation used. According to the principles of General Relativity, all bodies with mass distort and warp spacetime. When light passes through this distortion, an external observer will see the path of that light beam deflected. It is this phenomenon which produces gravitational microlensing events. As we monitor stars throughout our Solar System, there is the possibility of another body, by chance, eclipsing our line of sight to the background object. The eclipsing body is called the lens. When this occurs, the mass of the lens results in the bending of light from the background. The effect of this distortion can be to focus the light onto the observer. This can occur on an extreme scale, for example, if an eclipsing body is a galaxy. In this situation, it is sometimes possible to observe the background body as a ring, termed an Einstein Ring [15]. Usually, however, the complete ring cannot be resolved. Instead, we observe the background star to brighten as the lens moves across our plane of observation, hence the term 'microlensing' [32].

We utilise the notation presented by Rattenbury [32] and Perryman [29] here. Microlensing events are the result of a lens system moving across a plane perpendicular to the observer at some transverse velocity $v_\perp$, at some angle α. The closest approach to the centre of mass of the source system occurs at t_0, at a minimum distance of u_0. Microlensing events are typically defined with reference to a characteristic length scale, called the Einstein radius R_E. The Einstein radius depends on the mass of the lens system

[2] https://alexk.nz/microlensing/

M_L and the following distances: from observer to the lens D_L, from the lens to the source D_{LS}, and from observer to source D_S. The Einstein radius is

$$R_E = \left[2R_S \frac{D_L D_{LS}}{D_S}\right]^{1/2} \tag{1}$$

where $R_S = 2GM_L/c^2$ is the Schwarzchild radius of the source, G is the universal gravitational constant, and c is the speed of light. This yields a characteristic time unit called the Einstein crossing time $t_E = R_E/v_\perp$. For binary planet-star systems, the distance from the star to the source projected into the observer plane s is measured in this length scale. The mass ratio of lens planet to star $M_P/M_\star$ is labelled q. The radius of the source star in units of R_E is ρ. A detailed discussion of the mathematics behind amplification is outside the scope of this project.

Throughout this study, we use magnitude as our measure of brightness [26]. Magnitude is a unitless measure of the apparent brightness of a celestial body used by astronomers in some form since antiquity. It is a logarithmic measure, where smaller values represent brighter objects. More specifically, the magnitude M is defined as

$$M = -2.5 \log_{10}\left(\frac{F}{F_0}\right). \tag{2}$$

Magnitude is defined at specific photometric filters, where F is the flux density subject to that filter, and F_0 is the reference point in that filter. However, we are concerned only with magnitude differences subject to some magnification A. Magnification is the value determined by `MulensModel`. The base magnitude of the source is then adjusted by this magnification ΔM.

$$\Delta M = -2.5 \log_{10} A \tag{3}$$

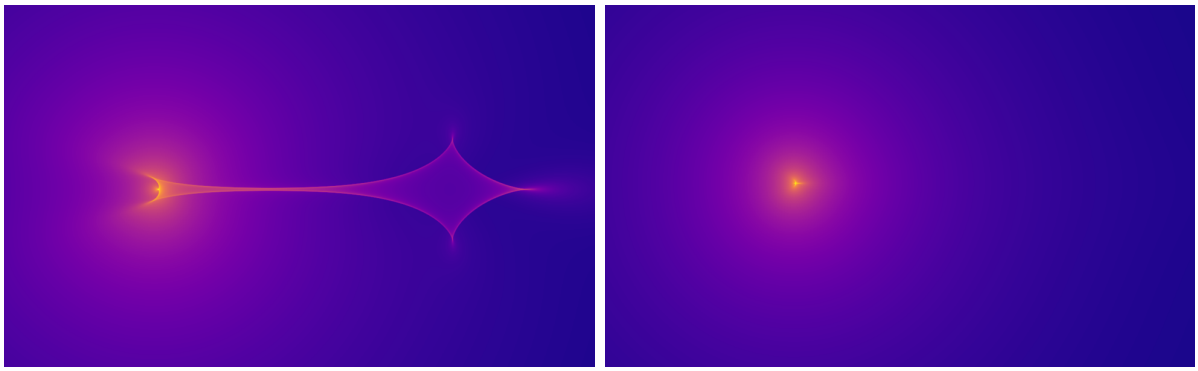
Gravitational microlensing events occur as a lens crosses a source, causing the source to be amplified. This allows the creation of theoretical magnification maps [29]. These maps illustrate the magnification of the source as the lens is seen at different locations in the observation plane. The source then moves in a straight line across it and we see a single slice of the magnification. Fig. 2 demonstrates the magnification of two different lens systems, demonstrating the strange, unpredictable effects that varying even one parameter can introduce. Fig. 2a shows a magnification map for $s = 1.3$ and Fig. 2a for $s = 2.2$. All other parameters are held constant. The strange diamond shape in Fig. 2a is the planetary caustic, magnification caused by the planet, which can be extreme. In Fig. 2b, the planetary caustic is not visible. This offers a brief insight into some of the complexities of working with these data.

4.4 Simplifying Assumptions

Here, we address the modelling assumptions under which we conducted the project. The assumptions are necessary to produce a manageable project, and furthermore yield meaningful results.

The parameter space of gravitational microlensing events can be extremely large. Microlensing events may considered for multi-planet systems, and multi-star sources or lenses. In each case, six parameters are needed to constrain the location and momentum of each body, as well as the planetary radii. Additionally, the transverse motion of the lens system with respect to the source affects the event observations. That stars are not point sources of light requires the consideration of limb-darkening effects, where the stars appear dimmer at their edges, adding additional time-dependent effects. Given the size of this parameter space, it is necessary to constrain it significantly in order to make any progress.

We constrained our investigations to events containing a one planet, one star lens system, and a one star source system. Furthermore, we hold all parameters of the



(a) $s = 1.3$ (b) $s = 2.2$

Figure 2: Magnification maps of two microlensing events with different planet-star separation s parameters. Other parameters are held constant. These images demonstrate some of the complexities of working with microlensing data.

lens system constant except the planet-star projected separation in the observer plane s, measured in units of the Einstein radius for the system. We vary s between 0.5 and 3.0. While this may appear to be an over-simplification, these constraints meet the following criteria, ensuring we still obtain useful results.

1. Systems like this are common in nature, and one planet systems are good approximations of multi-planet systems under some conditions. Indeed the first microlensing discovery was identified as a binary lens [4].
2. The reduced parameter space exhibits the highly non-linear behaviour and complexity which makes general microlensing events challenging to understand, as shown in Fig. 3.
3. Success under these assumptions would still be valuable, as a model has not been developed to predict planetary parameters accurately across the range of s we consider. Furthermore, the verification of the methodology is a critical finding of this study.
4. These events can be simulated quickly (on the order of one second per light curve).

Parameter	Value
t_0	0 days
u_0	-0.1
t_E	61.5 days
ρ	0.00096
q	0.01
α	25°
s	0.5 – 3

Table 1: Lens system parameters used for simulation.

We chose the binary lens parameters in Table 1 for our study. While artificial, there are similarities to real binary lens systems, such as the microlensing event OGLE 2003-BLG-235/MOA 2003-BLG-53, which led to the confirmation of the first planet by microlensing in 2004 [4].

Another way of thinking about holding all parameters constant is to consider a source star crossing the plane of observation of the lens system in the same place, at the same angle. The source star is the same. The only difference is the projected distance of the planet from its host star, in the plane of observation.

4.5 Gravitational Microlensing Event Simulation

A necessary part of this study involves the simulation of gravitational microlensing events under the assumptions discussed in Sec. 4.4. The numerical simulations are deterministic, with stochasticity entering in the observation of these events. To simulate a microlensing event, we first run the numerical simulation, discussed in Sec. 4.5.1 and Sec. 4.5.2. We developed a model for the observation noise, which we then inject into these simulations, realising the uncertainty, discussed in Sec. 4.5.3. We decided not to implement an error model based on the WFIRST design parameters as they are subject

to change as the telescope design progresses [28]. Instead, we built an error model based on observations of the fourth phase of the Optical Gravitational Lensing Experiment (OGLE), a ground-based, long-running microlensing survey. This provides a much more conservative estimate of error, as observations in this survey are subject to significant interference from the atmosphere. Finally, astronomical observations are subject to data collection issues which can cause outages in the light curves, and we developed an algorithm to understand the effect of these issues, outlined in Sec. 4.5.4.

4.5.1 Event Simulation

We used the Python package `MulensModel` [30] for all microlensing event simulation. This package was specifically designed to attend to the needs of simulation for WFIRST and is under active development, making it perfect for our needs. The code is open source and available on GitHub[3]. `MulensModel` provides an easy Python API to a variety of state of the art simulation algorithms.

Events were simulated using the parameters discussed in Table 1. We sampled from a uniform distribution between 0.5 and 3.0 for s, holding all other parameters constant. It is important that microlensing events in this range exhibit the non-linear complexities that make modelling arbitrary microlensing events challenging. We can determine by inspection that values in this range do exhibit this behaviour. Fig. 3 shows four different event simulations at different projected planet-star distance s. In Fig. 3a with $s = 0.9$, we see a 'classic' microlensing event: a large central magnification peak representing the lens star, and a smaller peak produced by the planet. If we increase s slightly to 0.98, Fig. 3b, we still see a central peak from the star, but the effect of the planet becomes extreme and produces two peaks. If s is increased just slightly more to 1.1, Fig. 3c, the effect of the planet is still extreme, yet markedly different. In Fig. 3d, there is virtually no effect from the planet. In this microlensing event, there may be no way to infer the existence of a planet at all.

It is by considering these complexities that we can justify such a reduced parameter space, as it is evident that varying of even one parameter in a constrained range produces extreme non-linear results.

All events are simulated along a sixty-day window, which is approximately the Einstein time t_E of all simulated events. The Einstein crossing time is the time taken for the source to travel the Einstein radius of the lens, the characteristic distance unit of the lens system.

4.5.2 Simulation Algorithm

As we have limited our simulations to binary lens systems, the package `MulensModel` makes use of the algorithm `VBBinaryLensing` (or VBBL), presented by Bozza in 2018 [5]. VBBL makes use of an optimised contour integration technique and considers finite source effects and limb darkening, higher-order effects needed for accurate simulations [29]. The optimised technique ensures that our simulations are relatively fast, and utilises decision trees to determine the parts of the simulation where more accurate treatment is needed. Simulations using this algorithm are therefore quick and accurate and allow error control, should it be needed. Finite source effects arise from the fact that the source is not an infinitesimal point, but a disk of a finite size. Relaxing this assumption makes simulation much slower, but it is a requirement for the accuracy needed in our study. Limb darkening is the phenomenon where the edges of a stellar disk are apparently less bright than the centre; the disk is not uniformly bright.

A comprehensive comparison and understanding of different simulation frameworks is outside the scope of this project. However, we have ensured that `MulensModel` using the VBBL algorithm meets our project requirements for speed, precision, and accuracy.

