BatSignal

A Comprehensive Guide

Giovannina Mansir, Patricio Rojo

November 18, 2019

Contents

1 Introduction

As the field of extrasolar planets is still relatively new in astronomy, there are not many packages available to analyze raw transit data. Many programs exist on many different platforms to create a model light curve, but very few will fit these models to data. The purpose of BatSignal is to have a flexible, open-source python package to fully analyze all types of transit data.

Presently, BatSignal is capable of handling multiple reduced light curves (divided by a reference star) of a single star. It is able to fit these either simultaneously or independently, at the user's discretion.

1.1 How it Works

BatSignal is able to fit models to data through the use of Gaussian Process Regression and MCMC analysis. Upon the creation of an instance, the program reads in the data, as well as an input file detailed below. This file will contain information about the planet and determine which parameters the user wishes to fit the data for.

With this information in hand, running the main program fits a model to the input light curve data. It first initializes a model with the Bad-Ass Transit Model cAlculatioN (BATMAN) using the user's guesses of the parameter values. It then sets up the sampler for emcee, the package used to run the MCMC analysis.

For each chain, the main program first calls `lnprob`. This finds the total probability of a given model being the true transit light curve. Its first action is to pass the test parameters to `lnprior`. `lnprior` tests each of the parameters being fit for to ensure that the test value is within 3σ of the original value. If all test parameters are within 3σ, then it computes the following value

$$value = -\sum_{\text{test values}} \frac{(p_{\text{test}} - p_{\text{true}})^2}{(2\sigma)^2}. \tag{1}$$

This weights each guess by its distance from the original input. It returns this back to `lnprob`, which then calls `lnlike`. The BATMAN model is first updated with the test parameters. Then, it creates the kernel necessary for the Gaussian process regression. It runs the gaussian process on the data, and then determines the likelihood of the model being the true fit. It returns this value to `lnprob`, which then adds it to the value returned from `lnprior`. `lnprob` then give this back to the main program. The MCMC

chain then determines if this model was more likely than the last step in the chain so that by the end, the resulting model is the most likely.

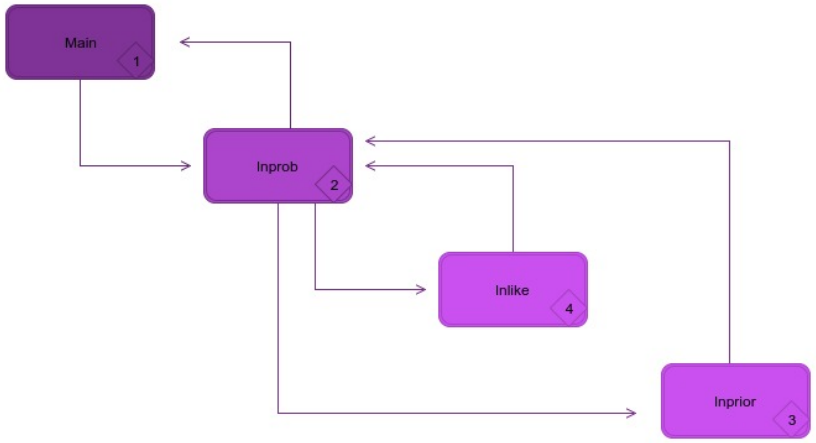


Figure 1: A schematic view of the program.

Note the MCMC maximization is performed with a burn-in phase that is about one-fifth the size of the full analysis. the program will print the time it takes to perform the burn-in (in seconds) and the full analysis upon completion. Finally, it will return the model and parameters in an object described in detail below.

2 Input

BatSignal utilizes the configparser module to create and read an input file. The first section, named "What To Fit" toggles which parameters the program can fit for. Multiple parameters may be fit at the same time, however if a law with multiple limb darkening coefficients are used, all coefficients must be fit at the same time.

The parameters BatSignal can fit for are the same as BATMAN, which BatSignal calls:

- Ratio of the Radii (in units of stellar radii)
- Limb Darkening Coefficients (defaults to quadratic limb darkening)
- Time of Inferior Conjunction
- Period

- Scaled Semi-major Axis (in units of stellar radii)
- Inclination (in degrees)
- Eccentricity
- Argument of Periastron Longitude (in degrees)

Following this, sections may be added titled with the name of the planet. Each section should contain known values (or guesses) of the 8 parameters, and their error. The program uses the error stated in the gaussian processes, so it is imperative that the user enters reasonable values.

