Bugs, Updates, and Upgrades

These notebooks are constructed at a typical undergraduate student level and are not considered to be commercial grade programs. All have been tested using Mathematica V. 9 and found to be operable, but there is no guarantee that they will remain so given future operating system Mathematica version changes. If you wish to report a bug in the original notebooks (not modified ones, please), email the Author at ddmeisel@frontiernet.net with details.

Before running this notebook be sure that HD108613.csv from the "Put into User Folder" are placed in the computer user folder so that *Mathematica* will be able to find the raw data.

Mass - Another Fundamental Property of Stars

```
Clear["Global`*"]
```

```
tA = SessionTime[];
```

A Word about Notebook Organization

This notebook and this commentary appears in Ch. 1 because mass determination is absolutely fundamental to modern astrophysics, not because the topic is easy or its explanation simple. But being firmly rooted in classical not relativistic mechanics should make it seem somewhat more familiar. Because we will be using actual observations within the computations, the subject of error analysis comes up rather abruptly here and computation of errors is always a complicating factor to any work, as undergraduate students well know. To perform a correct error analysis on your own is even more harrowing than doing it under an instructor's direction and when that is mixed in with a long and complicated solution to a physical problem, the result seems hopelessly difficult for the novice.

The organization we have chosen here and throughout these notebooks for our computational solutions is basically an algorithmic one, being more like a receipe than an exposition. They are after all basically computer programs. In an algorithm, input information needs to appear just before it is required in the progression of computation without regard to whether it was easy or hard to obtain or whether it needed lots of preprocesing and editing. Some programming languages require a sort of receipe ingredient list through variable declarations, but not *Mathematica.* In fact, in using *Mathematica* we often unwittingly contributes to the equation and comment clutter because in order to save intermediate results in a particular symbolic label that reminds the memory of what it represents, one has to keep changing the variable's name between steps of a calculation. This quirk often occurs because *Mathematica* in common with lots of computer languages allows iteration of the same variable across the regular equal sign. Like other computer languages, one can also hide long stretches of "code" within the *Mathematica* package construct and thus clean up the mess that we show publicly for all to see. But this hiding feature does not just show what needs to be seen for instructional purposes in preference to the "scratch pad" work that went on around it. You either hide stuff or you don't. We do go part way toward hiding programming content by using the built-in *Mathematica* constructs so many mathematical details that in older texts often obscured the astrophysics and physics themselves are at least now hidden from view.

So what can a novice student do ? Our suggestion is to read through the notebooks at first even if you intend to modify or run it and then outline the contents in a logical fashion correlated with the textbook. Most solutions consist of several parts: a) the geome-

try, b) the physics, c) the astronomy, and d) the astrophysics. Actually because our notebooks are pseudo algorithms, the most appropriate means would be to construct the equiva lent of “flow charts”. Thus reading our notebooks requires the same skills that one would use in reading any computer program. But unlike a normal computer program, we link the parts (sometimes extensive sections) together with verbal comments and section headings. Some of our notebooks are fairly simple, others like this one that incorporates actual data is very complicated, particularly if an error analysis is incorporated within it. You may not like to do error analysis, but at least you have *Mathematica* to do all the grunt work for you and ensure, that aside from typographical errors, the computation is done correctly. There is at least one bridge builder in prison because an error analysis was done incorrectly and people died as a result. So for a professional scientist, error analysis must ALWAYS be taken seriously.

Mass Estimates from the Orbit of a Spectroscopic Binary: A Case History of a Preliminary Astrophysical Study.

Introduction

Mass is a property of material objects said to be endowed by the Higgs boson of elemenetary paticle theory. Mass is the source of gravitational fields throughout the cosmos and everything that has mass contributes to this field. In General Relativity, mass curves space-time and this is where gravitational fields orignate. But even on a less sophisticated scale, unless born on an orbiting space craft and living continuously in space, humans intuitively, but perhaps qualitatively, know the effects of gravity and how moving masses are influenced by it. Because gravity is such a long range force, its effects are felt across immense distances. In this notebook, we explore a number of issues illustrating how the concept of mass occurs in the study of spectroscopic binaries and its value determined within astronomy and astrophysics.

Context of Single Line Spectroscopic Binaries within the Mass Determination Problem

If the radial velocity of one of two visual binary components can be determined from spectroscopic observations, then the angular orbital ambiguities in the orbital inclination and the ascending node Ω encountered in the analysis for orbital elements can be resolved directly. And if radial velocity information is available for both stars, the mass ratio can also be obtained as an alternative to using astrometric data on proper motions.

Can radial velocity information stand alone ? Unfortunately the radial velocities in a system of two stars may not yield the masses directly because the inclination is not derivable unless the system is also either a visual or eclipsing binary. But even in principle, the

analysis of a visual binary system is more complicated than for a spectroscopic binary so we defer consideration of that case until Ch. 6 where various types of stellar motion are considered.

The Fundamental Equation

But spectroscopic methods applied even to single line binaries can obtain the orbital period, the eccentricity, the daily motion n, and the argument of periapsis ω uniquely. They can also determine which nodes are which and the sign of the inclination when used in conjunction with visual or interferometric observations. The equation for orbital radial velocity (where z is along the line of sight, n is the daily (not yearly motion) = 2 π/T(days), a is the semi-major axis, i is the inclination, ν is the true anomaly, e the eccentricity, and ω the argument of periapsis) is

$$\frac{dz}{dt} = \frac{n\, a \sin i}{\sqrt{1-e^2}} \left(\cos(\nu+\omega) + e \cos\omega\right)$$

The observed radial velocities will have the motion of the center of mass in each and this is determined so that the line v=constant divides the radial velocity curve into two equal areas. This has to be done first so that dz/dt is isolated. Creating a smoothed radial velocity curve from the observations is our first challenge. But first a digression about errors.

Rudiments of Error Analysis

Although there may be a perception that theoretical study of a subject through computation can be accomplished without comparing with observations, scientific computation requires constant checking of the predicted results against reality through statistical analysis of available data. But there are often vast problems. First getting a realistic and correct error analysis is often more difficult than just getting the actual solution desired. Secondly the predicted quantities are often not the observational quantities, but rather physical quantities derived from the observations through theoretically derived formulae. The errors of such theoretical quantities are obtained through a process called **the propagation of errors**, a subject whose elementary rules with which most students who have taken an undergraduate physics or chemistry laboratory are familiar. Thirdly error analysis is not well understood by most students (and faculty ?) without lots of study and work. That is why we bring this up right in the beginning. We review these important concepts here before showing how we calculate them for a faint spectroscopic binary system where errors abound.

There are two main types of errors. Although real situations are usually a mixture of the two, we will consider the extremes where only one type is dominant:

a) **Independent Errors** - These are errors that occur independently of each other so that their linear correlations (as measured by the so-called covariance) are zero. Random errors are usually of this type.

b) **Correlated Errors** - These are errors that occur in "synchronism" with each other so that their interdependence is measured by their covariances (not zero). Systematic and bias errors are usually of this type.

Functions through which these errors can propagate are either **linear** or **non-linear**. Linear functions are things like polynomial sums. Non-linear functions are things like ratios of polynomial sums or transcendental functions. Error analysis can either be linear or nonlinear. Least squares analysis in *Mathematica* is also either linear or non-linear as we will be showing. Usually error analysis assumes the errors are "small enough" that 1st order linear theory can be applied and hence only first order partial derivatives are involved. Linearized error propagation is a well known technique (See Ch. 10 of Meyer (1975) for an excellent and full discussion.) and its relationship to least-squares is straight forward. But as Meyer points out that there are nuances arising from special situations that can lead one astray and the methods for handling such special cases are usually not found in the elementary treatments. In that case one is fairly much at the mercy of the math. For example:

a) For **random** or independent errors, the "standard" view point is that the errors are strictly independent (the covariance $S_{x(i)\,x(j)}$ between them is zero). There are instances where observational variables are partially correlated and if this is known more sophisticated analyses must be applied. (See Meyer for details.) If the errors can be treated as random, the general algorithm is to use the following whether the function through which the propagation is desired is linear or not:

$$S_{f\{x(i)\}} = \sqrt{\sum_i \left(\frac{\partial f(\{x(i)\})}{\partial x(i)} S_{x(i)}\right)^2}$$

Specific to a function of two variables:

$$S_z = \sqrt{\left(\frac{\partial f(x,y)}{\partial x} S x\right)^2 + \left(\frac{\partial f(x,y)}{\partial y} S y\right)^2}$$

a) Sums and differences

$$S_z = \sqrt{(S x)^2 + (S y)^2}$$

b) Products and quotients

$$\frac{S_z}{z} = \sqrt{\left(\frac{S_x}{x}\right)^2 + \left(\frac{S_y}{y}\right)^2}$$

Extension to other functions is a straight forward application of the first rule, examples of which will be shown below.

a) For **correlated** errors including systematic errors, Meyer discusses several ways to deal with them.

1) The simplest way given by Meyer is to treat the errors as predicted by the chain rule of the calculus keeping the negative signs if any. Then the error is the absolute value of the final answer. This is similar to what is done for the mean deviation as opposed to the rms deviation for the independent errors. This is very easy to implement via modification of the uncorrelated formula but in a least squares situation it is rare for the coefficients to be perfectly correlated rather they are related through the covariance function.

2) Another method (and one that we have chosen to implement below) assumes that the errors are linearly and first order correlated between the variables two at a time. Since the errors are correlated the covariances $S_{x(i)\,x(j)}$ between the variables are not zero. The problem with this second situation is that one must know the covariance matrix for all variables going into the propagation. While not a problem for least-squares fits particularly when *Mathematica* can give the requires values for the function coefficients to you through the options, “CorrelationMatrix” or “CovarianceMatrix”, it is a problem for all the other variables used in the propagation formulae. The general algorithm is to use the following whether the function through which the propagation is desired is linear or not:

$$S_{f\{x_i\}} = \sqrt{\sum_{i,j}^{n} \left(\frac{\partial f\{x_i\}}{\partial x_i}\right)\left(\frac{\partial f\{x_j\}}{\partial x_j}\right) S_{x_i\,x_j}}$$

where $S_{x_i\,x_i} = S_{x_i}^2$

Specifically to a function of two variables where ρ_{xy} is the linear correlation coefficient between x and y.

$$S_z^2 = \left(\frac{\partial f(x,y)}{\partial x}\,S\,x\right)^2 + \left(\frac{\partial f(x,y)}{\partial y}\,S\,y\right)^2 + 2\,\frac{\partial f(x,y)}{\partial x}\,S\,x\,\frac{\partial f(x,y)}{\partial y}\,S\,y$$

a) Sums and differences z=x±y

$$S_z = \sqrt{S_x^2 + S_y^2 \pm 2\,\rho_{x\,y}\,S_x\,S_y}$$

b) Products

$$\frac{S_z}{z} = \sqrt{\frac{S_x^2}{x^2} + \frac{S_y^2}{y^2} + \frac{2\,\rho_{xy}\,S_x\,S_y}{x\,y}}$$

c) Quotients

$$\frac{S_z}{z} = \sqrt{\frac{S_x^2}{x^2} + \frac{S_y^2}{y^2} - \frac{2\,\rho_{xy}\,S_x\,S_y}{x\,y}}$$

A main objection often raised in practice to these formulae is that the derived errors can be smaller than the random errors formula involving the same variances and that is a result that leads to the opinion that systematic errors can be more safely ignored than random errors of the same equations. But that is how correlated errors work. Uncorrected correlated errors treated as independent ones are larger than the equivalent independent errors because the error of one variable is “aliased” into its associated variables. When there are mixtures of the two type of errors, the true error falls somewhere in between the two extremes. The dilemma is chosing how to combine the two results. Lacking a priori information usually leads to a simple rms combination of random and systematic. Since the treatment of systematic effects is so controversial, investigators often want to err on the side of conservatism and deal only with the random errors but this makes the derived errors larger than they should be.

c) For **non-linear propagation of errors**, the true situation can be much more complicated than the first order expansion would indicate even in the independent error case. Meyers derives an expression for propagation involving second order derivatives and stresses that it MUST be used if the first order derivatives are zero, i.e. the case of function minimization for example.

