Solving for the Orbital Elements of Binary Systems Using Markov Chain Monte Carlo Simulations

Author:
Kyle Mede

Collaborator:
Yasuhiro Takahashi

Supervisor:
Dr. Masa Hayashi

University of Tokyo Research Internship Program (UTRIP)
Participant
The University of Tokyo
Tokyo, Japan

May 2010 - April 2011

Contents

List of Figures

1. Introduction

There are two major types of binary systems observed in space, star-planet and star-star. Knowing the parameters that describe the orbits of these systems is critical when trying to understand the variety of systems that exist and to better study the bodies involved.

In the case where the secondary body is significantly smaller than the primary, more than 100 times smaller, the system is most likely composed of a star and planet. For these systems, the large difference in size causes the center of mass for the system to be within the primary star, and thus only a single set of orbital parameters for the secondary needs to be determined. On the other hand, if the two bodies are of similar mass then the system is composed of two stars orbiting a common center of mass that will lie on a line somewhere between them at all times, as can be seen in figure 1. In order to fully describe these systems, there are two sets of orbital parameters that need to be found.

When observing either binary systems, there are only a few measurable values that can be attained. By observing the light from the individual objects and using techniques in astrometry and photometry, estimates on their masses can be made. After locating the center of each object, the angular separation between them and the angle between the line connecting the them and vertical from the center of the primary can be measured; these are the Separation and Position Angles of the system at the time of the observation, shown in figure 2. Due to the many parameters needed to fully describe the orbits and only these few measurements that can be made, the method of Monte Carlo Simulations is utilized to best determine the missing ones. In this method, the system is simulated using our understanding of the laws of physics that the orbits will obey to create a model; the model is then ran many times with random parameters and the resulting values are compared to those observed to find the complete parameter set which most likely describe the system.

The ability to use Monte Carlo, but also hone in on the solution while the simulation is running, is more accurate and efficient and is the idea behind Markov Chain Monte Carlo (MCMC) simulations. In MCMC, an algorithm is applied each time a random orbit is created in order to determine how much better it fits the observed values than the previous one and to try again if it is significantly worse. Overall, this approach requires random numbers to be generated, an orbit calculator to create outputs from these, an algorithm to compare each orbit to the previous, and to repeat the process many times until the best solution set is found. To avoid one large program that is hard to follow and debug, this simulator has been broken into two parts: The Orbit Calculator, which applies laws of physics to convert the random inputs to predicted observables for the system, and the MCMC Simulator which utilizes this calculator and performs the rest of the required tasks. These two parts of the code are covered in detail in sections 2 and 3. All of the code has been written in the Python programming language because of its clarity when reading it and the large supply of publicly available packages and libraries that can be utilized to speed-up the coding process.