It is interesting to note that we have not identified similarly efficient simulation techniques expanded beyond binary sources. A complete magnification map must be

[3] https://github.com/rpoleski/MulensModel

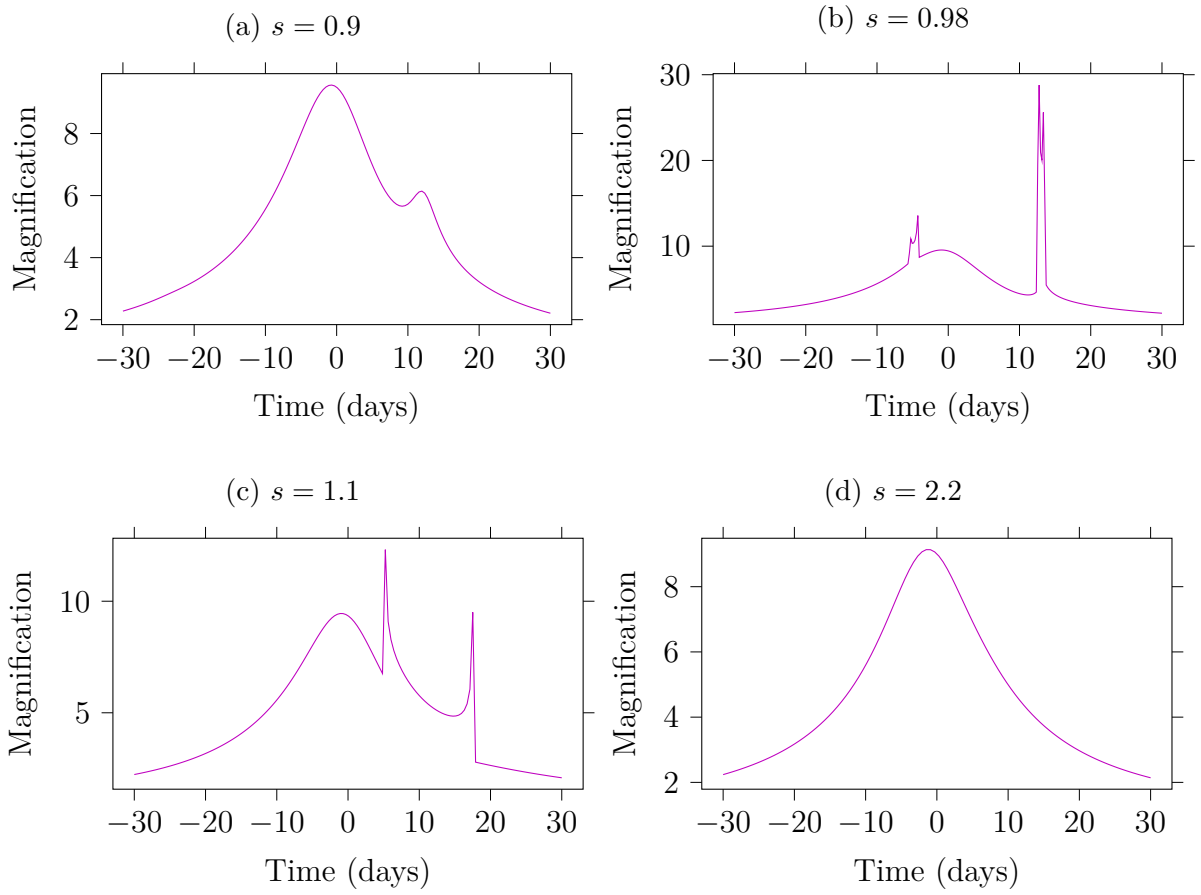


Figure 3: Magnification curves of gravitational microlensing events with varying s.

created in these cases, whereas VBBL avoids this [5]. Multi-source simulation of n-body sources is a computationally expensive endeavour and is the subject of heavy research interest in its own right. This only increases the need for a modelling approach to reduce the number of simulations required to understand the parameter space. A demonstration that this approach can work using the efficient simulation algorithm addressed here, therefore, has the potential to be of great interest.

4.5.3 Observation Uncertainty

Real observations are affected by observation error. This error depends most strongly on apparent brightness of the source star, termed the base magnitude. In gravitational microlensing events, the magnitude of the source can change significantly throughout the event, consequently, as does error. During periods of high magnification, the brightness of the source can appear to increase by multiple orders of magnitude, named high magnification events [29]. At such periods of magnification, the uncertainty is lower because the source star appears to be brighter.

In order to adequately consider the effect of observational error, we needed to develop an error model which reflected this dependence on the brightness of the source.

We rejected developing a complex error model based on the most recent design parameters of WFIRST, as the design parameters are in flux, and the benefits offered are slim. We also rejected developing an error model from observations of other space telescopes, such as the Gaia telescope because it observes in different wavelengths to current microlensing surveys and the planned wavelengths of WFIRST.

Our error model is an empirical distribution based on OGLE-IV observations [35], which are the best microlensing observations currently available. Furthermore, because the observations are ground-based, they provide a conservative estimate of error expected

by WFIRST.

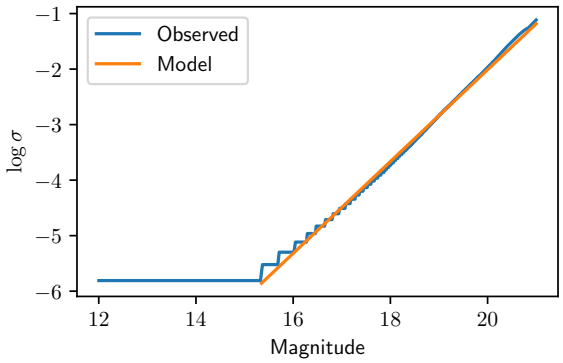


Figure 4: Observed and modelled noise (standard deviation) as a function of apparent magnitude.

We created the error model by obtaining a complete set of OGLE-IV observation data and considering the standard deviation on the observations as a function of the observed magnitude. We found there to be a linear relationship between magnitude and the logarithm of the noise. As magnitude is already a logarithmic unit for observed flux, we obtain a log-log model. In theory, observation error is dependent not just on the observed brightness, but also the visibility conditions at the time of observation. However, we found that OGLE-IV errors showed minimal variation on this basis, so we used a linear log-log model for our error. However, the data were also subject to an error floor, which we account for by using a piecewise linear function. The lowest error is $\sigma = 0.03$, occurring below magnitude values of 15.35.

Uncertainty is realised atop the deterministic simulation results. First, a standard deviation for the observation is determined from the error model. Then, the simulated value is adjusted by a random number drawn from a Gaussian distribution with mean of the simulated value, and standard deviation determined by the magnitude.

The overall error of a microlensing event is strongly affected by the base magnitude of the source star. In the OGLE-IV observations, sources are observed in a magnitude range of 12–21.7 [35]. WFIRST is expected to survey fainter stars. Our error model uses the base magnitude of the source as a proxy for the size of the error. A higher magnitude represents a dimmer star, so a higher magnitude represents a higher error. The issue with this error model is that it will not be equivalent to error with the same base magnitude in WFIRST. However, the error is expected to be much less and more accurate differences could be determined from our model as the design parameters of WFIRST become more stable.

Under this model of noise, peaks in brightness are observed to have lower error, a feature of genuine microlensing observations, as shown in Fig. 5.

To create this model, we used measurements from over 15,000 light curves from the OGLE-IV survey; all identified microlensing events from 2011 to May 2019, in total, 38 million individual observations. The piecewise function (Eqn. 4) gives the final model for standard deviation (noise) as a function of observed magnitude M. The mean squared error of the linear fit to the target range is 0.996, which is more than sufficient for our purposes. Fig. 4 shows the fit to the OGLE observation uncertainties.

$$\ln \sigma(M) = \begin{cases} 0.8728M - 18.55 & M \geq 15.35 \\ -5.809 & \text{otherwise} \end{cases} \tag{4}$$

This error model is implemented in `galens` by extending NumPy's `RandomState` class. An explicit seed is always provided when determining error to ensure that all data is reproducible.

A demonstration of the error values is shown in Fig. 5. Fig. 5a shows a simulated light curve and the 1σ and 2σ standard deviations. Fig. 5b shows an arbitrarily chosen microlensing light curve from the OGLE-IV survey [35]. We can see that error is highest at the highest magnitudes, corresponding to lower brightness. As brightness spikes, the error drops significantly. Our model approximates the error associated with the OGLE data well. Note that the two light curves have different base magnitudes, so the y-axes

are not directly comparable.

4.5.4 Mimicking Observation Outages

Aside from observation error, telescope observations, particularly those from ground-based telescopes, are subject to unpredictable data outages. Ground-based telescopes contain outages or exhibit reduced data quality due to bad weather, as a result of human error, or due to some other data management error. As such, we are also interested in evaluating the robustness of our models to missing values. A space telescope already significantly addresses these issues, as they are (clearly) not subject to weather conditions. However, there is still the possibility of missing values. Regardless of the cause of data outages, it is beneficial to understand the robustness of our model to missing data.

Our investigations and discussions indicated that when there is missing data, it is typically for an extended period, such a day, and not typically single data points. The cadence (time interval between observations) of the WFIRST telescope measurements will be every 15 minutes [28].

This problem could be made very complicated. However, we are only interested in the general behaviour of our model to data outages. We designed and implemented a framework to roughly mimic real-world data loss in a way that is reproducible, easy to understand and somewhat realistic. The assumptions of this model are as follows.

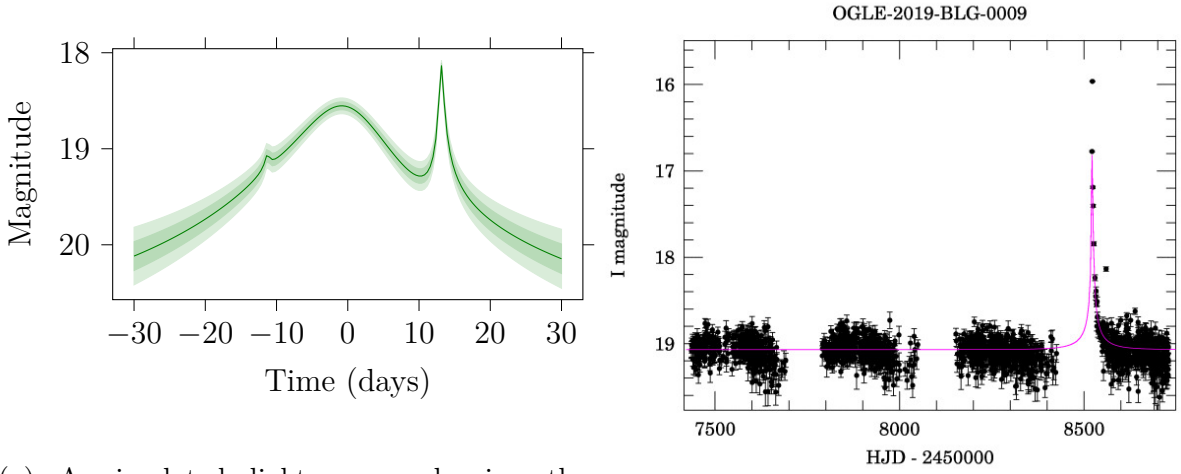
1. Data is lost only in chunks of a full day, and
2. each day in the complete time-series is lost with the same probability p.

Under these assumptions, we produced the following algorithm. Each day of the time-series is removed with probability p. In this way, each day is removed according to a Bernoulli trial, X, where

$$X \sim \text{Bernoulli}(p).$$

The parameter p, then represents the severity of missing data in that time-series on average, with $p = 0$ meaning no data is missing, $p = 0.1$ would mean each day has a 10% chance of being lost and so on. This approach allows us to evaluate the robustness of our model to data outages as a function of just one parameter, p.

The advantages of this model are that it is simple to implement, and simple to communicate to a subject matter expert, while still being a reasonable approximation to data conditions.



(a) A simulated light curve showing the 1σ and 2σ uncertainty windows at different magnitudes.

(b) A microlensing event from the OGLE-IV survey [35].

Figure 5: A comparison of simulated and real data, showing the dependence of error on magnitude. In each, we see that at high brightnesses, the error is lower than at fainter magnitudes.

Most machine learning models are not able to cope with missing values, so it is then necessary to develop a solution for dealing with the gaps in the time-series this removal creates. None of our models make predictions on the time-series directly. Instead, predictions are made on features extracted from these time-series. Many of the extracted features are also not able to deal with missing values, and two possible solutions arise. We could limit ourselves to machine learning models which can handle missing values, such as XGBoost, interpolate the missing values in some way (say, using the mean or a linear fit to the start and end of the gaps), or, we could concatenate the remaining data directly. Previous experience with `tsfresh` indicated that better results are obtained if no interpolation is performed; features are extracted from the concatenated time-series. We opted for the latter on this basis, and because designing an interpolation technique would unnecessarily introduce further assumptions and complexity to our modelling approach. See later sections for a discussion of robustness of the predictions to this method.