As computational analyses have become more sophisticated and wide spread, it now possible to apply Monte Carlo methods to error analysis. For example, one can use a random (really pseudorandom) number generator to calculate a set of both variable errors and function errors and apply them to the mean or derived parameter values. These errors are then applied to the evaluation of the function itself multiple times. The set of “new” values are then calculated as sample means and the propagated errors then become the standard deviation of those trial runs.

The most difficult decision in Monte Carlo analyses is what distribution function and what range of variable is to be used. Fortunately, *Mathematica* provides a wide variety of non-uniform pseudorandom generators so the proper choice is usually available. If observational or experimental errors are to be evaluated, the Gaussian pseudorandom number generator may be appropriate, not the usual uniform one. And even if a suitable generator is available, the parameters of the distribution is not always obvious. In theoretical and computational studies that have no real data available, Monte Carlo error estimation is used to indicate how sensitive given results are to errors both computational and observa-

tional but without a knowledge of what values will occur in practice such simulations may be ad hoc. Thus Monte Carlo methods can be more of an art than science, but since undergraduate students rarely encounter either correlated or Monte Carlo errors we give an example of each below.

The Observations and Their Preparation for Analysis

In this notebook we are following closely the analysis procedure given in Smart (1960) where it is shown that integrals of the single line radial velocity curve are related to the orbit parameters in the dz/dt equation above. If the radial velocity curve for the star being studied has high signal to noise and is relatively smooth with points fairly closely spaced, getting the required integrals is a matter of numerical integration, a task for which Mathematica is well suited. But the example we have chosen has a very noisy and sparsely populated radial velocity curve as is typical of modern astrophysical exploratory research and the challenges such data presents. It is this type of data that research scientists give to their undergraduate summer students to test their resourcefulness and so is a good example for us to consider. Our goal is to (a) take a limited, non-optimal data set, (b) produce a relatively smooth radial veleocity curve by piecewise least-squares curve fitting if necessary, (c) perform the necessary integrations analytically with the proper computations of the errors at each step, and (d) deduce the orbital properties (actually the short and easy part) giving the errors with the final results. Does this sound like work ? Well it is but fortunately the tricky math and arithmetic is handled quite easily by *Mathematica* to produce results that compare favorably with the published results. In a broad outline, the steps carried out here will be typical for our notebooks involving both theory and observation.

Importing spectroscopic binary data

There is a treasure trove of spectroscopic binary data both single line and double line continuing to be published in journal “The Observatory” by R.F. Griffin as a long series of papers describing continuations of work by Yoss and Griffin (1997, JA and A., **18**,161). The observations have been obtained with photoelectric spectrum scanners and many of very long period have been published recently. These distant stars if located closer to the sun would have been seen as visual binaries. Hence it is appropriate to consider one of them as our spectroscopic binary example. We have chosen HD 108613 observed from 1973 to 2010 (Griffin, 2010, Obs. **130,** p.358). The file consists of Julian Date and velocity in km/s only and will be used with unit weights. No spectrum type is available and there is evidence of photometric variability of unknown origin. Only the spectrum of one star can be seen which would seem to make the analysis simpler but alas it does not as will be seen.

It is worth pointing out for the novice that Griffin derived a set of orbital elements for this star but like the weights no element errors were published. Here we have carried a

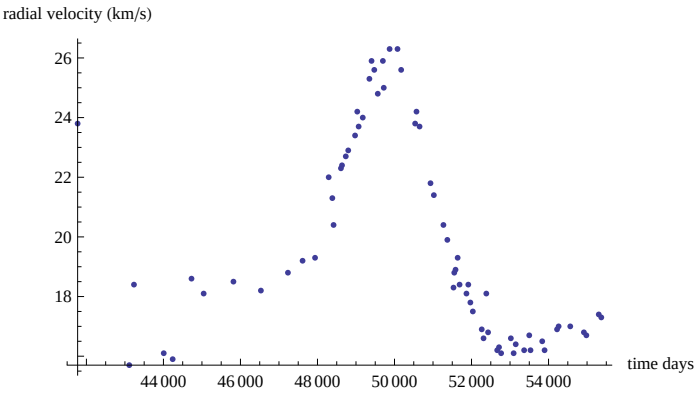
completely independent solution of the orbital elements using *Mathematica.* Unlike Griffin, we have provided a complete error analysis for all quantities including the usual independent errors, the linear correlation corrected errors and finally Monte Carlo derived non-linear errors. The required *Mathematica* expressions are complex, ugly and except for the Monte Carlo method resulted in totally outrageous estimates of the final errors. It is no wonder that Griffin, if he calculated formal, independent errors in the usual manner, did not publish any error estimates at all for his elements.

```
dataSS = Import["HD108613.csv"];
```

```
db1num = Length[dataSS]
```

```
70
```

```
ListPlot[dataSS, PlotRange → All,
 AxesLabel → {"time days", "radial velocity (km/s)"}]
```



This time series is not orthogonal. We need to see if the original points are linearly correlated or not. Hence

```
linfit = LinearModelFit[dataSS, {1, t}, t]
```

```
FittedModel[ 36.6351 − 0.00033362 t ]
```

The linear correlation coefficient is

```
rR = √linfit["RSquared"]
```

```
0.31003
```

Certainly the observations are somewhat linearly correlated. This almost guarantees that the power series coefficients will be strongly correlated also and that has serious consequences for the subsequent error analysis that will lead us astray for a while in the data analysis process.

Determination of the Period

The first task is to determine the period. We have selected 4 points where the

velocity was either 16.1 or 15.9. These were JD 44007.97, 44240.14, 53094.05, and 52768.97.

```
jD = {44 007.97, 44 240.14, 53 094.05, 52 768.97};

ΔT1 = jD[[4]] - jD[[2]]
```

```
8528.83
```

```
ΔT2 = jD[[3]] - jD[[1]]
```

```
9086.08
```

Here is the mean period in days (Griffin gets 8973 ± 157 days)

$$\Delta T3 = \frac{(jD[[4]] + jD[[3]])}{2} - \frac{(jD[[1]] + jD[[2]])}{2}$$

```
8807.46
```

Here is the mean period in years

```
ΔT3 / 365.25
```

```
24.1135
```

Here is the mean deviation on the period (days)

```
ΔΔT = ΔT2 - ΔT3
```

```
278.625
```

Here is the mean deviation on the period (years)

```
ΔΔT / 365.25
```

```
0.762834
```

```
ΔΔT / ΔT3
```

```
0.0316351
```

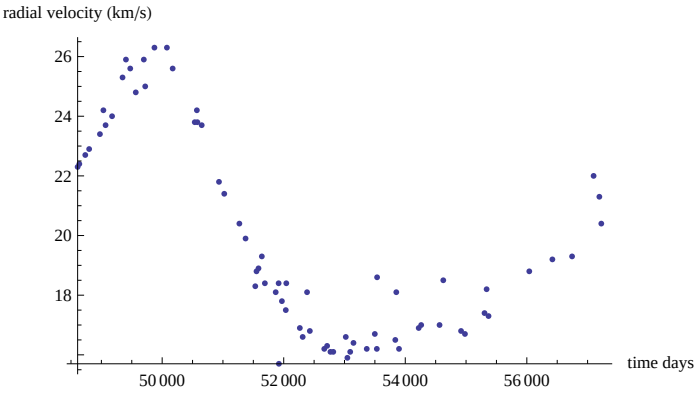
The raw data needs to be resynchronized.

```
dataS1 = dataSS;

numss = Length[dataS1]
```

```
70
```

```
Do[If[dataS1[[ii, 1]] < 48 500,
   dataS1[[ii, 1]] = dataS1[[ii, 1]] + ΔT3;], {ii, 1, numss}];

dataS2 = Sort[dataS1];
```

```
ListPlot[dataS2, PlotRange → All,
 AxesLabel → {"time days", "radial velocity (km/s)"}]
```




```
start = dataS2[[1, 1]]
```

```
48 609.2
```

Next we extend the data set past one period so that we can do the integration needed to find the γ velocity (the motion of the center of mass)

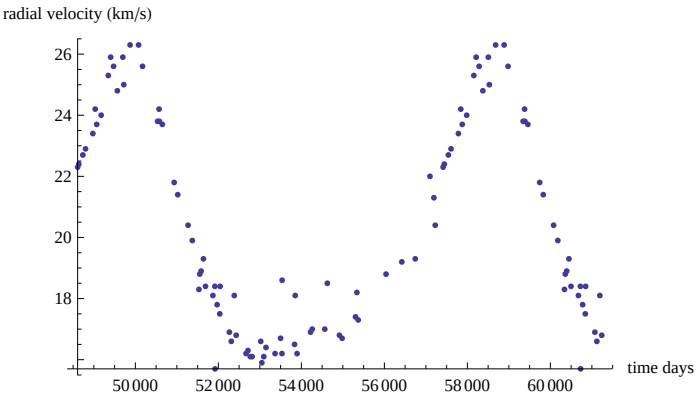
```
dataS3 = Table[{0, 0}, {ii, 1, numss + 40}];
```

```
Do[dataS3[[ii]] = dataS2[[ii]], {ii, 1, numss}]
```

```
Do[dataS3[[ii + numss]] = dataS2[[ii]], {ii, 1, 40}]
```

```
Do[dataS3[[ii + numss, 1]] = dataS2[[ii, 1]] + ΔT3, {ii, 1, 40}]
```

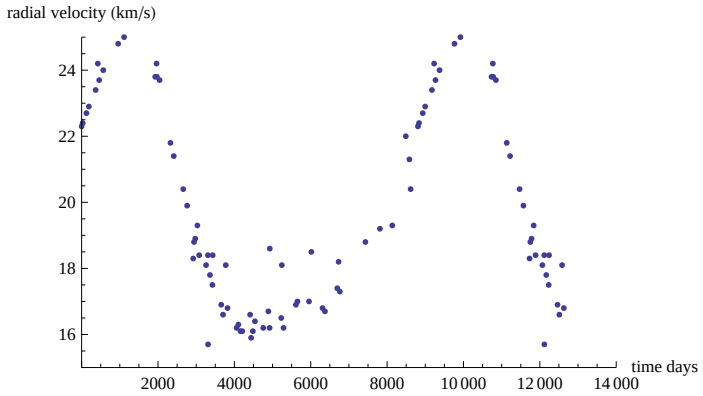
```
ListPlot[dataS3, AxesLabel → {"time days", "radial velocity (km/s)"}]
```



We redo the scale in days and replot.

```
Do[dataS3[[ii, 1]] = (dataS3[[ii, 1]] - start), {ii, 1, numss + 40}]
```

```
ListPlot[dataS3, PlotRange → {{0, 14000}, {15, 25}},
 AxesLabel → {"time days", "radial velocity (km/s)"}]
```



Piecewise Least Squares Smoothing of the Velocity Curve

This task is made difficult because there is no “standard” shape for a radial velocity curve. The best one can do is obtain some sort of self-consistant polynomial approximation. The velocity curve of HD 108613 is extremely noisy because the star itself is faint and perhaps very distant. In order to smooth this curve and reduce the observational scatter we will first fit segments of its light curve with cubic polynomials with no constant term. If the observations had been sampled at equal intervals, this smoothing could have be accomplished by Fourier smoothing techniques.