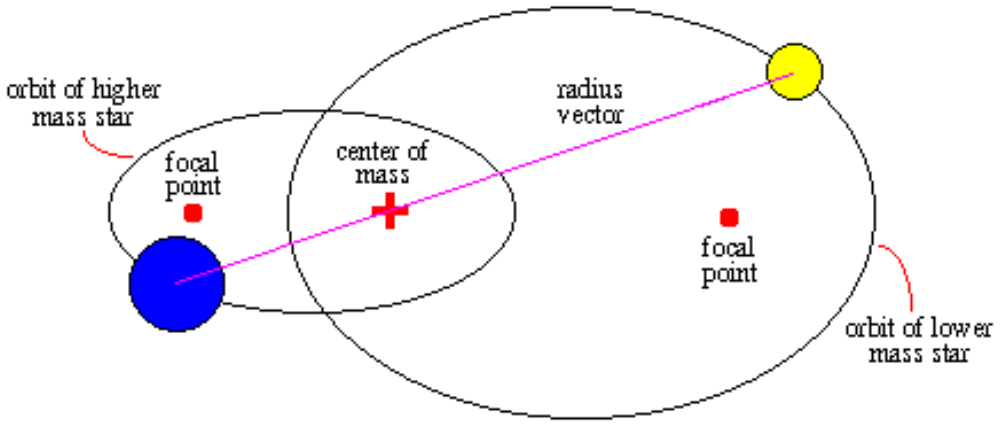

Fig. 1.— Orbits of a Binary Star System

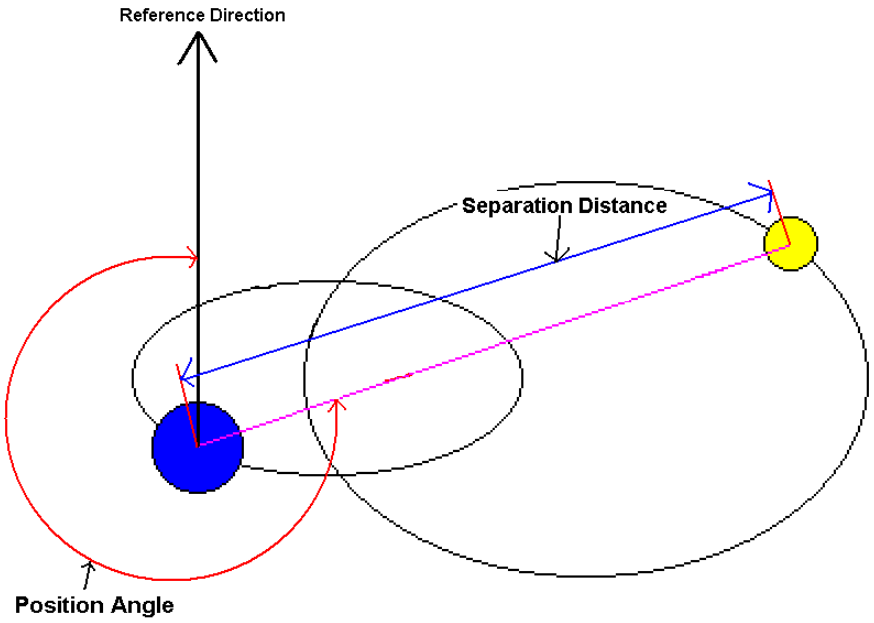

Fig. 2.— Diagram of Binary Star System and its Separation and Position Angles

2. The Orbit Calculator

2.1. Description

This Python function (orbitCalculator) calculates the orbital elements for either a binary star system or a singe planet's orbit. The equations used are the result of combining Kepler's laws, the definitions of the orbital elements and the naturally occurring symmetries and rules of a stable binary system.

The function is held in a toolbox, or library, titled orbitToolbox2.py. This is a collection of functions that are useful for creating an orbit simulator, such as a basic Monte Carlo or a more complex MCMC simulator. For simulators of these types, the inputs to orbitCalculator would be generated using a random number generator, such as random.uniform() in Python. In this manner, the function would only need to be called once per sample input set in a loop to provide the resulting outputs from those random numbers. These outputs would then be compared to the observed values through a chi-square statistic to provide an estimate of how closely they match.

The current function call follows:
orbitCalculator(t, Sys_Dist_PC, Sep_Angle_Measured_arcsec, inclination_deg, longAN_deg, e, T, period, argPeri_deg, Mass1=1, Mass2=1, verbose=False)

and returns:
(n, M_deg, E_lates_deg, TA_deg, Sep_Dist_AU_OP, PA_deg_measured, a1, a2)

2.2. Inputs

Parameter	Description	Suggested Range
t*	Epoch of observation/image [julian date]	[2400000.0, 2460000.0]
Sys_Dist_PC*	Measured system distance from Earth [PC]	[0.01, 50.0]
Sep_Angle_Measured_arcsec	Measured Separation Angle of stars ["]	[0.01, 2.0]
inclination_deg	Inclination [^{o}]	[0.001, 179.999]
longAN_deg	Longitude of Ascending Node [^{o}]	[0.001, 179.999]
e	Eccentricity of orbits [unitless]	[0.001, 0.999]
T	Time of Last Periapsis [julian date]	[t-period, t]
period	Period of orbits [yrs]	[1.0, 100.0]
argPeri_deg	Argument of Periapsis in orbital plane [^{o}]	[0.001, 89.999]
Mass1	Mass of primary [Msun](=1 $\rightarrow$ M2 is a planet)	[0.001, 10]
Mass2	Mass of companion [Msun](=1 $\rightarrow$ M2 is a planet)	[0.001, Mass1]
verbose	Send prints to screen? (Default = False)	[True/False]

* = Not normally random variables but measured/known.

Table 1: Inputs to orbitCalculator.