4.6 Feature Extraction

4.6.1 FRESH Algorithm

Physicists to date have been unable to manually determine a standard set of features to represent and characterise microlensing events. One way to sidestep this problem would be to utilise methods which do not require this kind of feature extraction, such as Neural Networks, which have been demonstrated to work in Physics applications [1]. However, the cost of such techniques is the loss of interpretability. A stated goal of this research is to determine a set of characteristics which can be used to inform an algorithm while maintaining some interpretability. For this to occur, we need to be able to report, explain, and justify the characteristics of the light curves which are most relevant for prediction. In this project, the 'customers' are physicists, and this interpretability is a critical metric of success. In light of this, we seek a set of time-series features, as few as possible, which are able to perform the task of light curve characterisation. In the machine learning paradigm, this problem is known as feature engineering.

Feature-based time-series analysis has been shown to provide performance and interpretability across a number of disciplines [12]. The presentation of the FRESH algorithm for this purpose by Christ et al. [7] has received over 60 citations at the time of writing in a broad range of research domains. We argue that the goals of this project, the structure of the data, and the advantages conferred by the technique demonstrate that a feature engineering approach is worthy of exploration.

The data of arbitrary microlensing events are of variable length, possibly non-uniformly spaced, non-periodic, and with different baselines. We seek a sort of 'alphabet', a standardised description of microlensing light curves which may evolve and adjust over time. For this purpose, we use the FRESH algorithm for reasons discussed [7]. The FRESH algorithm works by applying a pre-determined set of feature extractors to each time series and returning a fixed-length set of time series features for each time series.

A time series feature is the output of a function which maps the complete time series onto a single value, or fixed-length vector of values. A simple example is the 'maximum', a function which takes a full time series and returns one value, the maximum. Other more complex functions may include Fourier transformations, wavelet transformations, quantiles, and moments.

When each feature has been extracted for the time-series, a hypothesis test is conducted to determine if the feature is relevant for predicting the target variable of that feature. A comprehensive discussion of our feature selection techniques follows.

4.6.2 `tsfresh` Package

The implementation of the FRESH algorithm we make use of is the Python package `tsfresh` [6]. This Python package is well maintained and has been shown to work in a wide variety of applications across various industries. It has received over 4,000 stars on

GitHub. The strength of this algorithm is that it is a general technique and is not locked to one industry or type of problem.

The `tsfresh` package implements functions to extract nearly 700 features from every time-series. It also includes an implementation of the hypothesis tests we make use of in our model. The set of feature extractors implemented in this package have been collected from a variety of research disciplines, from finance to physics. Furthermore, included in this package are features identified as promising avenues of exploration, notably wavelet transformations. Wavelets are non-periodic, localised functions, like microlensing observations. Therefore, we argue that `tsfresh` includes features that would have been part of a more manual, targeted approach. We find that a wavelet component is identified as a relevant feature by our feature selection approaches.

Most machine learning models allow one to inspect the most important features, and in some cases, identify how they are used to arrive at a prediction. We can then look up the documentation of these features, and their meaning can be expressed to, and interpreted by, domain experts. In physics applications, in particular, this is very valuable, as physicists are frequently concerned with how a model arrives at a prediction. If this analysis is performed, we can also begin to understand these features and their reasons for effectiveness, thus gaining extra qualitative insight into the problem, which can lead to further understanding and solutions.

4.7 Feature Selection

The process of identifying a small set of time-series features for successfully representing microlensing light curves is termed feature selection. The nature of the FRESH algorithm is that a large number of features are calculated, and hypothesis testing and selection techniques are used to identify an appropriate subset of them. Alternatively, all nearly 700 features could be used directly in a machine learning model. However, this is not a good idea for several reasons. Firstly, using that many features can easily lead to overfitting, as there are many degrees of freedom for the model to consider. Secondly, calculating so many features can be an expensive process in itself for large time-series. If the process is too expensive, it may defeat our goal of developing an efficient, fast method of analysing microlensing light curves. Finally, with that many features, it would be impossible to develop an intuitive understanding or interpretation of how any model would work. In such a case, it would be difficult to trust, explain or troubleshoot the model.

4.7.1 Hypothesis Testing

This first step in feature selection involves performing a series of univariate hypothesis tests for feature relevance, and then accepting or rejecting features based on the resulting p-values. For this purpose, we use the Benjamini-Yekutieli procedure [2].

The first step is to undertake a Kendall-Tau rank test [17] for each feature. The null hypothesis, H_0, is that the feature is not useful for predicting the target (vector of s) from the feature. The alternative hypothesis is that it is useful for predicting the target. This test produces a p-value for each feature.

Next, we undertake the Benjamini-Yekutieli procedure [2], which is a method for multiple hypothesis testing. It is used to reject or keep features based on their p-values while favouring the elimination of features. It operates by considering a 'false discovery rate' (FDR), which is the rate at which we may expect on average to identify features as relevant falsely. This allows us to control the number of features selected, and ensure there is a known probability that features which are irrelevant leak into our model.

In most experiments, an overwhelming number (generally hundreds of the $\sim$700) were returned with very low p-values, sometimes with as many as 100 with p-values of machine zero. This meant even for extremely low false discovery rates, as low of 10^{-12}, hundreds of features may still be identified as relevant. While this does eliminate some features, it

indicates that many of the features we calculated may be useful for making predictions.

This result has a few consequences. Firstly, it provides early indications that the modelling approach we are taking may be successful. If many features are relevant, it means the general feature extraction approach is likely to yield a model which is capable of accurate predictions. Unfortunately, the problems with large numbers of features persist. It is then necessary to undertake further methods of feature selection to develop an interpretable, fast model.

The Benjamini-Yekutieli procedure was a precursor to all further feature selection discussed below. Operating in this way ensures that we are only considering deemed relevant according to fundamental relevance tests, reducing the chance of overfitting.

4.7.2 Random Forest Feature Importance

Here, we consider the use of Random Forest as a method of feature selection. Use of this algorithm for prediction of our target variable is discussed in Sec. 5.2.

Once we have eliminated features which fail the univariate hypothesis test and subsequent feature elimination procedure in `tsfresh`, we can train a Random Forest estimator with the remaining features. Random Forest roughly works by fitting many different models (decision trees) and uses this large ensemble of models (the 'forest') to make a prediction. We can fit large forests to the remaining relevant features. When we do this, the prediction results are discussed in Sec. 5. We use the Random Forest implementation in the scikit-learn package [27]. This implementation allows access to a feature importance vector which gives the relative importance of all features passed to the model. The sum of the total relevances is always one across all features. We then fit a new model keeping only some number of the top features. As seen in Fig. 6, the feature importance for the first few (approximately 10-15) features is quite large, and the remaining features drop off in importance quite quickly.

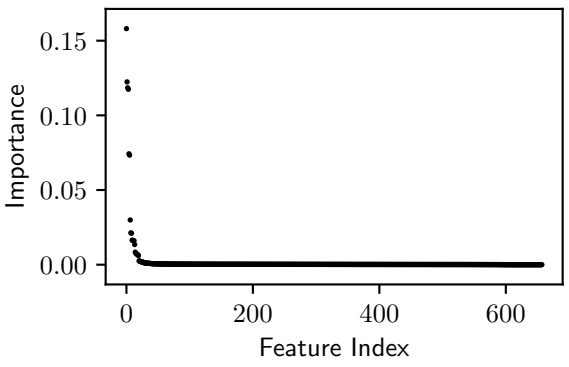


Figure 6: Plot of feature importance in the Random Forest model.

Once we did this, we then fitted a new Random Forest model keeping only the top 14 most relevant time-series features in one instance, and seven in another instance when used in association with Recursive Feature Elimination. The quality of this 14-feature, significantly reduced model, was still extremely high.

We ensured that retained features were mostly uncorrelated. This is because we wish to keep the number of features as low as possible to maintain speed and control model complexity. For each feature to add new information and value, it is then necessary that there is no strong correlation (positive or negative) with other variables in the model.

It is worthy of note that just because the successful Random Forest model maintained certain features, it does not mean that this is the only combination of features that work. The fact that the hypothesis testing procedure retained over 600 features in nearly all experiments conducted indicates that there is likely to be a vast number of different feature combinations which could be used to produce highly accurate models.

The conclusions drawn here are that there is potential for a few features to be sufficient to capture the challenges of our problem and still perform well.

4.7.3 Recursive Feature Elimination

The final method of feature selection used was Recursive Feature Elimination (RFE), a conceptually simple procedure. It works by fitting a model (parameterised by the user) using all features. It then drops the feature(s) deemed least important by the model and refits, iterating in this fashion until a desired number of features or is reached, or a performance metric drops below a given value. The features may be eliminated by dropping those deemed least important (for example in a Random Forest) or with smallest coefficients (as in a linear model). This method was used to obtain the seven features utilised in our Bayesian approach. Using this method, it is still important to ensure that remaining features are uncorrelated. The model which informs the RFE process can be any with some kind of quantification of the importance of different features.

4.7.4 Most Relevant Features

Our final models are based on the calculation of seven time-series features which are best able to make predictions across the Random Forest and Bayesian models. The features are not strongly correlated in our test data sets. This is a product of features only being kept if they strongly improve the quality of the solution. These features were determined by using Recursive Feature Elimination using Random Forest feature importances as the engine, further discussed in Sec. 5. A description of each of the features follows.

Mean Autocorrelation

This feature is the mean of the autocorrelations from 1 to 40. The autocorrelation function in `tsfresh` is defined for a given lag, l, as

$$R(l) = \frac{1}{(n-l)\sigma^2} \sum_{i=1}^{n-l} (x_i - \mu)(x_{t+l} - \mu). \tag{5}$$

The mean autocorrelation from $l = 1, ..., m$ is then calculated. We use $m = 40$.

$$\overline{R} = \frac{1}{m} \sum_{l=1}^{m} R(l). \tag{6}$$

In `tsfresh`, the feature name is `agg_autocorrelation__f_agg_"mean"__maxlag_40`.

C3

'C3' was proposed by Schreiber & Schmitz [33] as a measure of non-linearity in a time-series. For a given time-series $\{x_i\}$ of length n, it is calculated as below for a given lag, l.

$$\mathrm{C3}(l) = \frac{1}{n-2l} \sum_{i=1}^{n-2l} x_{i+2l} x_{i+l} x_i \tag{7}$$

Our model uses this time-series feature for a lag $l = 3$.
In `tsfresh`, the feature name is `c3__lag_3`.

Mean Absolute Change

This feature is the mean of the differences between each subsequent values in the time-series, $\{x_i\}$ of length n.

$$\mathrm{MAC} = \frac{1}{n} \sum_{i=1}^{n-1} |x_{i+1} - x_i| \tag{8}$$

In `tsfresh`, the feature name is `mean_abs_change`.

Range Count

This feature is the number of values between a minimum and maximum, in our case, in the interval $[-1, 1)$.

In `tsfresh`, the feature name is `range_count__max_1__min_-1`.

Change Quantiles

Here, the mean of absolute differences between values in the 20th percentile of the time-series is calculated. If both values in the time-series fall in this percentile range, their absolute difference is accounted for in this mean. In effect, this quantifies the intensity of 'peaks' seen in the light curve.

In `tsfresh`, the feature name is `change_quantiles__f_agg_"mean"__isabs_True__qh_0.2__ql_0.0`.

Aggregate Linear Trend

This feature calculator works by first fitting lines to chunks of the time-series using standard linear regression. Here, we use chunks of 50 data points. The minimum slope value of linear regressions are returned. This calculator assumes the time-series is uniformly sampled. The general method of aggregated linear trends is a product of research in Economics [24].

In `tsfresh`, the feature name is `agg_linear_trend__f_agg_"min"__chunk_len_50__attr_"slope"`.

Continuous Wavelet Transform

This feature is calculated by performing a continuous wavelet transform (CWT) with Ricker or Mexican Hat wavelets. They are defined as follows [31].

$$\psi(x) = \frac{2}{\sqrt{3a}\pi^{1/4}} \left(1 - \frac{x^2}{a^2}\right) \exp\left(-\frac{x^2}{2a^2}\right) \tag{9}$$

A CWT with width parameters 2, 5, 10, and 20 is performed. The first coefficient of the width 5 wavelet is used in our case.