1) Smoothing the velocity minimum curve

a) Here is the minimum “plus” portion fit to a cubic

```
curvevp = Table[{dataS3[[ii, 1]], dataS3[[ii, 2]]}, {ii, 40, 80}];
```

This is how we specify how the fit is to be done

```
xfit1 = LinearModelFit[curvevp, {1, x, x^2, x^3}, x, IncludeConstantBasis → False]
```

$$\text{FittedModel}\left[1.5942 + 0.00795247\,x - 1.43564\times10^{-6}\,x^2 + 9.04331\times10^{-11}\,x^3\right]$$

This is a summary of the results in standard statistical format

```
xfit1["ParameterTable"]
```

	Estimate	Standard Error	t-Statistic	P-Value
1	1.5942	6.86487	0.232225	0.817643
x	0.00795247	0.0033125	2.40075	0.0215044
x^2	-1.43564×10^{-6}	5.13168×10^{-7}	−2.79761	0.00812393
x^3	9.04331×10^{-11}	2.55677×10^{-11}	3.53701	0.00110924

This is the correlation coefficient of the fit

```
√xfit1["RSquared"]
```

```
0.981898
```

These are the parameters (compare with the table).

```
cfit1 = xfit1["BestFitParameters"]
```

$\{1.5942, 0.00795247, -1.43564 \times 10^{-6}, 9.04331 \times 10^{-11}\}$

These are the errors (compare with the table).

```
σfit1 = xfit1["ParameterErrors"]
```

$\{6.86487, 0.0033125, 5.13168 \times 10^{-7}, 2.55677 \times 10^{-11}\}$

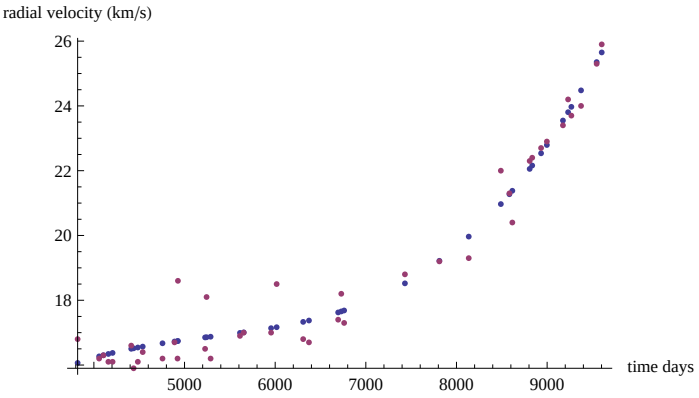
Here is the equation in standard *Mathematica* form for additional use.

```
eqnss1 = Normal[xfit1]
```

$1.5942 + 0.00795247\, x - 1.43564 \times 10^{-6}\, x^2 + 9.04331 \times 10^{-11}\, x^3$

Here are the points and the plot.

```
graphss1 = Table[
   {curvervp[[ii, 1]], eqnss1 /. x -> curvervp[[ii, 1]]}, {ii, 1, Length[curvervp]}];

ListPlot[{graphss1, curvervp}, AxesLabel → {"time days", "radial velocity (km/s)"}]
```



b) Here is the minimum “minus” portion fit to a cubic

```
curvervm = Table[{dataS3[[ii, 1]], dataS3[[ii, 2]]}, {ii, 20, 45}];

xfita = LinearModelFit[curvervm, {x, x^2, x^3}, x, IncludeConstantBasis → False]
```

FittedModel$\left[0.0298642\, x - 0.0000119634\, x^2 + 1.37756 \times 10^{-9}\, x^3\right]$

```
xfita["ParameterTable"]
```

	Estimate	Standard Error	t-Statistic	P-Value
x	0.0298642	0.00123737	24.1352	7.75491×10^{-18}
x^2	−0.0000119634	7.42711×10^{-7}	−16.1077	5.09545×10^{-14}
x^3	1.37756×10^{-9}	1.09384×10^{-10}	12.5938	8.37963×10^{-12}

```
√xfita["RSquared"]
```

0.999226

```
cfit1a = xfita["BestFitParameters"]
```

$$\{0.0298642, -0.0000119634, 1.37756 \times 10^{-9}\}$$

```
σfit1a = xfita["ParameterErrors"]
```

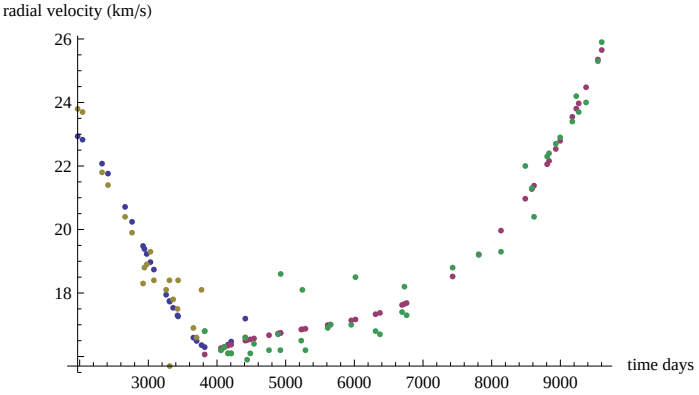
$$\{0.00123737, 7.42711 \times 10^{-7}, 1.09384 \times 10^{-10}\}$$

```
eqnssa = Normal[xfita]
```

$$0.0298642\, x - 0.0000119634\, x^2 + 1.37756 \times 10^{-9}\, x^3$$

c) Here is combined minimum velocity

```
graphssa = Table[
   {curvervm[[ii, 1]], eqnssa /. x -> curvervm[[ii, 1]]}, {ii, 1, Length[curvervm]}];

ListPlot[{graphssa, graphss1, curvervm, curvervp},
 AxesLabel → {"time days", "radial velocity (km/s)"}]
```



It certainly looks like the two curves overlap at some point. Let's examine the files from each to see if they indeed have a common point of interersection.

```
graphss1
```

```
{{3820.74, 16.0649}, {4057.95, 16.2672}, {4104.87, 16.3025},
 {4159.78, 16.3421}, {4206.24, 16.3741}, {4412.02, 16.5013},
 {4438.4, 16.5161}, {4484.86, 16.5413}, {4538.74, 16.5693}, {4757.05, 16.6717},
 {4889.78, 16.7268}, {4922.76, 16.7399}, {4927.24, 16.7416},
 {5226.82, 16.8525}, {5243.13, 16.8583}, {5287.77, 16.8741}, {5611.75, 16.9923},
 {5652.77, 17.0081}, {5954.84, 17.1376}, {6016.21, 17.1674}, {6308.82, 17.3322},
 {6372.74, 17.374}, {6694.81, 17.624}, {6729.18, 17.6551}, {6760.74, 17.6844},
 {7431.31, 18.5218}, {7811.42, 19.2178}, {8135.26, 19.9655}, {8489.46, 20.9692},
 {8584.18, 21.2734}, {8616.18, 21.3799}, {8807.46, 22.055}, {8835.36, 22.1592},
 {8933.29, 22.537}, {8997.14, 22.7934}, {9174.46, 23.5491}, {9231.35, 23.8054},
 {9267.32, 23.971}, {9374.18, 24.4796}, {9546.43, 25.353}, {9602.35, 25.6511}}
```

```
graphssa
```

```
{{1970.28, 22.9355}, {2041.68, 22.8284}, {2325.78, 22.0755},
 {2412.67, 21.7607}, {2661.18, 20.7127}, {2763.03, 20.2416},
 {2923.05, 19.4819}, {2942.97, 19.387}, {2975.98, 19.2302}, {3030.77, 18.9717},
 {3080.74, 18.7389}, {3259.07, 17.9464}, {3307.02, 17.7477}, {3311.5, 17.7295},
 {3360.96, 17.5336}, {3424.82, 17.2947}, {3433.29, 17.2643}, {3656.05, 16.5949},
 {3702.9, 16.4908}, {3774.79, 16.3598}, {3820.74, 16.2956}, {4057.95, 16.2389},
 {4104.87, 16.2882}, {4159.78, 16.374}, {4206.24, 16.4712}, {4412.02, 17.1942}}
```

```
graphssa[[22]]
```

```
{4057.95, 16.2389}
```

```
graphss1[[2]]
```

```
{4057.95, 16.2672}
```

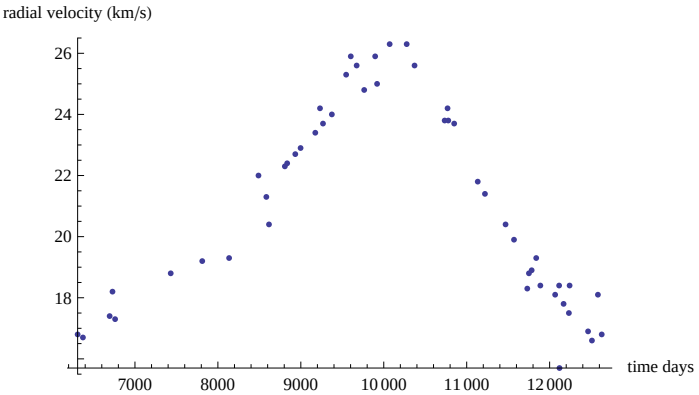
Yes indeed they do. graphssa [[22]] is identical to graphss1 [[2]]. The x value where this happens is x = 4057.9 . Thus the minimum part of the velocity curve can be synthesized piecewise with two cubics.

2) Smoothing the velocity maximum curve.

Now we examine the maximum part of the velocity curve to see if this can be smoothed in a similar manner. First we plot the whole range for reference without excessive scrolling.

```
curverv2 = Table[{dataS3[[ii, 1]], dataS3[[ii, 2]]}, {ii, 60, 110}];
```

```
plotdata = ListPlot[curverv2, AxesLabel → {"time days", "radial velocity (km/s)"}]
```

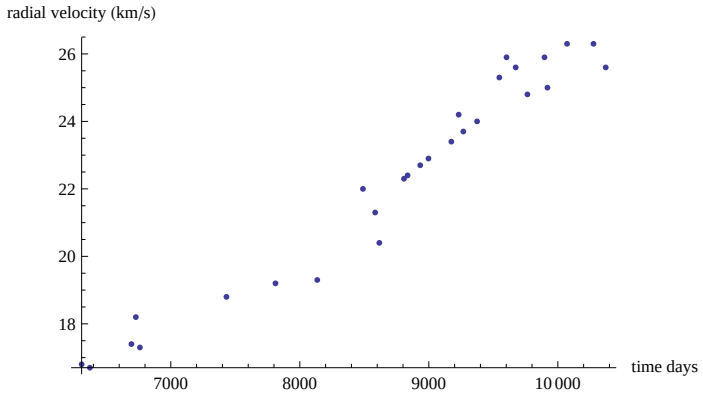


a) Here is the minimum “minus” portion fit to cubic

Next we subdivide this at approximately the maximum by trial and error

```
curverv2a = Table[{dataS3[[ii, 1]], dataS3[[ii, 2]]}, {ii, 60, 87}];
```

```
plotdata = ListPlot[curverv2a, AxesLabel → {"time days", "radial velocity (km/s)"}]
```




```
xfit2a = LinearModelFit[curverv2a, {x, x^2, x^3}, x, IncludeConstantBasis → False]
```

FittedModel[$0.00388753\,x - 3.21868\times10^{-7}\,x^2 + 1.90944\times10^{-11}\,x^3$]

```
xfit2a["ParameterTable"]
```