2.3. Outputs

Parameter	Description
n	Mean Motion [rad/yr]
M_deg	Mean Anomaly [^{o}]
E_latest_deg	Eccentric Anomaly [^{o}]
TA_deg	True Anomaly [^{o}]
Sep_Dist_AU_OP	Separation Distance in orbital plane [AU]
PA_deg_measured*	Measured Position Angle in image [^{o}]
a1	Semi-major axis of M1 [AU]
a2	Semi-major axis of M2 [AU]

* = For comparison to the observed/measured values.

Table 2: Outputs of orbitCalculator.

2.4. Equations

As said in section 2.1, these equations are the result of combining Kepler's laws, the definitions of the orbital elements and the naturally occurring symmetries and rules of a stable binary system.

First calculate the Mean Motion from the provided period:

$$n = \frac{2\pi}{period} \tag{1}$$

Use the Mean Motion, time of current epoch(t) and time of last periapsis(T) to get the Mean Anomaly:

$$M = n(t - T) \tag{2}$$

Apply Newton's method to calculate E:

$$E\_latest = E\_last - \frac{[M - E\_last + e \times \sin(E\_last)]}{[e \times \cos(E\_last) - 1.0]} \tag{3}$$

The loop completes when E_latest and E_last are the same to 10 decimal places.

Use the resultant E_latest to calculate the True Anomaly:

$$TA = \arccos\left(\frac{[\cos(E\_latest) - e]}{[1.0 - e \times \cos(E\_latest)]}\right) \tag{4}$$

Calculate the Separation Distance in the reference plane in [AU] from the provided Separation Angle ["] and System Distance [PC]:

$$Sep\_Dist\_RP = Sep\_Angle\_measured\_arcsec \times Sys\_Dist\_PC \tag{5}$$

Take Inclination into account to find the Separation Distance in the orbital plane:

$$Sep\_Dist\_OP = \frac{Sep\_Dist\_RP}{[(\sin(argPeri + TA) \times \cos(i))^2 + (\cos(argPeri + TA))^2]^{1/2}} \tag{6}$$

Use the True Anomaly and Separation Distance in the orbital plane to find the X and Y components of the Separation Distance in the reference plane (WRT the Line of Nodes):

$$Sep\_Dist\_RP\_Y = Sep\_Dist\_OP \times \sin(argPeri + TA) \times \cos(i) \tag{7a}$$

$$Sep\_Dist\_RP\_X = Sep\_Dist\_OP \times \cos(argPeri + TA) \tag{7b}$$

Now to find the angle between the Line of Nodes and the primary, corrected for inclination:

$$ang\_AN\_to\_M1\_corr = \arctan\left(\frac{Sep\_Dist\_RP\_Y}{Sep\_Dist\_RP\_X}\right) \tag{8}$$

The final measured Position Angle depends on the which quadrant M1 lies in on a X(Line of Nodes, positive to right in reference plane) Y($\perp$ to Line of Nodes, positive in upwards direction in reference plane) grid. Four quadrant options result in two final equations:

$$Position\_Angle = \begin{cases} LongAN\_deg + ang\_AN\_to\_M1\_corr + 180^0 & \text{if M1 in 1st or 4th quadrant} \\ LongAN\_deg + ang\_AN\_to\_M1\_corr & \text{if M1 in 2nd or 3rd quadrant} \end{cases} \tag{9}$$

Using the Separation Distance in the orbital plane, Eccentricity and True Anomaly to calculate the total of the Semi-major axis (a1+a2):

$$a\_total = \frac{Sep\_Dist\_OP \times (1 + e \times \cos(TA))}{1 - e^2} \tag{10}$$

Finally, if the system is a binary star system (indicated by setting both M1 and M2 to something other than 1), the individual Semi-major axis can be found from the mass ratio and the total of a1+a2 just found:

$$X = \frac{M2}{M1} \tag{11a}$$

$$a1 = \frac{a\_total}{1 + X} \tag{11b}$$

$$a2 = \frac{a1}{X} \tag{11c}$$

If default M1 and M2 are provided, the system is a star-planet and the planet's Semi-major is: a2 = a_total.

3. The MCMC Simulator

3.1. Description

The script mcmcOrbSimulatorUniform2.py is a Markov Chain Monte Carlo Simulator to find the Orbital Elements of a binary system. The simulator uses uniform random numbers for the parameters that are not defined as of observed values at the top of the script and the Markov Chain follows the Metropolis-Hastings algorithm. The script will use the parameters to conduct a MCMC simulation and will finish by plotting and saving the resulting values for the orbits in the chain using the orbElementPlotter and dataWriter functions in the orbitToolbox2.py. The script dataLoadAndPlot.py may be used after a simulation is ran to load the data written to disk in txt files back into memory and plot them using the same plotting function. While the simulator does not conduct statistical analysis on the results yet, once the method to do this is determined it will be easy to implement.