In `tsfresh`, the feature name is `cwt_coefficients__widths_(2, 5, 10, 20)__coeff_0__w_5`.

5 Results

5.1 Outline

The primary result of this study is the development of a process by which microlensing events can be represented in a reduced feature space, and their parameters predicted. We have validated this approach by reducing the microlensing parameter space to varying just the separation parameter s. We present two successful models for these predictions. First, we demonstrated that a Random Forest model makes excellent predictions and is robust to noise and data outages. Second, we implemented and fitted a Bayesian Linear Regression model. This model provides posterior probability distributions for predictions and also boasts strong performance. However, it is not robust to noise or data outages. The value of this model is that it enables an active learning procedure, a machine learning concept whereby the machine can determine areas of least knowledge automatically, and query a physical simulator to produce simulations to fill these gaps. We have validated this model under the assumptions about the microlensing events we have presented.

Fig. 7 shows an outline of the process we have designed for modelling and predicting microlensing events. The first phase is to simulate a series of microlensing light curves sampled uniformly from the parameter space and extract a large number of features that are statistically significant for predicting the separation parameter s. These simulations are used to fit an initial model. We obtained the first significant result by demonstrating that a time-series feature extraction method can produce high-quality predictions. This is discussed in Sec. 5.2, where we used a Random Forest model to show that time-series features, even very few features, may be used to predict the separation parameter.

The accuracy of the Random Forest model led us to the conclusion that a polynomial model of a small set of features may offer strong prediction performance, also. By enhancing existing code by Kempa-Liehr [16] and implementing the procedure for Bayesian Linear Regression described by Bishop [3], we were able to develop a polynomial model based on seven time-series features that can produce good predictions with an associated probability distribution. Furthermore, producing this model from the data is a fast, analytic procedure based on our implementation of material presented by Bishop [3].

The refinement of the model by efficiently performing new simulations forms the second part of the methodology. This component is enabled by our construction of probability distributions for predicted parameters.

The use of Bayesian Linear Regression also confers all the associated benefits of Bayesian statistics [21]. In future, an expanded version of this model may form part of a suite of models, all of which could be used and comprehensively compared to produce a characterisation of a new gravitational microlensing event, which forms the final stage of the methodology. This stage consists of predicting the parameters of new microlensing event as they are observed by WFIRST and subsequently identifying likely lens parameters, which can be refined by physicists.

5.2 Random Forest Model

The first model we present is a Random Forest, sitting atop the feature extraction process we discussed earlier. This model was used to demonstrate that a time-series feature method can be used to make accurate predictions of our target variable at all. Other approximation techniques for gravitational microlensing observations work well for the ‘simpler’ parts of the parameter space (those with less extreme non-linearities) but suffer from poor performance in the complex sections [18].

Our random forest model on 14 time-series features achieved an explained variance of 99.9%.

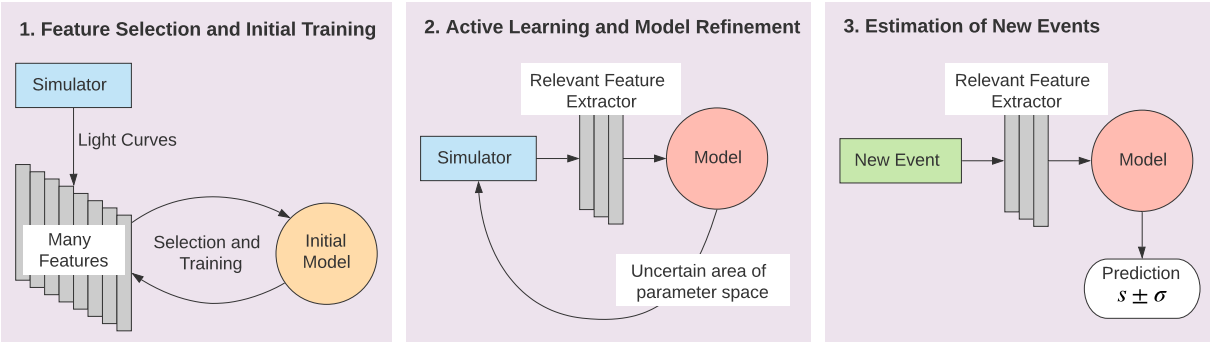


Figure 7: An outline of the methodology we have developed for predicting parameters of new light curves. The first step involves simulating a range of events and extracting a large number of features to produce an initial model. Secondly, that model is refined, and the number of features reduced. Finally, when new gravitational microlensing events are observed, the model is used to predict their parameters with uncertainty.

5.2.1 Random Forest Algorithm

Random Forest is a general, widely used, non-linear regressor with well-tested implementations. It is easy to parameterise and under some parameterisations lends itself well to interpretation by displaying one or more of the decision trees used to produce its outputs. By producing an ensemble of decision trees, the predictions of this regressor are non-linear with respect to the time-series features. With such highly non-linear light curves, this indicated it might be suitable for our purposes.

Another benefit of this model is that it can be extended to multiple targets quite easily. However, this feature is out of the scope of this project, but it is an important quality of the estimator as further work is done with this model. Random Forest also has a well-tested Python implementation in scikit-learn [27].

Random Forest is easy to parameterise. The most critical parameters are the maximum depth of each decision tree and number of decision trees. The scikit-learn package offers sensible defaults for these parameters which can be used in the first instance [27]. The fitting of the algorithm is then reasonably fast, as is prediction (as long as the parameters discussed are not selected to be too large).

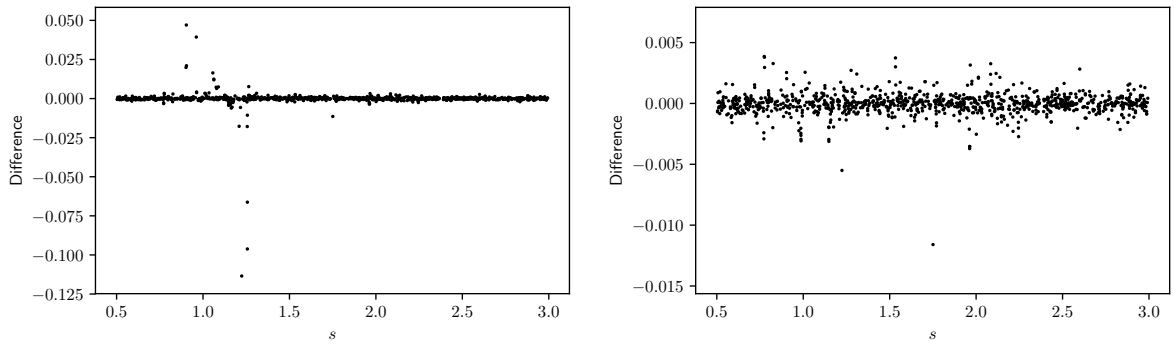
The downside of the algorithm is that it does not make predictions with uncertainties, and large forests can not be interpreted easily. As discussed in the feature selection section, a trained Random Forest makes accessible the importance of each feature given in the model relative to other features.

5.2.2 Fitting and Performance

We trained a Random Forest on 2,000 light curves simulated under the discussed assumptions, with the separation parameter uniformly sampled between 0.5 and 3.0. A further 1,000 light curves were used as a training set. All time-series used to train this model were produced without error distortion or the addition of any observation outages. Of the 696 features extracted from these time-series, 660 were deemed relevant by the Benjamini-Yekutieli procedure, with a false discovery rate of 10^{-15}. This is a larger than expected number of features deemed to be relevant by hypothesis testing. Fig. 6 shows the feature importances as determined by the Random Forest, ordered by feature importance. We can see the first few features have high importance, and this drops to near zero very quickly.

The first step in the process was to train a Random Forest of 10,000 decision trees, with no maximum depth using all 660 statistically significant features. The mean square error for this model was 0.999995 for both in and out-of-sample. These results are auspicious. However, the value of this model is the feature importance for each feature.

Of all 660 features, just 14 had a feature importance above 0.01. This appeared to be



(a) 14 feature Random Forest using feature importances

(b) 7 feature Random Forest with Recursive Feature Elimination

Figure 8: Residual plots as a function of s for two Random Forest models we developed. Both models have no maximum tree depth and 100 estimators.

a reasonable point to cut off the features, as features below that threshold are given to have very low importance, and 14 features is a reasonable number of features to build a model off regarding both speed and understanding.

We then trained a new model with just these fourteen features and with only 100 trees. Even for this reduced model, the out-of-sample explained variance was 99.9993% for out-of-sample predictions, meaning we can obtain good predictions from even a small number of features quite quickly. By observing the residual plot for this outcome, shown in Fig. 8a, we see confirmation of the quality of predictions. This is an interesting result because our technique offers high-quality results, even for challenging light curves, results we have not seen elsewhere. This demonstrates that it is possible to obtain good results using our general methodologies.

However, the features in this initially successful Random Forest model are determined only roughly in one sweep by feature importance. By utilising the Recursive Feature Elimination procedure, we were able to develop a more robust Random Forest using just seven features. A Random Forest was first trained on all statistically significant features. The least important feature was dropped, and the model retrained, and so on. By observing the residual plot for this outcome, shown in Fig. 8b, one can see that the maximum residual is reduced by approximately a factor of 10. The out-of-sample explained variance in this case is 99.9999%.

We have demonstrated that time-series features can be used to make accurate predictions of microlensing event parameters. These high-quality results from Random Forest indicate we may achieve success using a simpler model. It is for this reason, and because of the goal for uncertainty considerations that we also considered a Bayesian polynomial model based on a small number of features.

5.2.3 Robustness

The robustness of the most successful Random Forest model to the two distortions we present is an important part of its strength.

It is first worthy of note that this model has been trained on the raw output from simulations, and are undistorted. This is a reasonable assumption to make, as it makes sense to train the model on the best available data, and then test its performance against more realistic scenarios. The downside is that neither the feature selection nor the trained model is familiar with these distortions. This means there may be sets of features which are better capable of predicting the target under noisier conditions. Currently, we do not know how robust the features we have selected are to noise and gaps.

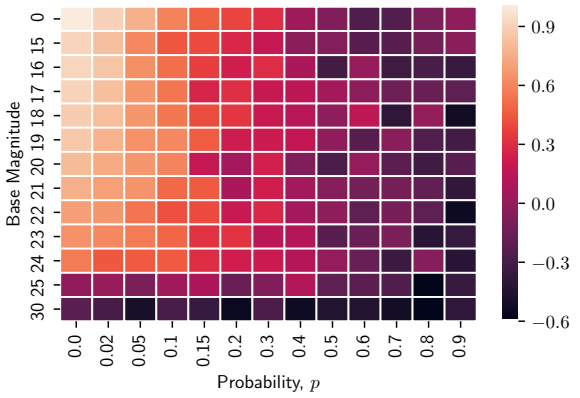


Figure 9: Out-of-sample explained variance scores at different base magnitudes and outage rates for the presented Random Forest.

The seven feature Random Forest model's performance under different base magnitudes and outage probabilities is presented in Fig. 9. Base magnitude corresponds to level of error, with 0 meaning no error was applied, and 15 indicates the error floor. High base magnitudes mean dimmer sources and greater levels of error. Probability p represents the probability of each day being removed, that is to say, the outage rate. Each point on the grid is a set of 100 microlensing light curves uniformly sampled from our parameter space. The error is then realised, and days removed with a probability p.

The most important region of Fig. 9 to consider is the top left. The error expected from the WFIRST telescope is expected to be much lower than that of the OGLE-IV survey used here. Furthermore, outages in space telescopes are more rare. As discussed, the actual error levels are yet to be determined but will probably not exceed the errors associated with a base magnitude of 19. Furthermore, with far fewer chances for data loss or interruption, a loss rate of 2% is likely to be reasonable. Within this range, the explained variance is still the moderate level of 75%. Beyond these values, the model breaks down as expected.