	Estimate	Standard Error	t-Statistic	P-Value
x	0.00388753	0.000821538	4.73202	0.0000746228
x^2	-3.21868×10^{-7}	1.96386×10^{-7}	−1.63895	0.113749
x^3	1.90944×10^{-11}	1.15822×10^{-11}	1.6486	0.111742

```
√xfit2a["RSquared"]
```

0.999579

```
cfit2a = xfit2a["BestFitParameters"]
```

$\{0.00388753, -3.21868\times10^{-7}, 1.90944\times10^{-11}\}$

```
σfit2a = xfit2a["ParameterErrors"]
```

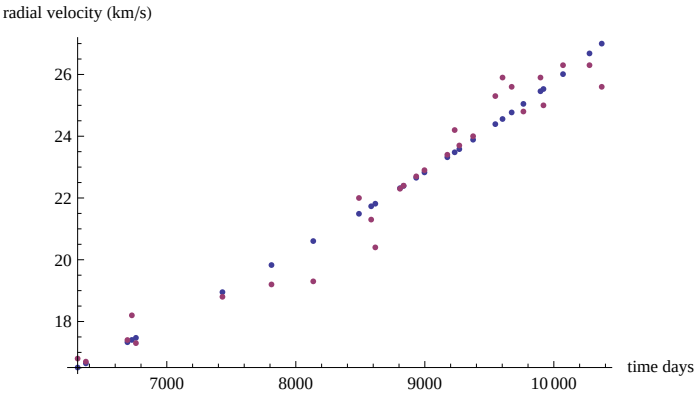
$\{0.000821538, 1.96386\times10^{-7}, 1.15822\times10^{-11}\}$

```
eqnss2a = Normal[xfit2a]
```

$0.00388753\,x - 3.21868\times10^{-7}\,x^2 + 1.90944\times10^{-11}\,x^3$

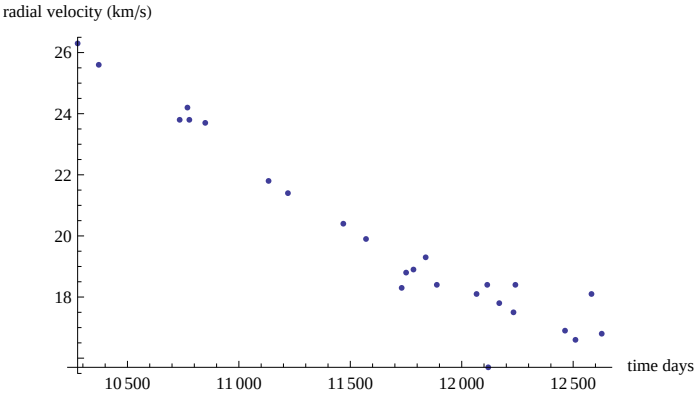
```
graphss2a = Table[{curverv2a[[ii, 1]], eqnss2a /. x -> curverv2a[[ii, 1]]},
    {ii, 1, Length[curverv2a]}];
```

```
plot2a = ListPlot[{graphss2a, curverv2a},
  AxesLabel → {"time days", "radial velocity (km/s)"}]
```



b) Here is the maximum “plus” portion fit to cubic

```
curverv2b = Table[{dataS3[[ii, 1]], dataS3[[ii, 2]]}, {ii, 86, 110}];

plotdatab = ListPlot[curverv2b, AxesLabel → {"time days", "radial velocity (km/s)"}]
```




```
xfit2b = LinearModelFit[curverv2b, {x, x^2, x^3}, x, IncludeConstantBasis → False]
```

$$\text{FittedModel}\left[0.0267956\,x-3.83809\times10^{-6}\,x^2+1.44294\times10^{-10}\,x^3\right]$$

```
xfit2b["ParameterTable"]
```

	Estimate	Standard Error	t-Statistic	P-Value
x	0.0267956	0.00342275	7.82866	8.44029×10^{-8}
x^2	-3.83809×10^{-6}	5.95292×10^{-7}	−6.4474	1.73721×10^{-6}
x^3	1.44294×10^{-10}	2.58104×10^{-11}	5.59054	0.0000127732

```
√xfit2b["RSquared"]
```

0.999543

```
cfit2b = xfit2b["BestFitParameters"]
```

$$\left\{0.0267956,\ -3.83809\times10^{-6},\ 1.44294\times10^{-10}\right\}$$

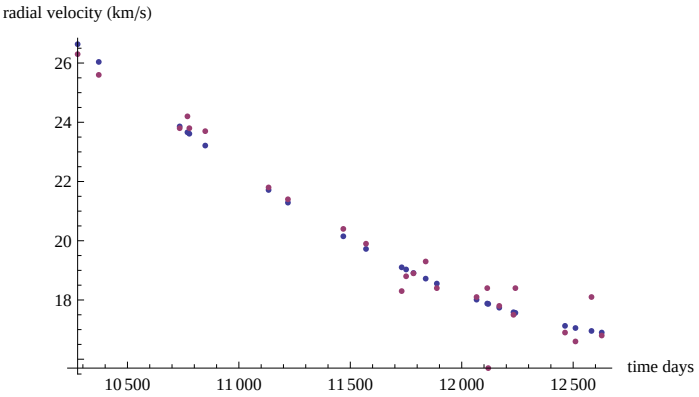
```
σfit2b = xfit2b["ParameterErrors"]
```

$$\{0.00342275,\ 5.95292 \times 10^{-7},\ 2.58104 \times 10^{-11}\}$$

```
eqnss2b = Normal[xfit2b]
```

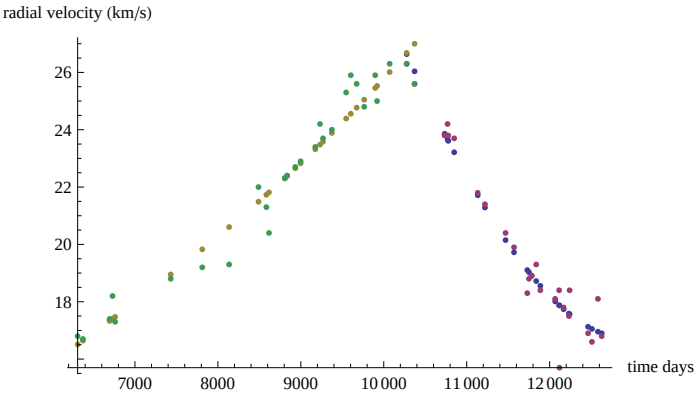
$$0.0267956\,x - 3.83809 \times 10^{-6}\,x^2 + 1.44294 \times 10^{-10}\,x^3$$

```
graphss2b = Table[{curverv2b[[ii, 1]], eqnss2b /. x -> curverv2b[[ii, 1]]},
    {ii, 1, Length[curverv2b]}];

plot2b = ListPlot[{graphss2b, curverv2b},
  AxesLabel → {"time days", "radial velocity (km/s)"}]
```



c)) Here is the maximum combined curve

```
plot2ab = ListPlot[{graphss2b, curverv2b, graphss2a, curverv2a},
  AxesLabel → {"time days", "radial velocity (km/s)"}]
```



It looks like the curves do intersect at a common point. We look for that point manually.

graphss2b

```
{{10276.4, 26.6363}, {10371.2, 26.0377}, {10734.3, 23.8598},
 {10769.2, 23.6612}, {10777.7, 23.6129}, {10849.1, 23.2135},
 {11133.2, 21.7144}, {11220.1, 21.2866}, {11468.6, 20.15}, {11570.5, 19.7236},
 {11730.5, 19.1031}, {11750.4, 19.0302}, {11783.4, 18.9116},
 {11838.2, 18.7209}, {11888.2, 18.5537}, {12066.5, 18.011}, {12114.5, 17.8799},
 {12119., 17.868}, {12168.4, 17.7401}, {12232.3, 17.5855}, {12240.8, 17.5658},
 {12463.5, 17.1264}, {12510.4, 17.0534}, {12582.2, 16.9548}, {12628.2, 16.9004}}
```

Length[graphss2a]

```
28
```

graphss2a

```
{{6308.82, 16.5096}, {6372.74, 16.6444}, {6694.81, 17.3296}, {6729.18, 17.4034},
 {6760.74, 17.4713}, {7431.31, 18.9507}, {7811.42, 19.8285}, {8135.26, 20.6048},
 {8489.46, 21.4886}, {8584.18, 21.7317}, {8616.18, 21.8145}, {8807.46, 22.317},
 {8835.36, 22.3914}, {8933.29, 22.6549}, {8997.14, 22.8285}, {9174.46, 23.3193},
 {9231.35, 23.4795}, {9267.32, 23.5814}, {9374.18, 23.8874}, {9546.43, 24.3912},
 {9602.35, 24.5575}, {9673.18, 24.7703}, {9764.11, 25.047}, {9896.46, 25.4566},
 {9919.53, 25.5288}, {10071.2, 26.0105}, {10276.4, 26.6812}, {10371.2, 26.9985}}
```

The intersection is at x=10276.4, graphss2b[[1]] and graphss2a[27]

Both maximum and minimum curve fits are “peaky” with the maximum more so. In most cases such behavior would be unphysical, so we formulate a least squares transition function for each curve. With curves generated at equispaced x values, *Mathematica* provides interpolations which are automatic spline fits by means of the function Interpolation[]. But because we already have unequally spaced points, we would rather formulate transition functions that can be determined from unequal intervals. The main point of a transition function is to match the slopes of two curves that have different slopes coming into a cusp point and obtain a continuous curve that comes to a maximum or minimum. Often cubic equations are used as splines but in our case we have had to go to 7th order plus a constant. Ideally we should fit the new function with a centered variable (t - epoch) were the epoch time is that that where the maximum (or minimum) occurs. But since the we really don’t know the epoch accurately ahead of time, we will use non-linear least squares fitting to find it. The points we need fit are to be found in the four plotting arrays (lists), graphss1 (post-minimum), graphssa (pre-minimum) and graphss2a (pre-maximum), graphss2b (post-maximum). Combining most of the points from each list should give enough data to do the regressions properly.