In its current form, it is a script that you would open, edit the parameters and then run it from a terminal using:

```
$ python mcmcOrbSimulatorUniform2.py
```

In the future, this script might be converted into a class object or function that can be called and have the parameters passed into it. Another option is for it to load a parameter dictionary/file which the user adjusts separately.

3.2. Adjustable Parameters

There are different types of parameters that the user may adjust to change the simulation in various ways.

First, for the parameters which adjust the overall way the simulation will go, such as how long it will run or the prints that will go to the screen while it progresses:

Parameter	Description	Suggested Range/Value
numSamples	Number of sample orbits to create	(int)[1e5, 1e8]
verbose	Send prints to screen?	[True/False]
numSamplePrints	Number of chain progress prints to screen	(int)[10, 50]
plotFileTitle	Plot and data file name base	(str)
showPlots	Show plot figures when finished?	[True/False]

Table 3: General Simulator Parameters.

Next are the parameters which set the ranges that the uniform random number inputs may take on:

Parameter	Description	Suggested Range/Value
longAN_degMIN	Minimum Longitude of Ascending Node [^{o}]	0.001
longAN_degMAX	Maximum Longitude of Ascending Node [^{o}]	179.999
eMIN	Minimum Eccentricity	0.001
eMAX	Maximum Eccentricity	0.999
periodMIN	Minimum Period [yrs]	1.0
periodMAX	Maximum Period [yrs]	[100.0, 1000.0]
inclination_degMIN	Minimum Inclination [^{o}]	0.001
inclination_degMAX	Maximum Inclination [^{o}]	179.999
argPeri_degMIN	Minimum Argument of Periapsis [^{o}]	0.001
argPeri_degMAX	Maximum Argument of Periapsis [^{o}]	89.999

Table 4: Ranges for Acceptable Random Number Inputs Parameters.

Last are the input parameters that are measured in the observations or known by other means:

Parameter	Description	Suggested Range/Value
Sep_Angle_arcsec_measured_REALs	List of Separation Angles ["]	[0.01, 2.0]
SA_mean_errors	List of errors in Separation Angles ["]	[0.001, 0.5]
PA_deg_measured_REALs*	List of Position Angles [^{o}]	[0.001, 359.999]
PA_mean_errors*	List of errors in Position Angles [^{o}]	[0.1, 3.0]
epochs	Epochs of observations [julian date]	[2400000.0, 2460000.0]
chiSquareMax	Maximum $\tilde{\chi}^2$ allowed	[1.0, 500.0]
Sys_Dist_PC	System Distance from Earth [PC]	[5.0, 50.0]
Mass1	Mass of primary [Msun](default = 1)	[0.001, 10]
Mass2	Mass of companion [Msun](default = 1)	[0.001, Mass1]

All above values must be of type 'float' or 'double'.
* = Used to calculate $\tilde{\chi}^2$ of proposed orbit during chain.

Table 5: Measured/Known Inputs Parameters.

3.3. Simulator Flow

1: First and initial set of orbital parameters (initial state) that satisfy chiSquareMax is found using the basic Monte Carlo simulator in basicOrbSimulator2.py for the measured values from the first epoch.

With this initial state, the MCMC loop is then entered.

2: A new state is proposed using uniform random numbers and the multiEpochOrbCalc function from orbitToolbox2.py. This function utilizes the single epoch version described in section 2, and also calculates the total $\tilde{\chi}^2$ for this state when applied to the data provided for all epochs observed.

3: The $\tilde{\chi}^2$ for this new state is then used along with that of the initial state in equation (12).

$$\alpha < \frac{chi\_square\_total\_last}{chi\_square\_total\_cur} \tag{12}$$

Where α is a uniform random number between 0.0 - 1.0.

4: If equation (12) is satisfied, then the new state is kept and the process is repeated but using this state's $\tilde{\chi}^2$ value for the chi_square_total_last parameter when calculating equation (12). If equation (12) is not satisfied, then the new state is not kept and the last accepted state remains.