5.3 Bayesian Linear Regression

We will start by presenting the mathematical background to Bayesian Linear Regression and its motivations. We have implemented an analytic solution to the problem of Bayesian Linear Regression as detailed by Bishop [3]. Let us first note that by linear regression, we refer to a linear model with respect to time-series features and their polynomial combinations, not linear with respect to the raw time-series. This lifts the significant constraint of linear regression that we would have restricted our model to be a linear function of the microlensing event observations. Instead, our model is linear with respect to a series of fixed 'basis functions', our time-series features and their polynomial combinations. We model our target variable as being a linear function of the outputs of these basis functions.

This model was suggested by the excellent results of the Random Forest procedure. The strength of this approach is that we can obtain an analytic expression for the uncertainty in a prediction as a function of the given time-series. This has ramifications for the communications of predictions to domain experts and the development of an active learning algorithm. Linear regression also has the advantages that it is quick to train and provides a somewhat interpretable model.

The greatest strength of Bayesian Statistics is its modelling of uncertainty. Machine learning techniques such as Random Forest do not provide this information. Bayesian statistics will provide not only a prediction but a complete posterior probability distribution function. This is important as a perfect prediction cannot be achieved, and if this model is to be useful to physicists, uncertainty will be necessary. Furthermore, if this model is to be extended beyond the current parameter space, most light curves will lend themselves to multiple interpretations, and a full, multi-modal posterior distribution can be used to identify a variety of possibilities. Furthermore, an understanding of the Bayesian approach is intuitive. Physicists can identify physically feasible regions in parameter space with some probability (potentially a uniform one), the prior, and a model

is used to adjust probabilities throughout parameter space in light of the information, time-series features based on microlensing observations [21]. Furthermore, Bayesian Statistics itself has only recently risen to prominence, and many of its impacts have yet to be realised for many domains [14].

5.3.1 Theory and Implementation

Let us begin with a presentation of the theory we have implemented, that of Bayesian Linear Regression. This is a summary of the theory presented by Bishop [3].

We start by considering a series of M basis functions $\phi_j(\mathbf{x})$, where $\mathbf{x}$ is a vector representing one microlensing event. In our case, these basis functions are feature extractors or polynomial combinations of features. We wish to make predictions of an unknown s. The prediction is given the label $\hat{s}$. Predictions are linear combinations of these basis functions, as shown in Eqn. 10. Here, $\phi_j(\mathbf{x})$ is a single feature, and $\boldsymbol{\phi}(\mathbf{x})$ is a vector of features of length M.

$$\hat{s}(\mathbf{x}, \mathbf{w}) = \sum_{j=1}^{M} w_j \phi_j(\mathbf{x}) = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}) \tag{10}$$

We include a regularisation term in our linear model to encourage the values weight coefficients $\mathbf{w}$ to tend towards zero, thus favouring model simplicity unless supported by the observations. The regularisation term is controlled by the sum of squares of the weight vector. We are then trying to minimise the sum of squared error plus our regularisation term, giving the error function,

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \{s_n - \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}_n)\}^2 + \frac{\alpha}{2} \mathbf{w}^\top \mathbf{w}. \tag{11}$$

For each observation $\mathbf{x}_n$, $n = 1, \ldots, N$, the feature vector $\boldsymbol{\phi}(\mathbf{x}_n)$ is determined. These are compiled into the $N \times M$ 'design matrix' $\boldsymbol{\Phi}$. Differentiating Eqn. 11 and setting it to zero gives a minimum error at

$$\mathbf{w} = \left(\alpha \mathbf{I} + \boldsymbol{\Phi}^\top \boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^\top \mathbf{s}. \tag{12}$$

The noise precision parameter β is related to the base noise present in the data, and identifies the uncertainty floor for the model, where β is determined by

$$\frac{1}{\beta} = \frac{1}{N} \sum_{n=1}^{N} \{t_n - \mathbf{w}^\top \phi(\mathbf{x}_n)\}^2. \tag{13}$$

Bishop shows that the posterior distribution for our weight vector will be a multivariate normal,

$$p(\mathbf{w}|\mathbf{s}) = \mathcal{N}(\mathbf{w}|\mathbf{m}_N, \mathbf{S}_N), \tag{14}$$

with mean $\mathbf{m}_N$ and variance $\mathbf{S}_N$. Bishop goes on to discuss that we can set the prior distribution for the weights to be a zero-mean isotropic normal, controlled by one precision parameter α, provided the choice of α is sufficiently small to enable a broad prior,

$$p(\mathbf{w}|\alpha) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I}), \tag{15}$$

where $\mathbf{I}$ is the $M \times M$ identity matrix. Bishop shows that under these conditions, we have

$$\mathbf{S}_N^{-1} = \alpha \mathbf{I} + \boldsymbol{\Phi}^\top \boldsymbol{\Phi}. \tag{16}$$

We now also arrive at a Gaussian for the predictive distribution,

$$p(s|\mathbf{s}) = \mathcal{N}(s|\mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}), \sigma_N^2(\mathbf{x})), \tag{17}$$

with variance given by

$$\sigma_N^2(\mathbf{x}) = \frac{1}{\beta} + \phi(\mathbf{x})^\top \mathbf{S}_N \phi(\mathbf{x}). \tag{18}$$

This procedure is implemented in our scikit-learn style estimator `BaysianLinearRegression`. See Appendix A for documentation of this code.

5.3.2 Polynomial Features

Bayesian Linear Regression was used because a polynomial model of features was suggested by the quality of Random Forest predictions. To allow this method to work with Linear Regression, the polynomial features must be constructed from the existing features.

These existing features are first standardised. That is to say, each feature is shifted by the sample mean of that feature, and scaled by the standard deviation of the feature. This ensure features are on similar orders so one feature does not dominate another when polynomial combinations are created.

The polynomial combinations are created by taking the initial $N \times M$ feature matrix where N is the number of samples, and $M = 7$ features, and calculating polynomial combinations. We generate up to third order polynomial features. For example, a feature matrix of the form $[a, b]$ will become a matrix of the form $[1, a, b, ab, a^2, b^2, ab^2, a^2b, a^3, b^3]$. This new matrix then becomes the feature matrix for the Bayesian Linear Regression.

The same seven features determined using Recursive Feature Elimination in the Random Forest section are utilised in this model, also. The reasons for this are two-fold. First, it aids in comparison of the two models, as well as improves interpretability. Second, the performance of the Linear Regression with Random Forest determined features was better than those that could be determined by linear regression.

5.3.3 Performance

Once the polynomial feature matrix was determined, we were able to fit our model as detailed above. Fig. 10 shows the in-sample and out-of-sample residuals for this model. We can determine by inspection that this model also produces highly accurate predictions of the separation parameter s on new data. The out-of-sample explained variance was 99.99733%.

The most important development here, however, is the introduction of uncertainty. Every estimate is associated with a probability distribution, allowing us to quantify the confidence in the results of every prediction. As expected, uncertainties are higher in regions of the separation parameter s already identified to be particularly challenging, near the Einstein radius of the system ($s = 1$).

5.3.4 Robustness

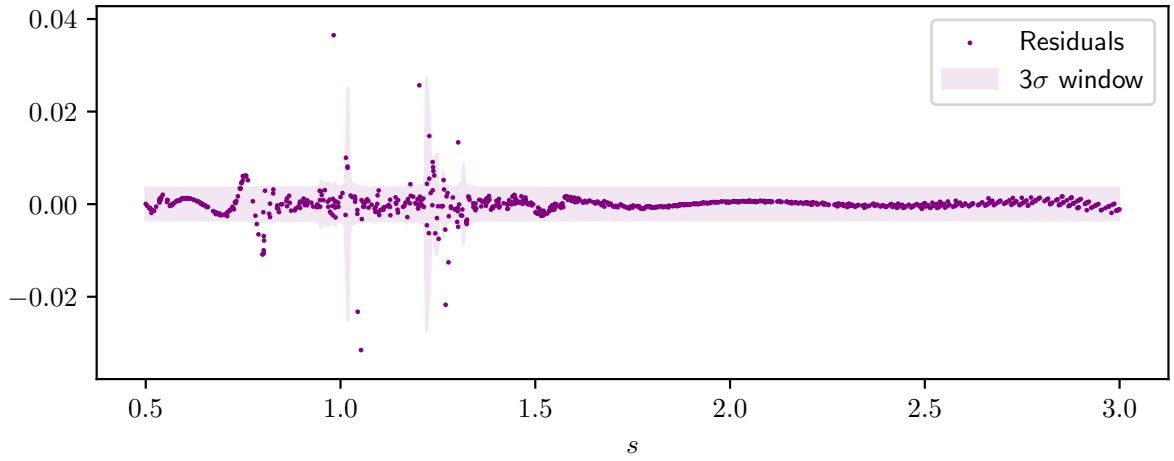
The Bayesian Linear Regression model is not robust to noise or data outages. The model was trained on data that was not artificially distorted by noise or outages, and achieves powerful results under these conditions. However, addition of even the minimum noise or a p =1% daily probability of a data outage effectively eliminates performance, and estimations are intolerable.

Further research will investigate methods to improve the robustness of this model to more realistic noise levels. This may be done utilising the feature selection and fitting methodology on data under these conditions to seek features more robust to these problems. However, currently, this performance represents the disadvantage of the Bayesian model.

5.3.5 Applications for Active Learning Algorithm

The real strength of this model is that it provides uncertainty alongside parameter estimations, successfully completing a goal of this study. This means that this Bayesian

(a) Out-of-sample residuals



(b) In-sample residuals

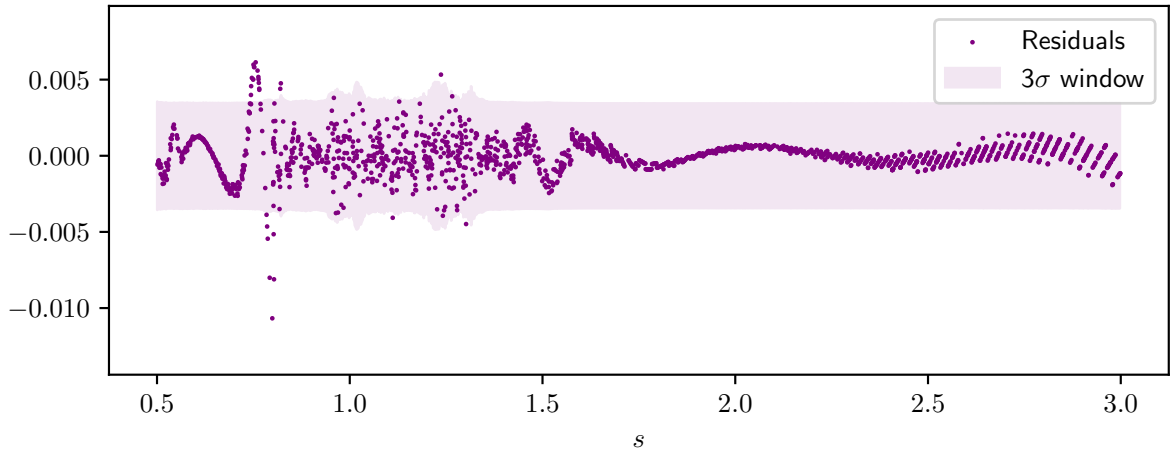


Figure 10: Residual plot and 3σ uncertainty windows for the Bayesian Linear Regression model fitted with third order polynomial terms on seven time-series features.

model may be utilised in an active learning procedure. In effect, the uncertainty quantifies the 'knowledge' that the model has about a particular region of parameter space. By identifying these regions, simulations may be performed that optimally address this lack of knowledge. As research on this problem extents this approach to a broader parameter space, this may become a crucial component in making simulation of the microlensing parameter space a computationally tractable problem.

Furthermore, as more simulations are added to the model, the Bayesian nature of this problem allows the posterior distribution from before the simulation to form the prior for the subsequent model. This will have the effect of reducing fitting time as new simulations are added.