Creating Smooth Velocity Curves That Transition Smoothly Through Maximum and Minimum

numminR = Length[graphss1]

```
41
```

```
numminL = Length[graphssa]
```

```
26
```

```
nummaxL = Length[graphss2a]
```

```
28
```

```
nummaxR = Length[graphss2b]
```

```
25
```

Here we select the number of samples from each file based on the numbers above.

```
numca = 25; numc1 = 40; numc2a = 26; numc2b = 23;
```

```
minimumf = Table[0, {ii, 1, (numca + numc1)}];
```

```
maximumf = Table[0, {ii, 1, (numc2a + numc2b)}];
```

```
nummin = Length[minimumf]
```

```
65
```

```
nummax = Length[maximumf]
```

```
49
```

The pre minimum values are inserted into the first part of the new array.

```
jj = numminL - numca;
```

```
Do[jj = jj + 1;
 minimumf[[ii]] = {graphssa[[jj, 1]], graphssa[[jj, 2]]};, {ii, 1, numca}]
```

The post minimum values are inserted into the 2nd part of the new array.

```
jj = 0;
```

```
Do[jj = jj + 1; minimumf[[ii]] = {graphss1[[jj, 1]], graphss1[[jj, 2]]};,
 {ii, numca + 1, nummin}]
```

The pre maximum values are inserted into the first part of the new array. Unlike the minimum case where the extreme points were trimmed off the beginnings and ends of the original files. Here we want to trim off mainly the points around the peak since trial and error fitting has shown that to be causing a lack of fit of the maximum curve.

```
jj = 0;
```

```
Do[jj = jj + 1;
 maximumf[[ii]] = {graphss2a[[jj, 1]], graphss2a[[jj, 2]]};, {ii, 1, numc2a}]
```

The first half post maximum values are inserted into the 2nd part of the new array.

```
jj = 1;
```

```
Do[jj = jj + 1; maximumf[[ii]] = {graphss2b[[jj, 1]], graphss2b[[jj, 2]]};,
 {ii, numc2a + 1, nummax}]
```

The minimum part of the velocity curve

```
mincurve1 = NonlinearModelFit[minimumf,
  an + bn (t - tmin) + cn (t - tmin)^2 + dn (t - tmin)^3 + en (t - tmin)^4 + fn (t - tmin)^5 +
   gn (t - tmin)^6 + hn (t - tmin)^7, {an, bn, cn, dn, en, fn, gn, hn, tmin}, t]
```

NonlinearModelFit::sszero :
The step size in the search has become less than the tolerance prescribed by the PrecisionGoal option, but the gradient is larger than the tolerance specified by the AccuracyGoal option. There is a possibility that the method has stalled at a point that is not a local minimum. ≫

FittedModel[−34237.+≪9≫+1.63301×10^−24 (5334.12+t)^7]

```
mincurve = Normal[mincurve1]
```

$$-34237.+22.3399\,(5334.12+t)-0.00616267\,(5334.12+t)^2+9.33049\times10^{-7}\,(5334.12+t)^3-8.38068\times10^{-11}\,(5334.12+t)^4+4.46908\times10^{-15}\,(5334.12+t)^5-1.31097\times10^{-19}\,(5334.12+t)^6+1.63301\times10^{-24}\,(5334.12+t)^7$$

The fact that the initial constant comes out to be the real minimum velocity while the epoch tmin comes out to be the time value expected for the minimum indicates that this 7th order polynomial is an excellent fit.

```
√mincurve1["RSquared"]
```

0.999967

```
minparams1 = mincurve1["BestFitParameters"]
```

$$\{an\to-34237.,\ bn\to22.3399,\ cn\to-0.00616267,\ dn\to9.33049\times10^{-7},\ en\to-8.38068\times10^{-11},\ fn\to4.46908\times10^{-15},\ gn\to-1.31097\times10^{-19},\ hn\to1.63301\times10^{-24},\ tmin\to-5334.12\}$$

```
minparams = {minparams1[[1, 2]], minparams1[[2, 2]], minparams1[[3, 2]],
    minparams1[[4, 2]], minparams1[[5, 2]], minparams1[[6, 2]],
    minparams1[[7, 2]], minparams1[[8, 2]], minparams1[[9, 2]]};
```

```
minerrors = mincurve1["ParameterErrors"]
```

$$\{9.47387\times10^{-27},\ 5.95141\times10^{-32},\ 3.55873\times10^{-32},\ 2.42883\times10^{-28},\ 1.40237\times10^{-24},\ 5.47644\times10^{-21},\ 8.8403\times10^{-25},\ 3.5173\times10^{-29},\ 1.52336\times10^{-42}\}$$

As expressed by Meyer (1975), power series expansions fit by least squares are notorious for producing coefficients that horridly correlated so that using just the standard errors in a error propagation based on a random model can mislead considerably. Examination of the covariance matrix shows just that tendency here.

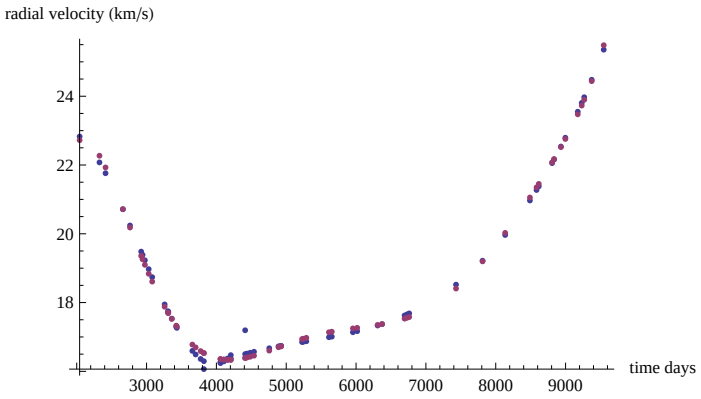
```
mincovariance = mincurve1["CovarianceMatrix"] // MatrixForm
```

$$\begin{pmatrix}
8.97542\times10^{-53} & -5.63829\times10^{-58} & -3.37149\times10^{-58} & -2.30104\times10^{-54} & -1.32858\times10^{-50} & -5.18\\
-5.63829\times10^{-58} & 3.54193\times10^{-63} & 2.11795\times10^{-63} & 1.4455\times10^{-59} & 8.34606\times10^{-56} & 3.25\\
-3.37149\times10^{-58} & 2.11795\times10^{-63} & 1.26645\times10^{-63} & 8.64354\times10^{-60} & 4.99064\times10^{-56} & 1.94\\
-2.30104\times10^{-54} & 1.4455\times10^{-59} & 8.64354\times10^{-60} & 5.89921\times10^{-56} & 3.40611\times10^{-52} & 1.33\\
-1.32858\times10^{-50} & 8.34606\times10^{-56} & 4.99064\times10^{-56} & 3.40611\times10^{-52} & 1.96663\times10^{-48} & 7.67\\
-5.18831\times10^{-47} & 3.25926\times10^{-52} & 1.94892\times10^{-52} & 1.33013\times10^{-48} & 7.67998\times10^{-45} & 2.99\\
8.34565\times10^{-51} & -5.24268\times10^{-56} & -3.13493\times10^{-56} & -2.13959\times10^{-52} & -1.23536\times10^{-48} & -4.82\\
-3.29485\times10^{-55} & 2.0698\times10^{-60} & 1.23767\times10^{-60} & 8.44706\times10^{-57} & 4.87719\times10^{-53} & 1.90\\
-1.44321\times10^{-68} & 9.06613\times10^{-74} & 5.42122\times10^{-74} & 3.69998\times10^{-70} & 2.1363\times10^{-66} & 8.34
\end{pmatrix}$$

```
mincurve1["ParameterTable"]
```

	Estimate	Standard Error	t-Statistic	P-Value
an	−34237.	9.47387×10^{-27}	-3.61383×10^{30}	$5.357546495727143\times10^{-1664}$
bn	22.3399	5.95141×10^{-32}	3.75372×10^{32}	$6.387315903063993\times10^{-1777}$
cn	−0.00616267	3.55873×10^{-32}	-1.73171×10^{29}	$4.174534455555614\times10^{-1590}$
dn	9.33049×10^{-7}	2.42883×10^{-28}	3.84156×10^{21}	$1.748916013089006\times10^{-1161}$
en	-8.38068×10^{-11}	1.40237×10^{-24}	-5.9761×10^{13}	$3.131580966350929\times10^{-724}$
fn	4.46908×10^{-15}	5.47644×10^{-21}	816055.	8.29629×10^{-284}
gn	-1.31097×10^{-19}	8.8403×10^{-25}	−148295.	2.46698×10^{-242}
hn	1.63301×10^{-24}	3.5173×10^{-29}	46428.	4.32024×10^{-214}
tmin	−5334.12	1.52336×10^{-42}	-3.50155×10^{45}	$3.137475671353368\times10^{-2503}$

```
mincurve2 = Table[
    {minimumf[[ii, 1]], mincurve /. t → minimumf[[ii, 1]]}, {ii, 1, Length[minimumf]}];

ListPlot[{minimumf, mincurve2}, AxesLabel → {"time days", "radial velocity (km/s)"}]
```



The maximum part of the velocity curve

The maximum of the rv curve is a bit sharper than that of minimum and so it was a somehwhat harder to reproduce it satisfactorily. Comments on how to construct the data file leaving peak points off are given above. First we redo the standard approach that worked so well on the minimum part of the radial velocity curve. In that polynomial model, the order was seven but for the maximum fit it did not yield the expected am and tmax values. In addition, the signs of the coefficients alternate, not a good sign that this is the correct fit. Finally *Mathematica* issues a warning about convergence to a local minimum of the sum of

the squares instead of the global minimum desired. That same failing of proper convergence warning was issed by Meyer in his discussion of non-linear least-squares methodology so it must be taken seriously here also.

```
maxcurve1 = NonlinearModelFit[maximumf,
  am + bm (t - tmax) + cm (t - tmax)^2 + dm (t - tmax)^3 + em (t - tmax)^4 + fm (t - tmax)^5 +
   gm (t - tmax)^6 + hm (t - tmax)^7, {am, bm, cm, dm, em, fm, gm, hm, tmax}, t]
```

NonlinearModelFit::sszero :
The step size in the search has become less than the tolerance prescribed by the PrecisionGoal option, but the gradient is larger than the tolerance specified by the AccuracyGoal option. There is a possibility that the method has stalled at a point that is not a local minimum. ≫

```
FittedModel[ ≪1≫ ]
```

```
√maxcurve1["RSquared"]
```

```
0.999947
```

```
maxcurve = Normal[maxcurve1]
```

$$-2.68747\times10^{8}-59\,631.4\,(-41\,183.2+t)-5.66458\,(-41\,183.2+t)^{2}-0.000298625\,(-41\,183.2+t)^{3}-9.43581\times10^{-9}\,(-41\,183.2+t)^{4}-1.78701\times10^{-13}\,(-41\,183.2+t)^{5}-1.87825\times10^{-18}\,(-41\,183.2+t)^{6}-8.45191\times10^{-24}\,(-41\,183.2+t)^{7}$$

```
maxparams1 = maxcurve1["BestFitParameters"]
```

$$\{am\to-2.68747\times10^{8},\ bm\to-59\,631.4,\ cm\to-5.66458,\ dm\to-0.000298625,\ em\to-9.43581\times10^{-9},\ fm\to-1.78701\times10^{-13},\ gm\to-1.87825\times10^{-18},\ hm\to-8.45191\times10^{-24},\ tmax\to41\,183.2\}$$

```
maxparams = {maxparams1[[1, 2]], maxparams1[[2, 2]], maxparams1[[3, 2]],
   maxparams1[[4, 2]], maxparams1[[5, 2]], maxparams1[[6, 2]],
   maxparams1[[7, 2]], maxparams1[[8, 2]], maxparams1[[9, 2]]}
```

$$\{-2.68747\times10^{8},\ -59\,631.4,\ -5.66458,\ -0.000298625,\ -9.43581\times10^{-9},\ -1.78701\times10^{-13},\ -1.87825\times10^{-18},\ -8.45191\times10^{-24},\ 41\,183.2\}$$

```
maxerrors = maxcurve1["ParameterErrors"]
```

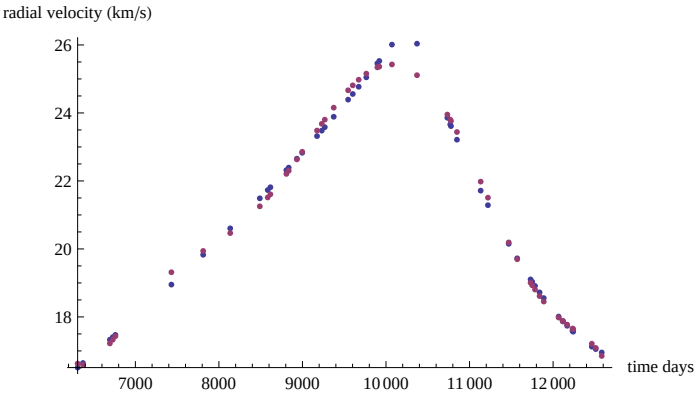
$$\{9.84994\times10^{-28},\ 2.8698\times10^{-33},\ 1.0309\times10^{-34},\ 1.95903\times10^{-30},\ 3.10672\times10^{-26},\ 3.28923\times10^{-22},\ 2.05741\times10^{-26},\ 3.20918\times10^{-31},\ 5.82092\times10^{-45}\}$$

```
maxcurve1["ParameterTable"]
```