Steps 2-4 are repeated until the loop has been conducted the number of times set using the numSamples parameter. Those states that satisfy the equation are stored to create the 'chain'.

The resulting chain will then have its parameters plotted and saved to disk. If verbose=True, then the median values of these parameters are displayed to the screen to give the user a very rough idea of the simulator's results.

4. Results

Even though this simulator is still in its early phases of development, some initial results can acquired. In order to determine the capabilities of our simulator, it is first being tested against the well studied binary system of β Pictoris following a similar approach to Currie et al. (2011). This system is a star-planet system, and thus the simple case of only one set of orbital parameters must be found. Using all the provided data from Currie et al. (2011), the simulator was ran for 1 Billion sample orbits and the resultant values are summarized in the histograms of figure 3. The median of our resultant values are shown in table 6, along with those of Currie et al. (2011) for comparison. It is important to note that these are simply early testing results and much work is still to needed to complete the simulator and conduct thorough statistical analysis of the results to give accurate predicted values and uncertainties.

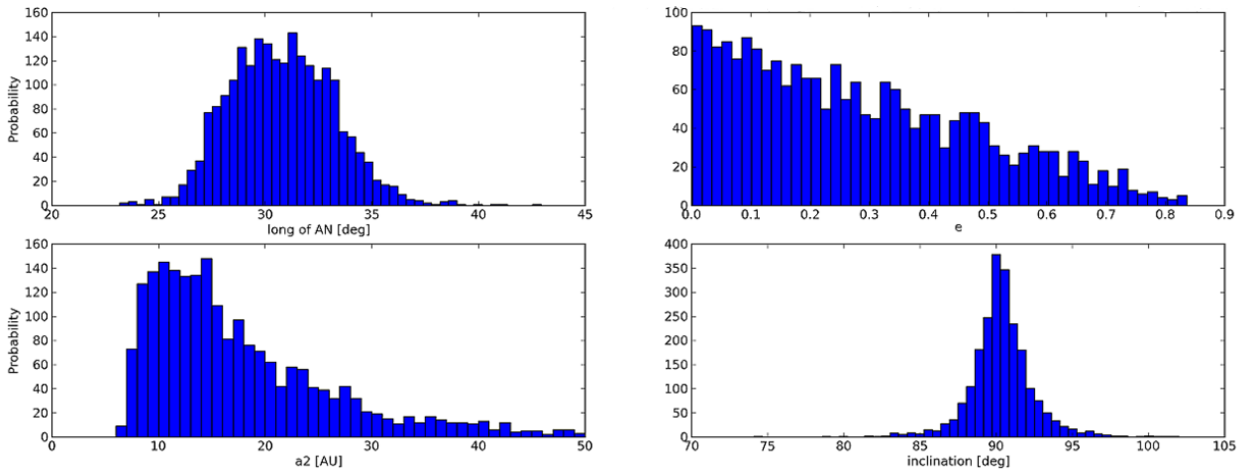

Fig. 3.— MCMC Simulator Results Summary for 1 Billion sample orbits

Parameter	Our Results	Currie et al. Results
Longitude of the Ascending Node[^{o}]	30.7	$30.9^{+31.2}_{-30.6}$
Eccentricity	0.24	$0.12^{+0.31}_{-0.03}$
Semi-major Axis [AU]	15.5	$11.0^{+15.9}_{-8.2}$
Inclination [^{o}]	90.3	$89.5^{+89.7}_{-89.2}$

Table 6: Results Comparison.

5. Discussion

5.1. The Metropolis-Hastings Algorithm and Random Numbers

The specific algorithm applied to make the Markov Chain of this simulator is the Metropolis-Hastings Algorithm. The way this algorithm is applied is described in section 3.3 above. The heart of this algorithm is equation (12) that is used to decide if the currently proposed state will be the next 'link' in the chain. The version of this equation is simplified in this simulator because a uniform random number generator is used to create the input orbital parameters. The original form is shown below in equation (13), where P is the probability distribution of a state and Q represents the proposal density of the a state WRT another state and is dependent on the way the random numbers are generated.

$$\alpha < \frac{P(x\prime)}{P(x_t)} \frac{Q(x_t; x\prime)}{Q(x\prime; x_t)} \tag{13}$$