6 Discussion

6.1 Key Findings

Here, we summarise the key outputs and findings of this study. In Sec. 6.1.1, we address the two successful models, a Random Forest and a Bayesian Linear Regression approach. This demonstrates that a time-series feature approach can successfully predict the separation parameter. In Sec. 6.1.2, we address how we can model probability distributions of the separation parameter and the ramifications of this for future research. In Sec. 6.1.3, we discuss our stand-alone code repository as a tool for reproducing and expanding our results. In Sec. 6.1.4, we discuss the methodology we have developed, including model validation techniques.

6.1.1 Time-series feature approach

We have shown that it is possible to predict the separation parameter of binary sources in microlensing events using a time-series feature extraction approach. Using seven features of simulated microlensing light curves, both a Random Forest and Linear Regression with polynomial components can achieve high prediction accuracy, in excess of 99.99% explained variance. This quality is retained even in challenging parts of the parameter space, where classical fitting techniques do not perform well [18]. This represents a significant finding because it demonstrates the potential of this approach, that a time-series feature extraction method can achieve where other methods have not, showing a path forward for further research, discussed in Sec. 6.2. Furthermore, the Random Forest algorithm maintains performance under the levels of error we can estimate to be present in WFIRST observations

6.1.2 Modelling Uncertainty

The Bayesian polynomial model we have presented includes an appropriate modelling of uncertainty in the estimation of the separation parameter from light curves. Firstly, this allows one to understand the degree of confidence the model has in making predictions, providing guidance on the follow-up analysis that may be needed. Secondly, this modelling of uncertainty enables the implementation of an active learning procedure to select new locations in parameter space to undertake simulations optimally. This provides a blueprint for exploring the vast parameter space of general gravitational microlensing events that will be the subject of further efforts.

6.1.3 Code Repository – `galens`

In addition to the powerful results of our models, a key output of this project has been our stand-alone Python code repository, `galens`. This repository is well documented and can be made available to outside researchers. In following the best practices of data science, we offer the ability to reproduce our results in future and enhance the work we have done. This code repository grants access to a variety of algorithms and functions to quickly simulate new light curves, distort them with custom error models, and perform experiments. Implemented in this package is the error model of the OGLE-IV microlensing survey and a simple algorithm for mimicking outages present in observations. The documentation of this repository is thorough and clear and is available in Appendix A.

6.1.4 Methodology

We have created a methodology for fitting algorithms to recover microlensing parameters from observations. The general methodology is outlined in Fig. 7 and is based on the FRESH algorithm [7], which has been demonstrated to work in a variety of contexts. The features identified as useful to a model can be explained and interpreted by physicists utilising this method. Furthermore, we have validated this methodology on the reduced parameter space by making excellent predictions of the separation parameter.

It is this methodology which will be extended in future research to envelop the wider problem of identifying parameters in arbitrary microlensing event observations.

6.2 Outlook

6.2.1 Increase the Parameter Space

With our initial models validated for the separation parameter, the most important next phase is to widen the study to the entire (or, at least, larger) parameter space of gravitational microlensing events. There are more parameters to consider for the binary systems, as well as simulations of multi-planet systems, and other more exotic systems, such as planets orbiting binary star systems. As the parameter space expands, we expect additional challenges to arise that have not presented themselves under our assumptions. Such effects to consider may be multi-modal posteriors, where we can quantify the confidence in more than one interpretation of a single light curve. There are also additional Bayesian techniques discussed by Bishop [3] that could be used to compare different models under different assumptions.

6.2.2 Robustness

The robustness of our Bayesian Linear Regression model to noise and outages is poor. To improve the value of this model, additional work is required to improve its performance in the presence of real data conditions. This could take place by adjusting the criteria by which features are selected, where predictors are favoured on the quality improvement they offer to the model, while also advancing predictors robust to the problems seen in real microlensing data.

7 Summary and Conclusions

As per the aims for this project, we have developed and validated a methodology for recovering microlensing event parameters from their observations. We began by demonstrating that time-series feature analyses can yield time-series features which can inform a successful algorithm. One result is that a Random Forest with using seven such time-series features can achieve an explained variance in excess of 99.99% for out-of-sample predictions of the separation parameter. In particular, this algorithm works successfully in the challenging regions of parameter space where the separation parameter is close to the Einstein ring radius of the planetary system. Secondly, we demonstrated that Bayesian Linear Regression could be used with polynomial combinations of the same seven features to make predictions of the separation parameter, with an explained variance also exceeding 99.99% out-of-sample. Additionally, this Bayesian model appropriately models a probability distribution for the resulting estimations.

The success of these models validates our methods with the reduced parameter space identified for this study. This methodology consists of extracting statistically significant time-series features from an initial set of simulations and refining this set of models. In doing so, we have demonstrated that highly accurate models can be fitted from as few as seven time-series features. The modelling of uncertainty enables an elegant, efficient method for refining this model by optimally exploring parameter space. This provides evidence that this method could be expanded to the larger parameter space and offer fast, high-quality predictions of WFIRST observations of microlensing events.

8 References

[1] S Baehr, O Sander, M Heck, C Pulvermacher, M Feindt, and J Becker. "Online-Analysis of Hits in the Belle-II Pixeldetector for Separation of Slow Pions from Background". In: *Journal of Physics: Conference Series* 664.9, 092001 (Dec. 2015), p. 092001. DOI: 10.1088/1742-6596/664/9/092001.

[2] Yoav Benjamini, Daniel Yekutieli, et al. "The control of the false discovery rate in multiple testing under dependency". In: *The annals of statistics* 29.4 (2001), pp. 1165–1188. DOI: 10.1214/aos/1013699998.

[3] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[4] Ian A Bond, A Udalski, M Jaroszyński, NJ Rattenbury, B Paczyński, I Soszyński, L Wyrzykowski, MK Szymański, M Kubiak, O Szewczyk, et al. "OGLE 2003-BLG-235/MOA 2003-BLG-53: A Planetary Microlensing Event". In: *The Astrophysical Journal Letters* 606.2 (May 2004), pp. L155–L158. DOI: 10.1086/420928. eprint: astro-ph/0404309.

[5] Valerio Bozza, Etienne Bachelet, Fran Bartolić, Tyler M Heintz, Ava R Hoag, and Markus Hundertmark. "VBBINARYLENSING: a public package for microlensing light-curve computation". In: *Monthly Notices of the Royal Astronomical Society* 479.4 (Oct. 2018), pp. 5157–5167. DOI: 10.1093/mnras/sty1791. arXiv: 1805.05653 [astro-ph.IM].

[6] Maximilian Christ, Nils Braun, Julius Neuffer, and Andreas W. Kempa-Liehr. "Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh – A Python package)". In: *Neurocomputing* 307 (2018), pp. 72–77. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2018.03.067. URL: http://www.sciencedirect.com/science/article/pii/S0925231218304843.

[7] Maximilian Christ, Andreas W. Kempa-Liehr, and Michael Feindt. "Distributed and parallel time series feature extraction for industrial big data applications". In: *CoRR* abs/1610.07717 (2016). arXiv: 1610.07717. URL: http://arxiv.org/abs/1610.07717.

[8] *Confirmed Planets Table*. URL: https://exoplanetarchive.ipac.caltech.edu/.

[9] Zoheyr Doctor, Ben Farr, Daniel E Holz, and Michael Pürrer. "Statistical gravitational waveform models: What to simulate next?" In: *Physical Review D* 96.12, 123011 (2017), p. 123011. DOI: 10.1103/PhysRevD.96.123011. arXiv: 1706.05408 [astro-ph.HE].

[10] Shawn Domagal-Goldman, Nancy Y Kiang, Niki Parenteau, David C Catling, Shiladitya DasSarma, Yuka Fujii, Chester E Harman, Adrian Lenardic, Enric Pallé, Christopher T Reinhard, et al. "Life Beyond the Solar System: Remotely Detectable Biosignatures". In: *arXiv e-prints*, arXiv:1801.06714 (2018), arXiv:1801.06714. arXiv: 1801.06714 [astro-ph.EP].

[11] Michael Feindt. "A Neural Bayesian Estimator for Conditional Probability Densities". In: *arXiv e-prints*, physics/0402093 (Feb. 2004), physics/0402093. arXiv: physics/0402093 [physics.data-an].

[12] Ben D. Fulcher. "Feature-based time-series analysis". In: *CoRR* abs/1709.08055 (2017). arXiv: 1709.08055. URL: http://arxiv.org/abs/1709.08055.

[13] B Scott Gaudi and Andrew Gould. "Planet parameters in microlensing events". In: *The Astrophysical Journal* 486.1 (1997), p. 85. URL: https://iopscience.iop.org/article/10.1086/304491.

[14] Zoubin Ghahramani. "Probabilistic machine learning and artificial intelligence". In: *Nature* 521 (May 2015), 452 EP -. URL: https://doi.org/10.1038/nature14541.

[15] Elizabeth Howell. *Hubble Telescope Discovers a Light-Bending 'Einstein Ring' in Space*. Apr. 2018. URL: https://www.space.com/40255-hubble-telescope-einstein-ring-photo.html.

[16] Andreas W. Kempa-Liehr and Alex Kennedy. *DSLab - Data Science Laboratory*. https://stash.auckland.ac.nz/projects/DSEA/repos/dslab. 2017–2019.

[17] Maurice G Kendall. "The treatment of ties in ranking problems". In: *Biometrika* 33.3 (1945), pp. 239–251.

[18] Somayeh Khakpash, Matthew Penny, and Joshua Pepper. "A Fast Approximate Approach to Microlensing Survey Analysis". In: *arXiv e-prints*, arXiv:1902.08092 (2019). arXiv: 1902.08092 [astro-ph.EP].

[19] Seung-Lee Kim, Chung-Uk Lee, Byeong-Gon Park, Dong-Jin Kim, Sang-Mok Cha, Yongseok Lee, Cheongho Han, Moo-Young Chun, and Insoo Yuk. "KMTNET: a network of 1.6 m wide-field optical telescopes installed at three southern observatories". In: *Journal of the Korean Astronomical Society* 49.1 (2016), pp. 37–44. DOI: 10.5303/JKAS.2016.49.1.037.

[20] Donald Ervin Knuth. "Literate programming". In: *The Computer Journal* 27.2 (1984), pp. 97–111.

[21] John K. Kruschke and Torrin M. Liddell. "Bayesian data analysis for newcomers". In: *Psychonomic Bulletin & Review* 25.1 (Feb. 2018), pp. 155–177. ISSN: 1531-5320. DOI: 10.3758/s13423-017-1272-1. URL: https://doi.org/10.3758/s13423-017-1272-1.

[22] David W. Latham and B. Scott Gaudi. "Microlensing Observations in Astrophysics". In: *Encyclopedia of Astrobiology*. Ed. by Ricardo Amils, Muriel Gargaud, José Cernicharo Quintanilla, Henderson James Cleaves, William M. Irvine, Daniele Pinti, and Michel Viso. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 1–1. ISBN: 978-3-642-27833-4. DOI: 10.1007/978-3-642-27833-4_1850-2. URL: https://doi.org/10.1007/978-3-642-27833-4_1850-2.

[23] Jack J Lissauer and Imke De Pater. *Fundamental Planetary Science: Physics, Chemistry and Habitability*. Cambridge University Press, 2013.

[24] Massimiliano Marcellino. "Linear aggregation with common trends and cycles". In: *Research in Economics* 54.2 (2000), pp. 117–131. ISSN: 1090-9443. DOI: https://doi.org/10.1006/reec.1999.0229. URL: http://www.sciencedirect.com/science/article/pii/S1090944399902296.

[25] Przemek Mróz, Andrzej Udalski, David P Bennett, Yoon-Hyun Ryu, Takahiro Sumi, Yossi Shvartzvald, Jan Skowron, Radosław Poleski, Paweł Pietrukowicz, Szymon Kozłowski, et al. "Two new free-floating or wide-orbit planets from microlensing". In: *Astronomy & Astrophysics* 622 (2019), A201. DOI: 10.1051/0004-6361/201834557. arXiv: 1811.00441 [astro-ph.EP].