	Estimate	Standard Error	t-Statistic	P-Value
am	-2.68747×10^{8}	9.84994×10^{-28}	-2.72841×10^{35}	$5.046549556495583\times10^{-1387}$
bm	−59 631.4	2.8698×10^{-33}	-2.0779×10^{37}	$2.719066400059621\times10^{-1462}$
cm	−5.66458	1.0309×10^{-34}	-5.4948×10^{34}	$3.477840580841610\times10^{-1359}$
dm	−0.000298625	1.95903×10^{-30}	-1.52435×10^{26}	$6.545737547048920\times10^{-1017}$
em	-9.43581×10^{-9}	3.10672×10^{-26}	-3.03723×10^{17}	$6.923152552084629\times10^{-669}$
fm	-1.78701×10^{-13}	3.28923×10^{-22}	-5.43293×10^{8}	$5.470725699408782\times10^{-319}$
gm	-1.87825×10^{-18}	2.05741×10^{-26}	-9.12921×10^{7}	5.27307×10^{-288}
hm	-8.45191×10^{-24}	3.20918×10^{-31}	-2.63367×10^{7}	2.07478×10^{-266}
tmax	41 183.2	5.82092×10^{-45}	7.07503×10^{48}	$1.4133990054423 66\times10^{-1923}$

The non-convergence most likely results because the procedure was unable to reach the peak points properly at that order as can be seen below

```
maxcurve2 = Table[
    {maximumf[[ii, 1]], maxcurve /. t → maximumf[[ii, 1]]}, {ii, 1, Length[maximumf]}];

ListPlot[{maximumf, maxcurve2}, AxesLabel → {"time days", "radial velocity (km/s)"}]
```



Several attempts were made to increase the order of the fit but because the pre and post maximum curves are so well reproduced these did not produce the desired results. Next we try specifying the desired time of peak passage to force the non-linear method to pass the local minimum. First we sort the original data using a *Mathematica* “pure” function (similar to that given in the online Help under Sort[]) that produces a list with the maximum velocity entry first. (For more information about “pure” functions in *Mathematica* put Pure Functions in the help search field and a number of links to on-line references will appear.) You can now see the maximum points and select either their times or velocities or some combination of the two.

```
smax = Sort[maximumf, #1[[2]] > #2[[2]] &]
```

```
{{10 371.2, 26.0377}, {10 071.2, 26.0105}, {9919.53, 25.5288},
 {9896.46, 25.4566}, {9764.11, 25.047}, {9673.18, 24.7703}, {9602.35, 24.5575},
 {9546.43, 24.3912}, {9374.18, 23.8874}, {10 734.3, 23.8598}, {10 769.2, 23.6612},
 {10 777.7, 23.6129}, {9267.32, 23.5814}, {9231.35, 23.4795}, {9174.46, 23.3193},
 {10 849.1, 23.2135}, {8997.14, 22.8285}, {8933.29, 22.6549}, {8835.36, 22.3914},
 {8807.46, 22.317}, {8616.18, 21.8145}, {8584.18, 21.7317}, {11 133.2, 21.7144},
 {8489.46, 21.4886}, {11 220.1, 21.2866}, {8135.26, 20.6048}, {11 468.6, 20.15},
 {7811.42, 19.8285}, {11 570.5, 19.7236}, {11 730.5, 19.1031},
 {11 750.4, 19.0302}, {7431.31, 18.9507}, {11 783.4, 18.9116}, {11 838.2, 18.7209},
 {11 888.2, 18.5537}, {12 066.5, 18.011}, {12 114.5, 17.8799}, {12 119., 17.868},
 {12 168.4, 17.7401}, {12 232.3, 17.5855}, {12 240.8, 17.5658},
 {6760.74, 17.4713}, {6729.18, 17.4034}, {6694.81, 17.3296}, {12 463.5, 17.1264},
 {12 510.4, 17.0534}, {12 582.2, 16.9548}, {6372.74, 16.6444}, {6308.82, 16.5096}}
```

Now you have a choice about where to put the maximum. In the previous fit, the maximum of the function comes out to be nearest the second point so we chose it. The problem with that earlier maximum curve fit is that the derived coefficients and their errors were incorrect because the sum of the squares was a local not global minimum. But now we can chose a time closer to the actual maximum for the epoch. If we were purists we would run this with this epoch and then run it again with the actual functional maximum. But doing this numerically achieves little in terms of smaller errors so we run only once. You can try using the functional maximum if you wish to see that numerically it is not necessary to rerun. The difference between the two is 1 in the fourth significant figure.

```
tmax1 = smax[[2, 1]]
```

```
10 071.2
```

```
Δtmax1 = 0.5;
```

Since we now specify the epoch of the expansion we can return to the LinearModelFit[] routine to analyze the maximum curve once again.

```
yfunction = Collect[Expand[am1 + bm1 (t - tmax1) + cm1 (t - tmax1)^2 + dm1 (t - tmax1)^3 +
    em1 (t - tmax1)^4 + fm1 (t - tmax1)^5 + gm1 (t - tmax1)^6 + hm1 (t - tmax1)^7], t]
```

$$\begin{aligned}
&\text{am1} - 10\,071.2\ \text{bm1} + 1.01429\times 10^{8}\ \text{cm1} - 1.02151\times 10^{12}\ \text{dm1} + \\
&1.02878\times 10^{16}\ \text{em1} - 1.0361\times 10^{20}\ \text{fm1} + 1.04348\times 10^{24}\ \text{gm1} - 1.05091\times 10^{28}\ \text{hm1} + \\
&\left(\text{bm1} - 20\,142.4\ \text{cm1} + 3.04286\times 10^{8}\ \text{dm1} - 4.08603\times 10^{12}\ \text{em1} + \right. \\
&\quad \left. 5.1439\times 10^{16}\ \text{fm1} - 6.21662\times 10^{20}\ \text{gm1} + 7.30435\times 10^{24}\ \text{hm1}\right)\ t + \\
&\left(\text{cm1} - 30\,213.6\ \text{dm1} + 6.08573\times 10^{8}\ \text{em1} - 1.02151\times 10^{13}\ \text{fm1} + \right. \\
&\quad \left. 1.54317\times 10^{17}\ \text{gm1} - 2.17582\times 10^{21}\ \text{hm1}\right)\ t^2 + \\
&\left(\text{dm1} - 40\,284.7\ \text{em1} + 1.01429\times 10^{9}\ \text{fm1} - 2.04302\times 10^{13}\ \text{gm1} + 3.60073\times 10^{17}\ \text{hm1}\right)\ t^3 + \\
&\left(\text{em1} - 50\,355.9\ \text{fm1} + 1.52143\times 10^{9}\ \text{gm1} - 3.57528\times 10^{13}\ \text{hm1}\right)\ t^4 + \\
&\left(\text{fm1} - 60\,427.1\ \text{gm1} + 2.13\times 10^{9}\ \text{hm1}\right)\ t^5 + \left(\text{gm1} - 70\,498.3\ \text{hm1}\right)\ t^6 + \text{hm1}\ t^7
\end{aligned}$$

We now have the coefficient expressions in the original expansion to get back into the assumed form.

```
maxcurve3 =
 NonlinearModelFit[maximumf, yfunction, {am1, bm1, cm1, dm1, em1, fm1, gm1, hm1}, t]
```

FittedModel[43 992.2 − 35.0002 t + 0.0118019 t^2 − ≪23≫ ≪1≫ + ≪1≫ − ≪23≫ t^5 + 5.58285 × 10^−19 t^6 − 8.45191 × 10^−24 t^7]

```
√maxcurve3["RSquared"]
```

0.999947

Once again we examine the covariance matrix for the least squares fit to see how badly the various values are correlated with each other and as for the analysis of the minimum curve the coefficients are highly correlated.

```
maxcovariance = maxcurve3["CovarianceMatrix"] // MatrixForm
```

1.03978 × 10^−37	−1.91224 × 10^−42	−1.34936 × 10^−39	−4.24425 × 10^−37	−3.26738 × 10^−33	−4.98
−1.91224 × 10^−42	3.51678 × 10^−47	2.48159 × 10^−44	7.80554 × 10^−42	6.009 × 10^−38	9.15
−1.34936 × 10^−39	2.48159 × 10^−44	1.75111 × 10^−41	5.50792 × 10^−39	4.2402 × 10^−35	6.46
−4.24425 × 10^−37	7.80554 × 10^−42	5.50792 × 10^−39	1.73245 × 10^−36	1.3337 × 10^−32	2.03
−3.26738 × 10^−33	6.009 × 10^−38	4.2402 × 10^−35	1.3337 × 10^−32	1.02673 × 10^−28	1.56
−4.98058 × 10^−37	9.15972 × 10^−42	6.46348 × 10^−39	2.03301 × 10^−36	1.56509 × 10^−32	4.23
5.35091 × 10^−40	−9.8408 × 10^−45	−6.94407 × 10^−42	−2.18418 × 10^−39	−1.68146 × 10^−35	−2.6
1.16368 × 10^−43	−2.14011 × 10^−48	−1.51015 × 10^−45	−4.74999 × 10^−43	−3.65672 × 10^−39	−7.34

```
maxcurve4 = Normal[maxcurve3]
```

43 992.2 − 35.0002 t + 0.0118019 t^2 − 2.18529 × 10^−6 t^3 +
 2.39942 × 10^−10 t^4 − 1.56205 × 10^−14 t^5 + 5.58285 × 10^−19 t^6 − 8.45191 × 10^−24 t^7

```
maxparams4 = maxcurve3["BestFitParameters"]
```

{am1 → 25.4266, bm1 → −0.000057498, cm1 → −3.24258 × 10^−6, dm1 → −4.48887 × 10^−10,
 em1 → 5.69323 × 10^−13, fm1 → 1.1243 × 10^−16, gm1 → −3.75603 × 10^−20, hm1 → −8.45191 × 10^−24}

```
maxparams5 = {maxparams4[[1, 2]], maxparams4[[2, 2]],
   maxparams4[[3, 2]], maxparams4[[4, 2]], maxparams4[[5, 2]],
   maxparams4[[6, 2]], maxparams4[[7, 2]], maxparams4[[8, 2]]}
```

{25.4266, −0.000057498, −3.24258 × 10^−6, −4.48887 × 10^−10,
 5.69323 × 10^−13, 1.1243 × 10^−16, −3.75603 × 10^−20, −8.45191 × 10^−24}

```
maxerrors5 = maxcurve3["ParameterErrors"]
```

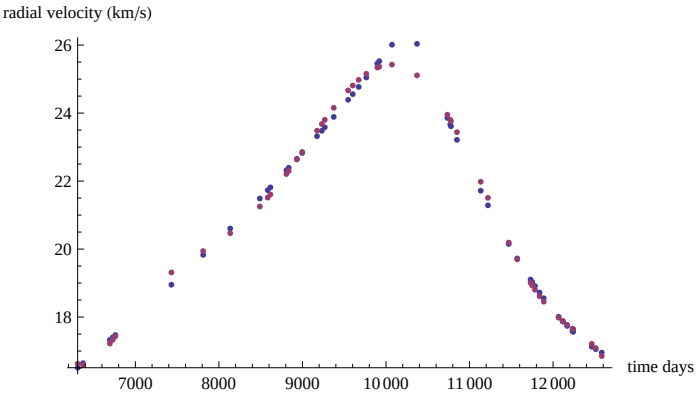
{3.22456 × 10^−19, 5.93025 × 10^−24, 4.18463 × 10^−21, 1.31623 × 10^−18,
 1.01328 × 10^−14, 2.0577 × 10^−18, 1.69114 × 10^−21, 3.93211 × 10^−25}

```
maxcurve3["ParameterTable"]
```