When creating the BatSignal instance, the planet name may be added after the input file and light curve. If no planet is named, the program defaults to the first section below "What To Fit".

If running multiple transits, note that the lightcurve file may be a list of filenames. Within the configuration file, there is a section called "Multi-transit". In this section, the user may decide which transits to use for a given fit. A value of "0" indicates that the transit should not be used for the fit of that parameter. A value of "1" is used to fit the parameter for that transit independently from the other transits, and a value of "2"or more is used to use the transit simultaneously with the other transits to find a single value for that parameter.

3 Quickstart Guide

In order to run, BatSignal requires the reduced data from a planetary transit, and a few guesses for the parameters of the star system. The input file is described in detail in section 2. The data file should contain at least two columns: Time and Normalized Flux. An optional third column may contain error on the flux measurements.

The user may create an instance using:

```
instance = BatSignal('Input.cfg', 'LightCurve.txt', 'Planet Name')
```

The optional `'Planet Name'` keyword is only necessary if there are multiple planets in the input file.

To fit a model to the light curve data, run :

```
output = instance.bat()
```

By default, this will assume the use of a quadratic limb darkening law. Batman, however, has many limb darkening options. Other laws may be used by adding `'nonlinear'`, `'linear'`, `'uniform'`, `'squareroot'`, `'logrithmic'`, or `'exponential'` to the calling sequence.

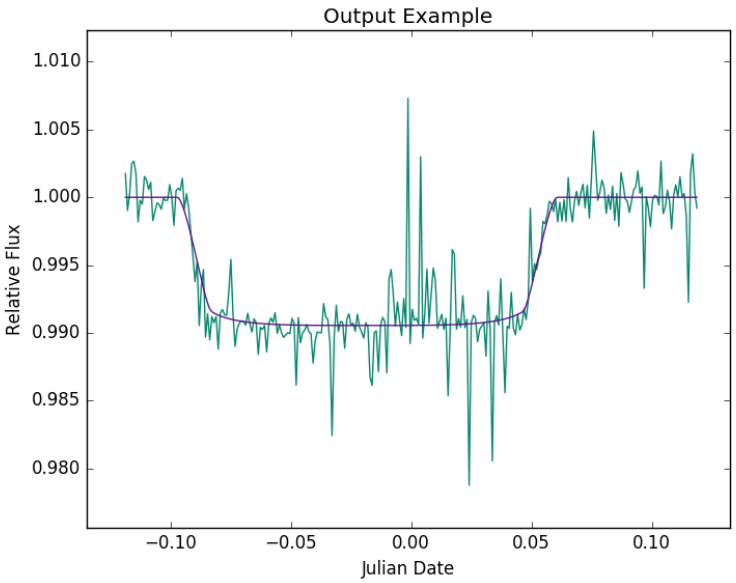
Additionally, this does not plot anything by default. Adding `'model'` will plot the data overlaid with the fit model. Adding `'corner'` will display the corner plot from the MCMC sampler. And finally, `'chain'` will display the walkers used in the MCMC computation

The `return` object will contain the following attributes:

- `self.input_light_curve`: The name of the file containing the input light curve
- `self.date, self.flux. self.error`: The input light curve data
- `self.input_param_file`: The name of the file file containing the parameter information
- `self.names`: The names of the parameters the user wished to adjust
- `self.variables`: The original guess for the adjusted parameters
- `self.model`: The fit to the light curve found by BatSignal
- `self.results`: The adjusted values of the variables, with lower and upper error bars
- `self.bat`: The main light curve fitting program
- `self.create_param_file`: A program to generate an empty input file
- `self.create_test_data`: A program to create synthetic data to ensure the module runs properly on the user's machine

4 Error Analysis

In order to say with conviction that this package works, a complete error analysis was conducted.



(a)