[26] National Solar Observatory Sacramento Peak. *Magnitude*. June 2001. URL: https://web.archive.org/web/20080206074842/http://www.nso.edu/PR/answerbook/magnitude.html.

[27] Fabian Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *CoRR* abs/1201.0490 (2012). arXiv: 1201.0490. URL: http://arxiv.org/abs/1201.0490.

[28] Matthew T Penny, B Scott Gaudi, Eamonn Kerins, Nicholas J Rattenbury, Shude Mao, Annie C Robin, and Sebastiano Calchi Novati. "Predictions of the WFIRST Microlensing Survey. I. Bound Planet Detection Rates". In: *The Astrophysical Journal Supplement Series* 241.1 (2019), p. 3. DOI: 10.3847/1538-4365/aafb69. arXiv: 1808.02490 [astro-ph.EP].

[29] Michael Perryman. *The Exoplanet Handbook*. 2nd ed. Cambridge University Press, 2018.

[30] R. Poleski and J. C. Yee. "Modeling microlensing events with MulensModel". In: *Astronomy and Computing* 26, 35 (Jan. 2019), p. 35. DOI: 10.1016/j.ascom.2018.11.001. arXiv: 1803.01003 [astro-ph.IM].

[31] Jun Qin and Pengfei Sun. "Applications and comparison of continuous wavelet transforms on analysis of a-wave impulse noise". In: *Archives of Acoustics* 40.4 (2015), pp. 503–512.

[32] Nicholas James Rattenbury. "Planetary microlensing: From prediction to discovery". In: *Modern Physics Letters A* 21.12 (2006), pp. 919–933. DOI: 10.1142/S0217732306020470. arXiv: astro-ph/0604062 [astro-ph].

[33] Thomas Schreiber and Andreas Schmitz. "Discrimination power of measures for nonlinearity in a time series". In: *Physical Review E* 55.5 (1997), p. 5443. DOI: 10.1103/PhysRevE.55.5443. arXiv: chao-dyn/9909043.

[34] Dirk Schulze-Makuch and William Bains. "Time to consider search strategies for complex life on exoplanets". In: *Nature Astronomy* 2.6 (2018), pp. 432–433. DOI: 10.1038/s41550-018-0476-2.

[35] A Udalski, MK Szymański, and G Szymański. "OGLE-IV: fourth phase of the Optical Gravitational Lensing Experiment". In: *arXiv preprint* 65.1 (Mar. 2015), pp. 1–38. arXiv: 1504.05966 [astro-ph.SR].

[36] Guido Van Rossum, Barry Warsaw, and Nick Coghlan. "PEP 8: style guide for Python code". In: *python.org* 1565 (2001).

[37] *WFIRST*. URL: https://wfirst.gsfc.nasa.gov/about.html.

[38] Greg Wilson, Jennifer Bryan, Karen Cranston, Justin Kitzes, Lex Nederbragt, and Tracy K. Teal. "Good enough practices in scientific computing". In: *PLOS Computational Biology* 13.6 (June 2017), pp. 1–20. DOI: 10.1371/journal.pcbi.1005510. URL: https://doi.org/10.1371/journal.pcbi.1005510.

[39] *YAPF - A formatter for Python files*. URL: https://github.com/google/yapf.

A `galens` Package Documentation

galens Documentation

Contents

Module galens

Sub-modules

- galens.modelling
- galens.preparation
- galens.processing
- galens.utils

Namespace galens.modelling

Sub-modules

- galens.modelling.bayesreg

Module `galens.modelling.bayesreg`

Functions

Function `rms`

```
def rms(w, x, t)
```

Returns root mean square error of a polynomial model.

Args

`w: iterable` Polynomial coefficients.
`x: iterable` Samples.
`t: iterable` Target values with same shape as x.

Returns

`float` Root mean square error.

Classes

Class `BayesianLinearRegression`

```
class BayesianLinearRegression(alpha=0.001, fit_intercept=True)
```

A Bayesian linear regression with Gaussian prior.

Differs from BayesianPolynomialRegression in that the feature matrix must be passed explicitly.

From Kempa-Liehr (2016): Don't Overfit - The Engineers' (totally incomplete) Guide to Computational Analytics.

Parameters

`alpha: float, optional` Regularization parameter.
`fit_intercept: bool, optional` Whether to calculate with an intercept.

galens Documentation

Ancestors (in MRO)

- sklearn.base.BaseEstimator
- sklearn.base.RegressorMixin

Methods

Method `fit`

```
def fit(self, X, y)
```

Fitting function for Bayesian linear regression with Gaussian prior.

Args

X (array-like or sparse matrix): The training input feature matrix. Of shape [n_samples, n_feats]). y (array-like): The target values. Targets are class labels in classification, real numbers in regression. Of shape [n_samples].

Returns

`object` Returns self.

Method `predict`

```
def predict(self, X)
```

Predict function for estimator.

Args

X (array-like of shape = [n_samples, n_features]): The input samples.

Returns

y (`array` of shape = [`n_samples`]) Returns :math:`x^2` where :math:`x` is the first column of `X`.

Method `predict_cdf`

```
def predict_cdf(self, X)
```

Estimates conditional density functions.

Args

X (array-like): The input samples. Of shape [n_samples, n_features].

Returns

`y: array` Returns [x^2] where [x] is the first column of [X]. Of shape = [n_samples].

galens Documentation

Class `BayesianPolynomialRegression`

```
class BayesianPolynomialRegression(M=10, alpha=0.001)
```

A Bayesian linear regression with Gaussian prior.

From Kempa-Liehr (2016): Don't Overfit - The Engineers' (totally incomplete) Guide to Computational Analytics.

Parameters

`M : int, optional` Complexity with respect to order of polynomials.
`alpha : float, optional` Regularization parameter.

Ancestors (in MRO)

- sklearn.base.BaseEstimator
- sklearn.base.RegressorMixin

Methods

Method `fit`

```
def fit(self, X, y)
```

Fitting function for Bayesian linear regression with Gaussian prior.

Args

X (array-like or sparse matrix): The training input samples. Of shape [n_samples, 1]. y (array-like): The target values. Class labels in classification real numbers in regression. Of shape [n_samples].

Returns

`object` Returns self.

Method `predict`

```
def predict(self, X)
```

Predict function for estimator.

Args

X (array-like of shape = [n_samples, n_features]): The input samples.

Returns

y (array of shape = [n_samples]) Returns :math:x^2 where :math:x is the first column of X.

Method predict_cdf

```
def predict_cdf(self, X)
```

Estimates conditional density functions.

Args

X (array-like of shape = [n_samples, n_features]): The input samples.

Returns

y (array of shape = [n_samples]): Returns :math:x^2 where :math:x is the first column of X.

Module galens.modelling.features

Classes

Class FeatureTransformer

```
class FeatureTransformer(columns, column_id='lc_index', column_value='
magnitude', column_sort='t')
```

Transforms light curves into set features only.

Args

columns : list tsfresh style feature names to extract.
column_id : str The name of the id column to group by.
column_value : str The name of the data itself column.
column_sort : str The name of the sort column.

Attributes

feature_dict_ : dict Feature dictionary for tsfresh.

Ancestors (in MRO)

- sklearn.base.TransformerMixin

Methods

Method `fit`

```
def fit(self, X=None, y=None)
```

Creates feature dictionary from columns for tsfresh.

Args

`X` Ignored.
`y` Ignored.

Method `transform`

```
def transform(self, X)
```

Extracts features from X with tsfresh, returning features.

Args

`X: pandas.DataFrame` Data frame for feature extraction.

Returns

pandas.DataFrame: Features data frame.

Module `galens.modelling.modelio`

Functions

Function `load_model`

```
def load_model(file_name, dataset_name='model')
```

Unpickles a model from an HDF5 file.

Args

`file_name: str` HDF5 file to save to.
`dataset_name: str` Attribute name in HDF5 file.

Returns

`object` Unpickled object.

Function `save_model`

```
def save_model(model, file_name, dataset_name='model')
```

Pickles and saves model to HDF5 file.

Args

`model: pickleable` Object to pickle and save.
`file_name: str` HDF5 file to save to.
`dataset_name: str` Name of dataset in HDF5 file.

Namespace `galens.preparation`

Sub-modules

- galens.preparation.errors
- galens.preparation.gaia
- galens.preparation.generate_curves
- galens.preparation.kmtnet
- galens.preparation.magmap
- galens.preparation.metagrid
- galens.preparation.moa
- galens.preparation.ogle

Module `galens.preparation.errors`

Functions

Function `ogle_observations`

```
def ogle_observations(ogle_path)
```

Concatenates all OGLE magnitudes with their errors.

Args

`ogle_path: str` Path to raw OGLE download.

Returns

np.ndarray: 2D array of magnitudes with errors.

galens Documentation

Classes

Class `ErrorState`

```
class ErrorState(*args, **kwargs)
```

Exposes methods to add error to microlensing light curves.

Ancestors (in MRO)

- mtrand.RandomState

Methods

Method `observe`

```
def observe(self, magnitude, model='ogle_v1')
```

Adds error to magnitude, drawing from a random distribution.

Args

`magnitude : array` Magnitudes to add error to. This should be an array, else some models error.
`model : str, optional` Error model to use. Defaults to `ogle_v1`.

Returns

number or array: Magnitudes with random error added.

Method `sigma_ogle_v1`

```
def sigma_ogle_v1(self, magnitude)
```

Determines the error as a function of magnitude.

This implements the error function described in experiments/2019_05_04_01.

Args

`magnitude : number or array` I magnitude to get the error of.

Returns

number or array: Error, has the same shape as magnitude.

Method `sigma_ogle_v2`

```
def sigma_ogle_v2(self, magnitude)
```

Determines error as a function of magnitude.

Implements function in experiments/2019_05_23_01.

Args

`magnitude : number or array` I magnitude to get error of.
`Returns` number or array: Error, has the same shape as magnitude.

Module `galens.preparation.gaia`

Functions

Function `download_extract`

```
def download_extract(links, location, force=False)
```

Atomically downloads URLs, extracting them if they are .gz files.

Downloads files at the URLs in the `links` list. Files with the `.gz` extension are extracted. The zipped file is removed.

Args

`links : list` List of URLs to download.
`location : str` Path to local download location.
`force : bool, optional` If False, existing files will be skipped. Defaults to False.

Function `get_files_to_download`

```
def get_files_to_download(remote)
```

Gets a list of files to download from the remote index page.

This is specifically designed for those simplest of index pages consisting of a set of links to the files of interest.

Args

`remote : str` path to remote directory

Returns

list list of complete URLs to download.

Module galens.preparation.generate_curves

Functions

Function calc_magnification

```
def calc_magnification(model_params, magnification_methods, time_steps)
```

Gets a magnification curve from Mulens.

Args

model_params: dict See MulensModel documentation.
magnification_methods: list See MulensModel documentation.
time_steps: list Points in time to calculate magnification at.

Returns

array or None: Magnifications, same shape as time_steps if no errors. None if MulensModel threw an error.

Function calculate_magnitude

```
def calculate_magnitude(magnification, base_magnitude=15)
```

Calculates magnitude from magnification. (Info in log 2019-05-05.)

Args

magnitude: pandas.Series Magnification values.
base_magnitude: number, optional Defaults to 15. TODO: change this to vary curve to curve.

Returns

pandas.Series: I magnitude values.

Function compute_time_derivatives

```
def compute_time_derivatives(df, column='magnitude', n=2)
```

Computes and appends time derivatives of column to a given dataframe.

Args

df: pandas.DataFrame Data frame to operate on.
column: str, optional Column to differentiate. Defaults to “magnitude”.
n: int, optional Number of derivatives. Defaults to 2.

Returns

df Modified dataframe.
list Column names of original column and derivatives

Function get_parameter_grid

```
def get_parameter_grid(params)
```

Creates a an iterable parameter grid.