	Estimate	Standard Error	t-Statistic	P-Value
am1	25.4266	3.22456×10^{-19}	7.88529×10^{19}	$2.4290875842145 56\times10^{-784}$
bm1	−0.000057498	5.93025×10^{-24}	-9.69571×10^{18}	$5.072310266715133\times10^{-747}$
cm1	-3.24258×10^{-6}	4.18463×10^{-21}	-7.74879×10^{14}	$4.970314516241459\times10^{-579}$
dm1	-4.48887×10^{-10}	1.31623×10^{-18}	-3.41041×10^{8}	$2.041183143498667\times10^{-318}$
em1	5.69323×10^{-13}	1.01328×10^{-14}	56.1862	2.03213×10^{-40}
fm1	1.1243×10^{-16}	2.0577×10^{-18}	54.6386	6.29318×10^{-40}
gm1	-3.75603×10^{-20}	1.69114×10^{-21}	−22.21	1.78698×10^{-24}
hm1	-8.45191×10^{-24}	3.93211×10^{-25}	−21.4946	6.16687×10^{-24}

Our little trick worked. As can be seen our function appears exactly the same, but the constant parameter is what would be expected for the maximum value of the function. We indeed have arrived at the global minimum with coefficients that will have minimum variance. Now we are ready to move on with the solutions.

```
maxcurve5 = Table[{maximumf[[ii, 1]], maxcurve4 /. t → maximumf[[ii, 1]]},
    {ii, 1, Length[maximumf]}];

ListPlot[{maximumf, maxcurve5}, AxesLabel → {"time days", "radial velocity (km/s)"}]
```



Deriving the Orbital Parameters for HD 108613

Now we are ready to integrate the maximum and minimum curves as needed to derive the orbital elements. First we need to verify the maximum and minimum times since these are where the half integrals are to be taken from when it comes time to calculate the orbital elements. Of course there are multiple roots from Solve[]=0 from which the correct root can only be selected from graphs by inspection and then selected using the proper *Mathematica* component notation. In the case of the minimum, the tmin0 value (from the least-squares fit) should be reasonably close although not likely to be exactly the time obtained by Solve[]=0.

First we turn to the determination of the time of minimum.

```
min1 = Solve[∂t mincurve == 0, t]
```

```
{{t → 2085.69}, {t → 4186.23}, {t → 6071.83 - 538.378 i},
 {t → 6071.83 + 538.378 i}, {t → 9195.28 - 1041.39 i}, {t → 9195.28 + 1041.39 i}}
```

There are two real solutions. The smaller can be rejected by consulting the graph so the next one is chosen.

```
tmin0 = min1[[2, 1, 2]]
```

```
4186.23
```

The predicted radial velocity at this time is

```
lims1 = mincurve /. t → tmin0
```

```
16.3353
```

Now we turn to the maximum determination

```
max1 = Solve[∂t maxcurve4 == 0, t]
```

```
{{t → 6372.08}, {t → 7765.5 - 616.775 i}, {t → 7765.5 + 616.775 i},
 {t → 10 062.3}, {t → 12 326.3 - 326.461 i}, {t → 12 326.3 + 326.461 i}}
```

The time of maximum given by the function and verified from the graph is

```
tmax0 = max1[[4, 1, 2]]
```

```
10 062.3
```

The predicted radial velocity at this time is

```
lims2 = maxcurve4 /. t → tmax0
```

```
25.4269
```

Next we need to find the two time intersection points for a given rv level and then substitute as limits in the integral. This is to be done by trial and error by adjusting the rv0 value computing the two integrals comparing their values and readjusting the rV0 value until there is near equality. In the present case, we get 19.746 km/s for the γ velocity. Griffin gets 19.8 km/s. Once we were satified with the actual rv0 value then we calculated the times at which this is achieved for the two curves using Solve[]=0 to give times

```
rv0 = 19.746;
```

We will need an error estimate of this. Although this value is quoted to three decimals, the adjustment process suggests the following is reasonable.

```
Δrv0 = 0.05;
```

```
min2 = Solve[ mincurve == rv0, t]
```

```
{{t → 1630.62}, {t → 2846.57}, {t → 5490.94 - 1767.28 i}, {t → 5490.94 + 1767.28 i},
 {t → 8030.71}, {t → 9725.36 - 1483.77 i}, {t → 9725.36 + 1483.77 i}}
```

```
max2 = Solve[ maxcurve4 == rv0, t]
```

```
{{t → 5917.56}, {t → 7682.97}, {t → 7773.07 - 1024.06 i}, {t → 7773.07 + 1024.06 i},
 {t → 11 559.9}, {t → 12 673.8 - 644.873 i}, {t → 12 673.8 + 644.873 i}}
```

Now we take the definite integrals as required. Be sure to manually change the indices of the limits to correspond to the real roots of the equations if that changes during your iteration ! Also note that the lower limit of the maximum curve should properly correspond to the value of the upper limit of the minimum integral. and so we have made it that way. Also note that we have made no special sophisticated attempt to make the two curve fits blend smoothly in the vicinity of rv0.

Here is the integral of the (rv0-minimum curve)

```
tuL = min2[[5, 1, 2]]
```

```
8030.71
```

```
tlL = min2[[2, 1, 2]]
```

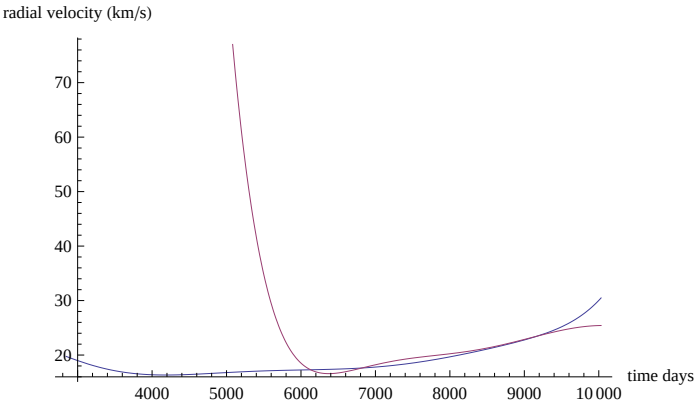
```
2846.57
```

$$\text{integ1} = \int_{\text{tlL}}^{\text{tuL}} (\text{rv0} - \text{mincurve})\, dt$$

```
11872.2
```

Plotting the two functions mincurve and maxcurve shows that the two functions have a range of “osculation” where they stay close together from about 7000 to at least 10000. Thus for the other integral we can use the upper limit of integ1 as the lower limit of integ2 calculated below without dire consequence.

```
plota = Plot[{mincurve /. t -> t2, maxcurve /. t → t2},
  {t2, tlL, tuL + 2000}, AxesLabel → {"time days", "radial velocity (km/s)"}]
```



Here is the integral of the (maximum curve-rv0)

```
tuU = max2[[5, 1, 2]];
```

```
tlU = min2[[5, 1, 2]];
```

$$\text{integ2} = \int_{\text{tlU}}^{\text{tuU}} (\text{maxcurve4} - \text{rv0})\, dt$$

```
11875.8
```

Now that a rv0 value has been established we can proceed, but notice we have not established any of the errors of the process so far. We now take a detour to discuss the

propagation of errors.

A Digression Concerning the Propagation of Errors

One of the virtues of using *Mathematica* is not only that it can calculate the errors for you, but that it can compare results from all the three error propagation categories mentioned above. Finding the propagated random errors in the case of integrals of a polynomial is not trivial and often turns out (to the embarrassment of the researcher) many times larger than the integral value itself. This is not just limited to the case of the integral of a polynomial. That is why the results of a "proper" (independent errors) error analysis is rarely quoted when the observational errors are large. In a high signal to noise case where equal weights are assumed, error independence turns out to be a reasonable assumption. In this spectroscopic binary case, we find that assumption to be far from justified. The symptom of this is that the propagated errors are very large, sometimes 2-3 times larger than the numbers themselves. In such cases, the true errors are most likely correlated with unknown linear (or even non-linear) correlation coefficients. When that happens, assuming the coefficients to be orthogonal produces erroneous results and simply means that the true errors remain largely unknown. In spite of the caveats, we provide the errors under the three situations mentioned near the first of this notebook for much of the spectroscopic binary analysis primarily because *Mathematica* makes the arduous calculations relatively easy. But to do the proper error analysis requires that the least squares "predicted radial velocities" be calculated explicitly from the series coefficients. Here we give the minimum curve and maximum curve expansions as needed.

Here are the power series expansions of each curve. These produce the same results as mincurv1/.t -> te and maxcurve4/.t -> te used above. In the case of the minimum curve series expansion the "targument" and its error are both available in the minparams and minerror vectors as the last entry. In the maximum curve case, the number was selected by "eye" and hence no error is assigned. We can assign this to have the same error as does tmin0. Hence

```
Δtph1 = minerrors[[9]];
```

Another issue is what error to assign to individual observation times. A reasonable estimate might be conservatively estimated at 0.1 day. So in that case we take

```
Δte1 = 0.1;
```

```
minseries[te_] := Module[{}, n0 = Length[minparams];
  (Sum[minparams[[n + 1]] (te - tph)^n, {n, 0, n0 - 2}]) /. tph → minparams[[n0]]]
```

```
maxseries[te_] := Module[{}, n0 = Length[maxparams5];
  (Sum[maxparams5[[n + 1]] (te - tph)^n, {n, 0, n0 - 1}]) /. tph → tmax0]
```

Here are the integrals of each curve from zero

```
intminseries[te_] := Module[{}, n0 = Length[minparams];
  (rv0 (te - tph) - Sum[minparams[[n + 1]] (te - tph)^(n+1) / (n + 1), {n, 0, n0-2}]) /. tph -> minparams[[n0]]]

intmaxseries[te_] := Module[{}, n0 = Length[maxparams5];
  (Sum[maxparams5[[n + 1]] (te - tph)^(n+1) / (n + 1), {n, 0, n0-2}] - rv0 (te - tph)) /. tph → tmax1]
```

We now illustrate the three main error analysis methods for the integrals needed for spectroscopic orbits because the star being analyzed (if the results are taken at face value) appears to be very unusual. But since it turns out that the parameters in question appear to have large errors, the results may actually be suspect and we need to decide which is the case.

a) Random Errors

Here is a function that calculates the random error in the definite integral of a fitted non-linear polynomial. Here n0 = the number of coefficents including the constant term, tph = the epoch time from the fit (this is always the last coefficient in the fits), Δtph = the error of the epoch time from the fit, t1 = the time for which the error is desired, Δt1 the error on the time, c0 = a list giving all power law coefficients in the polynomial starting with the constant term first and going to progressively higher powers, Δc0 = the errors of the coefficients in the same order as the previous list, and Δrv0 = the error of the velocity determination.

This function has been left with undermined parameters and coefficients so that it will generate errors for any integral of polynomial. To be useful we have to put this into a Module[] where the unspecified parameters can be specified after the function has been generated.