The first fraction in equation (13) is the ratio of the probability distributions. Another way of describing these, is that they are equal to the probability that a state is the actual set of values for describing the observed measurement(s). In this simulator, equation (14) is used to calculate this probability, where $\tilde{\chi}^2$ is the chi-square given by equation (15), with NE being the total number of epochs the data includes. If the $\tilde{\chi}^2$ of a state is low, that indicates the predicted values very closely match the observed ones and so it is likely that this state is one which describes the orbits; with this description it is then understandable that this ratio is commonly referred to as the 'likelihood ratio'. The value of this ratio has a wide range of values it can take and the larger the value, the better the current state is compared to the previous one and thus the more likely it will satisfy equation (13) and become the next 'link' in the chain.

$$P(x) = \frac{1}{\tilde{\chi}^2} \tag{14}$$

$$\tilde{\chi}^2 = \sum_{i=1}^{i=NE} \frac{(observed_i - model)^2}{(\text{error in observed}_i)^2} \tag{15}$$

The second fraction in (13) is that of the proposal densities. In the case of a uniformly random number, there is no dependence on the previous value when generating a new value. This is one of the key setbacks of using uniform random in comparison to random numbers which follow a more

complicated distribution, such as a gaussian, delta or triangle function. One advantage though, is a uniform distribution is simpler and is therefore faster. Another important property of a uniform distribution is that the proposal density is symmetric, and thus the fraction of these densities is always 1. It is because of this fact that the Q ratio is not present in the version of equation (13) used in this simulator. This is also the case with other symmetric distributions, such as the gaussian which is commonly used proposal distribution in MCMC simulations. That means the only contributor to determining if the current proposed state will be the next 'link' is the likelihood ratio. While this is mathematically valid, it is the minimum amount of information from the proposed state being used, making the chain have weak connections between the 'links' in the chain.

$$\mathcal{N}(\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{16}$$

If a gaussian function, such as that in equation (16), is used for generating the random numbers, then σ tunning must be done during the 'burn in' stage of the Markov Chain. This is the stage of the chain where the information about the state the chain was started from is 'lost' through taking sufficient steps forward such that the current state is significantly different from the initial. During this stage the σ value used must be tunned to result in an acceptance rate of the proposed states being approximately 50%. In the case of a one-dimensional gaussian, a 50% acceptance rate has been theoretically shown to provide the most efficient Markov Chain (Gelman et al. (1994)). To do this, periodically the acceptance rate is calculated and if it is above 50% then the σ value is raised, else it is lowered by some interval and the process is repeated until the acceptance rate is as close to 50% as possible. This process takes up additional time during the simulation and thus choosing to use a gaussian random number generator should be done so knowing the consequences.

Another important consequence of using a gaussian random number generator is the lack of bounds in the values it produces. Because of this, it produces values that are outside the bounds set by the Min/Max parameters outlined in table 2. One method to solve this is to check the resulting value is within the bounds and place the generation of the number and the check within a 'while loop', so the process is repeated until a satisfactory value is found. This approach is quick to generate a satisfactory value if the median of the gaussian is near the center of the allowed values, but if it is near the edges or the σ value is large, then this can take a long time. In some periods of the σ tunning when it has a large value, the 'while loop' must be repeated up to millions of times until a satisfactory value is generated.

A random number generated using a triangle distribution may be a suitable solution to these issues. A triangle function is simple when compared to a gaussian and does not contain a σ parameter to tune, so it would only require a small amount of time to calculate. It also has a median value, and thus the newly proposed states will have some information of the previous one. Another advantage is it is commonly a non-symmetric function and would therefore contribute information about the previous and current state to equation (13), creating stronger connections of the 'links' in the chain. The issue of bounds on the output values is also solved because the function requires not only a median but minimum and maximum values as well. While a triangle random number generator has not been tested in this simulator, it presents a possible future improvement if no serious negative consequences with its use are found theoretically. For the time being, the simulator will continue to use uniform random numbers until further investigations can be made.

5.2. Calculating $\tilde{\chi}^2$ and Future Work

The method by which $\tilde{\chi}^2$ is calculated can make a large impact on the results.