This differs from the sklearn function because it can handle scalar values in the dictionary.

Args

params: dict Dictionary of parameters.

Return

sklearn.model_selection.ParameterGrid: Grid. **int**: Grid length.

Function get_target_df

```
def get_target_df(df)
```

Gets target dataframe for light curves df.

Args

df: pandas.DataFrame Light curves.

Returns

pandas.DataFrame: Light curve targets (model_params).

galens Documentation

Classes

Class `LightCurveExperiment`

```
class LightCurveExperiment(model_params, file_name, location=None,
time_steps=None, magnification_methods=None, error_seed=None, errors=
None, base_magnitude=None, **kwargs)
```

Generates light curves based on an experiment configuration.

When this class is initialised with an appropriate configuration, light curves are produced and saved according to how it is configured.

A light curve is generated for each combination of model paramaters, corresponding to one microlensing event.

Currently this class is capable of handling binary lens systems only.

Attributes

`model_params: dict` Model parameters for Mulens. Each model parameter should be a singular value (of the form expected by Mulens) or a list of values. If a list of values is given, each value is considered in turn, such that all combinations with other parameters are considered.

`file_name: str` Name of the experiment file (generally ending in .hdf5) This becomes the name of the folder where the experiment is saved.

`location: str` The root experiments folder.

`time_steps` A list of times at which to determine magnification.

`magnification_methods: list` See documentation for MulensModel.

`output_magnification: bool` Indicates if a magnification plot should be produced for each parameterisation. See Mulens documentation for details.

`output_trajectory_plot: bool` Indicates if a trajectory plot should be produced for each parameterisation. See Mulens documentation for details.

`t_range: list` The start and end points of `self.time_steps`.

`folder: str` The complete path to the experiment folder.

Initialises and creates experiment data.

Args

`model_params: dict` Model parameters for Mulens.

`file_name: str` Defaults to None. Name of the experiment file.

microlensing@b5891cc

galens Documentation

location : str, optional Defaults to None. Root folder for the experiment. “data/experiments/” is used if None is provided.

time_steps : list, optional Defaults to None. A list of times at which to determine magnification. If None is provided, this defaults to `np.linspace(t_0 - 50, t_0 + 50, 100)`.

magnification_methods : list, optional See Mulens documentation for details. Defaults to [t_0 - 50, “VBBL”, t_0 + 50] if None is given.

error_seed : int, optional Seed for errors.

errors : str, optional Error implementation. Currently “ogle_v1” and “ogle_v2” are supported.

base_magnitude : list or number Base magnitude for light curves. If this parameter is a list, a set of light curves for each value will be generated and become columns. If a number is given, one set will be generated at that base magnitude. Default is [15].

Properties

parameter_grid : sklearn.model_selection.ParameterGrid A generator yielding each combination of Mulens parameters.

grid_length : int The number of parameter combinations.

index_columns : list The column names for the index file.

Methods

Method create_magnification_df

```
def create_magnification_df(self)
```

Creates light curves with Mulens.

Method output_light_curves

```
def output_light_curves(self)
```

Gets light curves and saves them to the experiment HDF5 file.

Method plot_magnification

```
def plot_magnification(self, model, file_name)
```

Plots and saves a magnification plot of a given model.

Args

model : MulensModel.Model Mulens model to plot.

file_name : str The light curve file name to match.

Method `plot_trajectory`

```
def plot_trajectory(self, model, file_name)
```

Plots and saves a trajectory plot of a given model

Args

`model : MulensModel.Model` Mulens model to plot.
`file_name : str` The light curve file name to match.

Method `save`

```
def save(self, data, key, path=None)
```

Save a data to HDF5.

Args

`data` Data to save.
`key : str` Dataset name.
`path : str, optional` HDF5 file path. Defaults to self.data_file.

Module `galens.preparation.kmtnet`

Functions

Function `download_kmtnet`

```
def download_kmtnet(config, force=False)
```

Downloads each year of KMTNet observations in the configuration

Args

`config : dict` downloads configuration
`force : bool` if False, existing event files are not downloaded

Function `download_kmtnet_by_year`

```
def download_kmtnet_by_year(config, year, force=False)
```

Downloads one year of KMTNet events

Args

`config: dict` downloads config
`year: str` year of events
`force: bool` if False, existing event files are not downloaded

Function `get_index`

```
def get_index(index_url, year)
```

Parses the HTML index of a year of KMTNet events to a pandas data frame

Args

`index_url: str` URL of event page
`year: str` year of the event page

Returns

`pandas.DataFrame: index page, including links to DIA files`

Function `kmt_event_to_link`

```
def kmt_event_to_link(event, index_url)
```

Determines the link to an event's DIA files from its ID

Args

`event: str` event ID
`index_url: str` base URL

Returns

`str` url to event DIA file

Module galens.preparation.magmap

Functions

Function magnification_grid

```
def magnification_grid(t_E, rho, q, s, size=None, t_0=0.0, res=600)
```

Creates a magnification map.

Args

t_E: float Einstein time scale in days.
rho: float Source size as a fraction of the Einstein radius.
q: float Mass ratio between the two lens bodies.
s: float Separation of the lens bodies relative to Einstein ring size.
size: float or None Dimension of the map, relative to Einstein ring size.
res: int Grid will be shape (res, res).

Returns

numpy.ndarray: Magnification map.

Module galens.preparation.metagrid

Functions

Function base_mag_cut_grid

```
def base_mag_cut_grid(**kwargs)
```

Generates light curves on a base magnitude and cut prob grid.

The model_params can contain (function, args) tuples which are as many times as needed to realise randomness at each iteration.

Args

kwargs: dict See gencurves.LightCurveExperiment.

Returns

pandas.DataFrame: Light curves.

Function `evaluate_params`

```
def evaluate_params(params)
```

Calls functions defined in params.

Purpose is to allow for defining model parameter definitions by random function arguments.

Args

`params: dict` Dict of parameters to evaluate.

Returns

`dict` Evaluated parameters, with the same keys as params.

Module `galens.preparation.moa`

Functions

Function `download_cross_ref_ogle`

```
def download_cross_ref_ogle(config, force=False)
```

Download the early file for OGLE/MOA cross references

Args

`config: dict` download config dictionary
`force: bool` overwrite existing file

Function `download_moa`

```
def download_moa(config, force=False)
```

Download MOA events for the years specified in the config.

Args

`config: dict` download config dictionary
`force: bool` overwrite existing files

Function `download_moa_early_years`

```
def download_moa_early_years(config, year, force=False)
```

Download MOA events for the years 2006 to 2015.

Args

`config: dict` download configuration dictionary
`year: str` year to download
`force: bool` overwrite existing files

Function `download_moa_file`

```
def download_moa_file(url, payload_string, save_location, force=False)
```

Downloads a MOA file with all its little nuances.

Args

`url: str` base url
`payload_string: str` query string
`save_location: str` local save location
`force: bool` overwriting existing files

Function `download_moa_recent_years`

```
def download_moa_recent_years(config, year, force=False)
```

Download MOA files for 2016 to present.

Args

`config: dict` download configuration dictionary
`year: str` year to download
`force: bool` overwrite existing files

Module `galens.preparation.ogle`

Functions

Function `download_ogle`

galens Documentation

```
def download_ogle(config, force=False)
```

Downloads OGLE4 data according to config

Args

config: dict downloads config dictionary

force: bool overwrite already downloaded files If False, and the event folder already exists, nothing will be downloaded (even if folder is empty).

Function download_ogle_by_year

```
def download_ogle_by_year(config, ftp, save_location, force=False)
```

Downloads one folder of OGLE4 data.

Args

config: dict downloads config dictionary

ftp (ftplib.FTP): This should be an FTP object, with working directory set to the folder to be downloaded (i.e. a year, e.g. 2019). **save_location** : `str` : folder to save events to

force: bool overwrite already downloaded files If False, and the event folder already exists, nothing will be downloaded.

Namespace galens.processing

Sub-modules

- galens.processing.feature_extraction
- galens.processing.gaps

Module galens.processing.feature_extraction

Classes

Class FeatureExtractor

galens Documentation

```
class FeatureExtractor(file_name, model_params, location=None, extraction_settings
=None, column='magnitude', lc_key='light_curves', **kwargs)
```

Extracts features from an experiment file.

Initialises the object, and loads features if they already exist.

Args

`file_name : str` Experiment HDF5 file.
`model_params : dict` Model parameters used for simulation.
`location : str, optional` Location of file. Defaults to None.
`extraction_settings : optional` tsfresh settings. Defaults to None.
`column : str, optional` Name of column to extract features from. Defaults to None. If None, column names containing "magnitude" will be used.

Methods

Method `extract_features`

```
def extract_features(self)
```

Extracts and saves features of the light curves.

Returns

`pandas.DataFrame: table of relevant features`

Method `select_features`

```
def select_features(self)
```

Method `test_significance`

```
def test_significance(self)
```

Finds features of light curves that are significant for determining model parameters.

TODO: Fix the fact that this won't work for multiple basse magnitudes.

Returns

`pandas.DataFrame: table of relevant features`

galens Documentation

Module galens.processing.gaps

Functions

Function cut

```
def cut(df, p, gap_width, random_state=None)
```

Removes chunks of length `gap_width` from `df` with probability `p`.

Args

`df: pandas.DataFrame` Light curve data frame.
`p: float` Bernoulli probability for each chunk in range [0, 1]. 0 means no chunks will be removed, 1 means all.
`gap_width: int` Length of any cut chunks.
`random_state: numpy.random.RandomState` Seeded random number generator, for reproducibility.

Function cut_all

```
def cut_all(probabilities, lc, gap_width)
```

Makes cuts in the light curves at probabilities.

This function assumes the `lc_index` is uncorrelated with the model parameters. The light curve will be split into len(probabilities) sections, ordered by `lc_index`, with any remaining curves dropped. Each subsection will have cuts removed at the corresponding probabilities.

Args

`probabilities: list` List of probabilities for Bernoulli trials. Where each value in the range [0, 1].
`lc: pandas.DataFrame` Data frame of light curves.
`gap_width: int` Width of each cut out.
`Returns` pandas.DataFrame: Data frame of all light curves, with cuts.

Function cut_days

```
def cut_days(df, p, gap_width_days=1, random_state=None)
```

Classes

Class `MakeGaps`

```
class MakeGaps(probabilities, gap_width=None, gap_seed=None, **kwargs)
```

Allows the removal of chunks from light curve data.

The goal here is to roughly model real collection error issues in telescope data.

Initialises object to remove chunks from light curves.

Methods

Method `cut_all_and_save`

```
def cut_all_and_save(self, key='cut_curves', probabilities=None, lc=
None, gap_width=None)
```

Makes cuts and saves to the experiment HDF5 file.

Args

key (str, default “cut_curves”): HDF5 table key. **`probabilities`** : `list` : Set of cut probabilities to make.

`lc: pandas.DataFrame` Custom light curves data frame.
`gap_width: int` Length of any gap to cut out.

Module `galens.utils`

Functions

Function `download_file`

```
def download_file(url, save_location, errors='ignore', force=True)
```

Downloads a file to disk

Args

`url: str` URL of file
`save_location: str` local save location

errors : str treatment of non 200 status codes If ignore, a failed download returns silently Otherwise, an requests.exceptions.HTTPError is raised.

Raises

requests.exceptions.HTTPError: Raised if a non 200 status code is reached, and errors is not “ignore”.

Function `get_current_commit`

```
def get_current_commit()
```

Gets current git hash of the repository.

Function `get_download_config`

```
def get_download_config(path='conf/download.yaml')
```

Load YAML config for downloads

Function `put_git_hash_in_hdf5`

```
def put_git_hash_in_hdf5(path_to_hdf5_file)
```

Adds the current git hash as an attribute to an HDF5 file.

Args

path_to_hdf5_file : str Path to file.

Function `sanitise_string`

```
def sanitise_string(string)
```

Sanitise a string as a column title.

Generated by *pdoc* 0.6.3 (https://pdoc3.github.io).