Figure 2: An example of data overlaid with the fit model

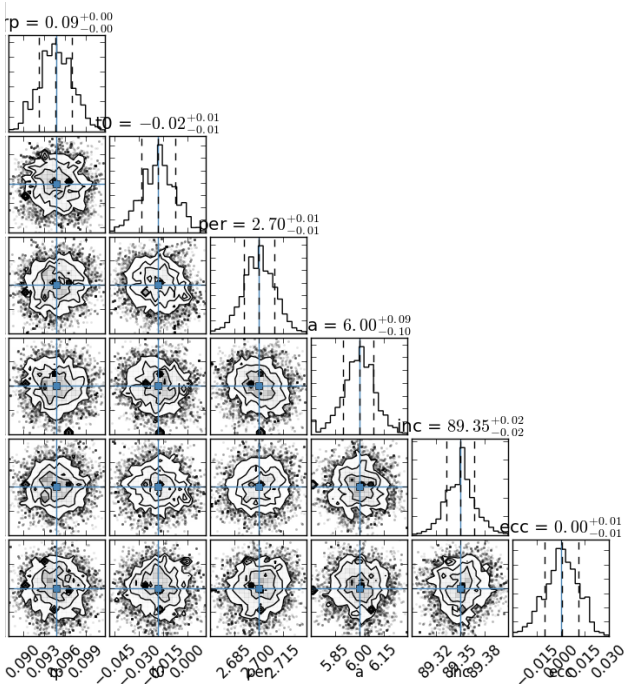


Figure 3: An example of the corner plot

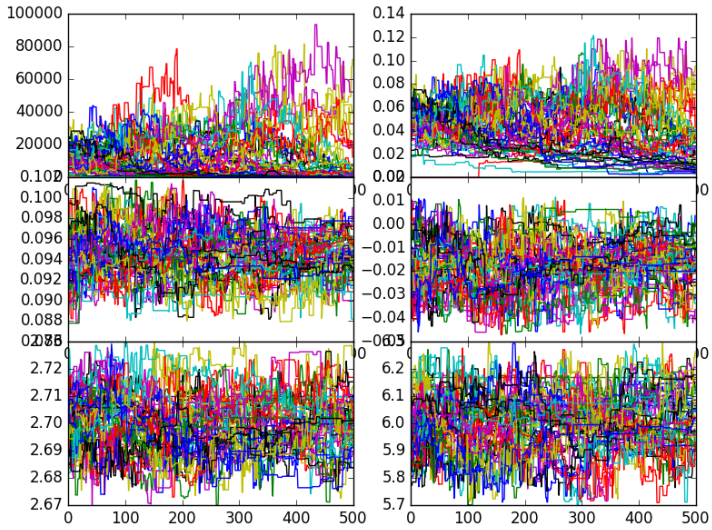


Figure 4: An example of the walker graph

4.1 Synthetic Data

To create the most realistic data possible, we started with a simple model transit synthesized using batman. The parameters for the model were arbitrarily based off of the real planet WASP-61. For the white noise, we pulled random numbers from a Cauchy distribution. The light curve of the target star is nearly always divided by the curve from a nearby reference star. If the white noise is usually considered Gaussian for each of these, then dividing the two would result in Cauchy distribution. The width of the distribution was varied to simulate different observing conditions.

The red noise was created by taking the real part of the square of the inverse Fourier transform of the white noise. This then would also vary with the change in width of the Cauchy distribution. Together, we added these to the batman model and used this simulated data to run our error analysis.

Knowing the true parameters of our fictitious planet, we were able to analyze the accuracy of BatSignal.

4.2 Analysis

The fist point to note is that the error on the fit parameter is strongly dependent on the user's stated error. In Figure 6,the ratio of the radii was fit for a single synthetic light curve. The dashed line indicates the true value. 10 trials were run, with the only difference being the error the user input for