In this model the $\tilde{\chi}^2$ being currently used is the standard chi-square statistic shown in equation (15). This formula is actually missing a key element, the number of degrees of freedom ν = N - m, where N is the number of observables being fit to the model with m parameters (Bevington & Robinson (2003)). Including this factor makes it the reduced chi-square statistic, given by equation (17). In the case of the model being a very close approximation to the real observed system, the $\tilde{\chi}^2$ should be nearly unity, $\tilde{\chi}^2 = 1$. A value for $\tilde{\chi}^2$ less than 1 does not indicate a better fit of the model and in fact points to a possible error in the assigned uncertainties of the measured observables and/or the model itself (Bevington & Robinson (2003)). While this simple inclusion of ν normally just reduces the value of the resulting $\tilde{\chi}^2$, in our model nine of the input parameters contribute to the calculation of the output measured Position Angle that is used in calculating $\tilde{\chi}^2$. Since the current data set we are testing the model against only has 4 epochs of observation, this makes the ν value negative and thus $\tilde{\chi}^2$ negative. A negative value for the degrees of freedom does not make logical sense and a negative $\tilde{\chi}^2$ implies the model is better than reality, which is also not realistic.

$$\tilde{\chi}^2 = \frac{1}{\nu} \sum_{i=1}^{i=NE} \frac{(observed_i - model)^2}{(\text{error in observed}_i)^2} \tag{17}$$

This negative ν problem must be further investigated and solved. One obvious solution would be to observe the system at more epochs than there are model parameters as that would make N > m and thus a positive ν. While this solution is simple, it requires many years and telescope time to aquire that many observations of the system. Another option is to develop a model that requires less input parameters than the number of observed epochs. In our case of four available observations, it makes it very difficult, or even impossible, to develop such a model. One further solution would be to create a model that is capable of predicting more values that could be compared to those observed in the $\tilde{\chi}^2$. This solution would have the greatest effect on the value of ν, as with each observable the value of N goes up by N = (number of observables)X(number of observations), so with only one more observable the value of N doubles, in our case. The most efficient approach is to combine these last three to make a model that not only requires less parameters, but also outputs more predicted values to compare to the new observed values from the data. For the time being, this remains as further work and equation (15) without ν will be used until this new model is ready.

One further check of our $\tilde{\chi}^2$ values would be to apply an F-test on them. The statistic $\tilde{\chi}^2$ includes components of not only the model but also the observations making it difficult to test just the model alone. An F-test combines two different methods of calculating $\tilde{\chi}^2$ and then compares the results to see how reasonable their relation is (Bevington & Robinson (2003)). The highest priority is to correct the model itself, but following that up with a thorough F-test would ensure our final results are valid.

Another area that requires further work is the analysis of the results from our simulator. The over simplified histograms of figure 3 are not on their own meaningful until statistical analysis is applied to determine the best predicted values and their errors. To do this, many types of comparisons of the parameters must be made and confidence levels determined. As mentioned in section 4, the histograms

are mearly a method for getting a rough preliminary idea of the simulator's results and can easily be seen as irrelevant when looking at the one for the Eccentricities in figure 3.

6. Conclusion

We have written a MCMC simulator, composed of a main simulator script, mcmcSimulatorUniform2.py, and an orbit calculator function contained in the orbitToolbox.py toolbox. While the results of our simulator are preliminary, they do show some agreement to those found by Currie et al. (2011), giving us hope that we are on the right track. In order to start getting more accurate results, there is much work to be done. The orbit calculator, also refered to as our 'model', needs to be reviewed and modified to not only take less input parameters but also output more predicted observable values. More measurements must be made from the current data to create more observable values to compare to those output by the model such that the resulting ν value is positive and may be included in our $\tilde{\chi}^2$ calculations. Once the chain of our simultor is producing resulting parameter sets near the end with $\tilde{\chi}^2$ values close to unity, indicating our model very closely matches the actual binary system's orbits, we can conduct proper statistical analysis of the resulting parameters to give us peak values and their uncertainties. Following the thurough testing of the simulator on the data of β Pictoris system investigated by Currie et al. (2011), it must be tested against a binary star system and then finally on our own data of the complicated τ boo system.

REFERENCES

Bevington, P., & Robinson, D. 2003, Data reduction and error analysis for the physical sciences, McGraw-Hill Higher Education (McGraw-Hill)

Currie, T., Thalmann, C., Matsumura, S., Madhusudhan, N., Burrows, A., & Kuchner, M. 2011, ApJ, 736, L33+

Gelman, R., , Roberts, G. O., Gelman, A., & Gilks, W. R. 1994, Weak Convergence And Optimal Scaling Of Random Walk Metropolis Algorithms

This preprint was prepared with the AAS LaTeX macros v5.2.