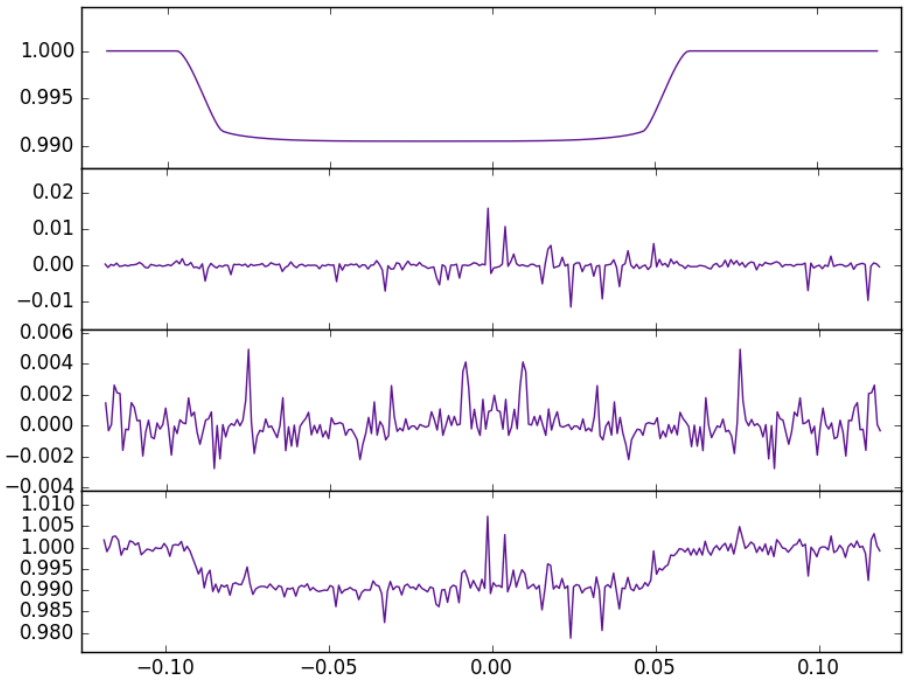


Figure 5: The first panel depicts a transit modeled using batman of a fictional planet. Next is the Cauchy-based white noise, followed by its inverse Fourier transform as the red noise. These three panels were added together to create the synthetic data shown in the last panel.

their guess. The diverging solid lines indicate the user's error value. While the value of the fit ratio remained fairly consistent, the error on each point is clearly strongly correlated with the user's error.

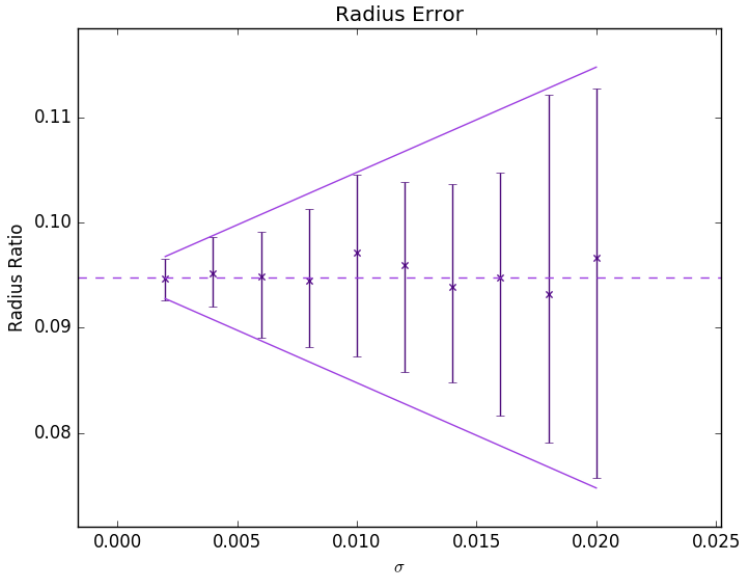


Figure 6: Fit of the ratio of the radii with increasing user error.

That said, the fit did preform admirably with an increasing noise level in the data. The following trials were completed for two synthetic planets (purple and teal). The dashed lines indicate the true values, while the shaded regions show the user's error. For each point, the value of σ was increased before the white and red noise was added to the model. Presenting the fit of the semi-major axis first,

As the purple planet in Figure 7 exemplifies extremely well, when the noise in the data increases, the error bars of the fit parameter also increase.

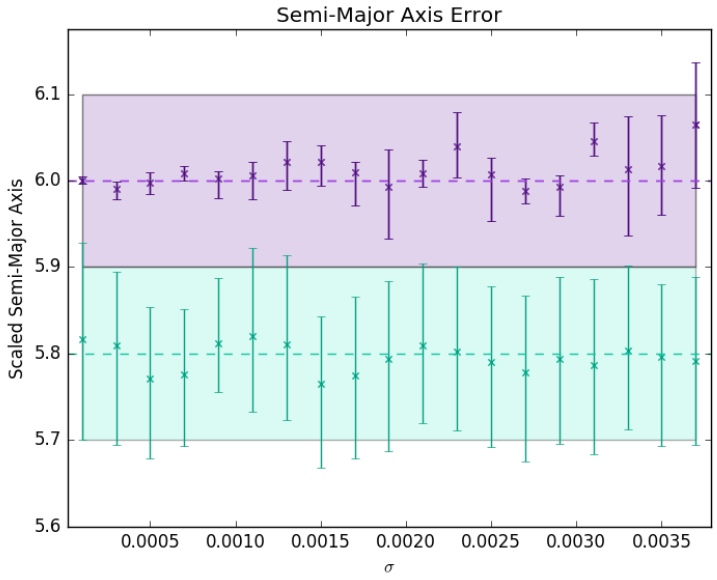


Figure 7: Fit of the semi-major axis with increasing noise in the data.