Before doing the integral error analysis, we first do the “prediction” errors for the max and min series according to standard propagation of random errors.

```
erminseries[te_] := Module[{}, n0 = Length[minerrors]; Clear[tph, Δte, Δtph];
  Sqrt[((Sum[n minparams[[n + 1]] (te - tph)^(n-1) Sqrt[Δte1^2 + minerrors[[n0]]^2], {n, 0, n0-2}]) /.
      {tph -> minparams[[n0]], Δte → Δte1, Δtph → minerrors[[n0]]})^2 +
    ((Sum[(te - tph)^n minerrors[[n + 1]], {n, 0, n0-2}]) /. {tph -> minparams[[n0]],
      Δte → Δte1, Δtph → minerrors[[n0]]})^2]]
```

```
ermaxseries[te_] := Module[{}, n0 = Length[maxerrors5]; Clear[tph, Δte, Δtph];
 Sqrt[(((Sum[n maxparams5[[n + 1]] (te - tph)^(n - 1) Sqrt[Δte^2 + Δtph1^2], {n, 0, n0 - 1}]) /.
        {tph → tmax1, Δte → Δte1, Δtph → Δtph1})^2 +
    ((Sum[(te - tph)^n maxerrors5[[n + 1]], {n, 0, n0 - 1}]) /. {tph → tmax1, Δte → Δte1, Δtph → Δtph1})^2)]]
```

Using the analogous pattern, we do the “prediction” errors for the max and min integrals according to standard propagation of random errors.

```
erintminseries[te_] := Module[{}, n0 = Length[minerrors]; Clear[tph, Δte, Δtph];
 Sqrt[(((Sum[minparams[[n + 1]] (te - tph)^n Sqrt[Δte1^2 + minerrors[[n0]]^2], {n, 0, n0 - 2}]) /.
        {tph -> minparams[[n0]], Δte → Δte1, Δtph → minerrors[[n0]]})^2 +
    ((Sum[((te - tph)^(n + 1))/(n + 1) minerrors[[n + 1]], {n, 0, n0 - 2}]) /. {tph -> minparams[[n0]],
        Δte → Δte1, Δtph → minerrors[[n0]]})^2)]]

erintmaxseries[te_] := Module[{}, n0 = Length[maxerrors5]; Clear[tph, Δte, Δtph];
 Sqrt[(((Sum[maxparams5[[n + 1]] (te - tph)^n Sqrt[Δte1^2 + Δtph1^2], {n, 0, n0 - 1}]) /. {tph → tmax1,
        Δte → Δte1, Δtph → Δtph1})^2 + ((Sum[((te - tph)^(n + 1))/(n + 1) maxerrors5[[n + 1]], {n, 0, n0 - 1}]) /.
        {tph → tmax1, Δte → Δte1, Δtph → Δtph1})^2)]]
```

Here is the total error for the evaluation of the power series for the minimum

```
errmin[tU_, ΔtU_] := Module[{}, n1 = Length[minparams];
  answ1 = Sqrt[erminseries[tU]^2 + ΔtU^2 + minerrors[[n1]]^2]]
```

Here is the total error for the evaluation of the power series for the maximum

```
errmax[tU_, ΔtU_] :=
  Module[{}, n2 = Length[maxparams5]; answ3 = √(ermaxseries[tU]^2 + ΔtU^2 + Δtph1^2)];
```

Here is the total error for the integral representing the minimum curve integral

```
errintmin[tU_, ΔtU_, tL_, ΔtL_] := Module[{}, n1 = Length[minparams];
  answ1 = √(erintminseries[tU]^2 + ΔtU^2 + minerrors[[n1]]^2);
  answ2 = √(erintminseries[tL]^2 + ΔtL^2 + minerrors[[n1]]^2); √(answ1^2 + answ2^2)]
```

Here is the error for the integral representing the maximum curve integral

```
errintmax[tU_, ΔtU_, tL_, ΔtL_] :=
  Module[{}, n2 = Length[maxparams5]; answ3 = √(erintmaxseries[tU]^2 + ΔtU^2 + Δtph1^2);
   answ4 = √(erintmaxseries[tL]^2 + ΔtL^2 + Δtph1^2); √(answ3^2 + answ4^2)];
```

b) Correlated Errors

Because we have mentioned the possibility of systematic errors and the coefficients for the power series coefficients are far from independent as shown by the displayed covariance matrices from each fit, we develop the errors propagated through the covariance matrix rather than through the sum of squares alone. First we have to put the mincovariances "matrix" into a proper form that can do a matrix multiplication. The first complication is that the mincurve1["CovarianceMatrix"]//MatrixForm used above disguises the fact that the assignment to mincovariance put the matrix one *Mathematica* expression deeper than you might think from the displayed form above. In addition for the minimum curve, we have to eliminate the fact that the last row-column is for the tph argument whose error form is different from that of the coefficients themselves. Hence the restoration of the proper list structure that is one row column shorter than **mincovariance** is needed for the calculation of the coefficient covariances. As long as matrices are of small dimension their handling in *Mathematica* is relatively straight forward but when the matrices are built from those constructed in the nonlinear least squares routine it is a bit more complicated. The following algorithms were found by trial and error to be of the correct list depth required by *Mathematica* operations. These are the correlated prediction errors for the power series produced by the least squares process.

Correlated Errors for Minimum Curve Expansion

```
serminseries[te_] :=
 Module[{}, Clear[i, j]; ns = Length[minerrors]; s = Table[{j - 1}, {j, 1, ns - 1}];
  aminmatrix = Table[Table[mincovariance[[1, i, j]], {i, 1, ns - 1}], {j, 1, ns - 1}];
  vector = Table[((te - minparams[[9]])^s[[i]]), {i, 1, (ns - 1)}];
  bmatrix = Flatten[aminmatrix.vector]; Transpose[vector].bmatrix]
```

```
serminarg[te_] :=
 Module[{}, Clear[i, j]; ns = Length[minerrors]; s = Table[{j - 1}, {j, 1, ns - 1}];
  vector = Table[(minparams[[i]] s[[i]] (te - minparams[[9]])^(s[[i]]-1)
     Abs[minerrors[[9]]], {i, 2, (ns - 1)}]; Flatten[Transpose[vector].vector]]

serminte[te_] :=
 Module[{}, Clear[i, j]; ns = Length[minerrors]; s = Table[{j - 1}, {j, 1, ns - 1}];
  vector = Table[(minparams[[i]] s[[i]] (te - minparams[[9]])^(s[[i]]-1) Abs[Δte1],
    {i, 2, (ns - 1)}]; Flatten[Transpose[vector].vector]]
```

Correlated Errors for Maximum Curve Expansion

```
sermaxseries[te_] := Module[{}, Clear[i, j]; nt = Length[maxerrors5];
  amaxmatrix = Table[Table[maxcovariance[[1, i, j]], {i, 1, nt}], {j, 1, nt}];
  ss = Table[{j - 1}, {j, 1, ns}]; vector = Table[((te - tmax1)^ss[[i]]), {i, 1, nt}];
  bmatrix = Flatten[amaxmatrix.vector]; Transpose[vector].bmatrix]

sermaxarg[te_] :=
 Module[{}, Clear[i, j]; nt = Length[maxerrors5]; ss = Table[{j - 1}, {j, 1, nt}];
  vector = Table[(maxparams5[[i]] ss[[i]] (te - tmax1)^(ss[[i]]-1)) Abs[minerrors[[9]]],
    {i, 2, nt}]; Flatten[Transpose[vector].vector]]

sermaxte[te_] :=
 Module[{}, Clear[i, j]; nt = Length[maxerrors5]; ss = Table[{j - 1}, {j, 1, nt}];
  vector = Table[(maxparams5[[i]] ss[[i]] (te - tmax1)^(ss[[i]]-1)) Abs[Δte1], {i, 2, nt}];
  Flatten[Transpose[vector].vector]]
```

Total error for the power series representing the minimum curve

```
serrmin[te_] :=
 Module[{}, answ1 = serminseries[te] + serminarg[te] + serminte[te]; √answ1 ]
```

Total error for the power series representing the maximum curve

```
serrmax[te_] :=
 Module[{}, answ2 = sermaxseries[te] + sermaxarg[te] + sermaxte[te]; √answ2 ]
```

Using the analogous pattern, we do the “prediction” errors for the max and min integrals according to the propagation of correlated errors. (Meyer,1975)

Correlated Errors for Minimum Curve Integral

```
serintminseries[te_] :=
 Module[{}, Clear[i, j]; ns = Length[minerrors]; s = Table[{j - 1}, {j, 1, ns - 1}];
  aminmatrix = Table[Table[mincovariance[[1, i, j]], {i, 1, ns - 1}], {j, 1, ns - 1}];
  vector = Table[((te - minparams[[9]])^(s[[i]]+1))/(s[[i]] + 1), {i, 1, (ns - 1)}];
  bmatrix = Flatten[aminmatrix.vector]; Transpose[vector].bmatrix]
```

```
serintminarg[te_] :=
 Module[{}, Clear[i, j]; ns = Length[minerrors]; s = Table[{j - 1}, {j, 1, ns - 1}];
  vector = Table[(minparams[[i]] (te - minparams[[9]])^s[[i]]) Abs[minerrors[[9]]],
    {i, 2, (ns - 1)}]; Flatten[Transpose[vector].vector]]

serintminte[te_] :=
 Module[{}, Clear[i, j]; ns = Length[minerrors]; s = Table[{j - 1}, {j, 1, ns - 1}];
  vector = Table[(minparams[[i]] (te - minparams[[9]])^s[[i]]) Abs[Δte1],
    {i, 2, (ns - 1)}]; Flatten[Transpose[vector].vector]]
```

Correlated Errors for Maximum Curve Integral

```
serintmaxseries[te_] := Module[{}, Clear[i, j]; nt = Length[maxerrors5];
  amaxmatrix = Table[Table[maxcovariance[[1, i, j]], {i, 1, nt}], {j, 1, nt}];
  ss = Table[{j - 1}, {j, 1, ns}]; vector = Table[((te - tmax1)^(ss[[i]]+1)/(ss[[i]] + 1)), {i, 1, nt}];
  bmatrix = Flatten[amaxmatrix.vector]; Transpose[vector].bmatrix]

serintmaxarg[te_] :=
 Module[{}, Clear[i, j]; nt = Length[maxerrors5]; ss = Table[{j - 1}, {j, 1, nt}];
  vector = Table[(maxparams5[[i]] (te - tmax1)^ss[[i]]) Abs[minerrors[[9]]], {i, 2, nt}];
  Flatten[Transpose[vector].vector]]

serintmaxte[te_] :=
 Module[{}, Clear[i, j]; nt = Length[maxerrors5]; ss = Table[{j - 1}, {j, 1, nt}];
  vector = Table[(maxparams5[[i]] (te - tmax1)^ss[[i]]) Abs[Δte1], {i, 2, nt}];
  Flatten[Transpose[vector].vector]]
```

Total correlated error for the integral over the minimum curve (note that for some reason these functions need to be immediately evaluated.)

```
serrintmin[tuL0_, tlL0_] :=
 Module[{}, answ1 = serintminseries[tuL0] + serintminarg[tuL0] + serintminte[tuL0];
  answ2 = serintminseries[tlL0] + serintminarg[tlL0] + serintminte[tlL0];
  √(answ1[[1]] + answ2[[1]])]
```

Total correlated error for the integral over the maximum curve

```
serrintmax[tuU0_, tlU0_] :=
 Module[{}, answ3 = serintmaxseries[tuU0] + serintmaxarg[tuU0] + serintmaxte[tuU0];
  answ4 = serintmaxseries[tlU0] + serintmaxarg[tlU0] + serintmaxte[tlU0];
  √(answ3[[1]] + answ4[[1]])]
```

c) Non-linear Propagation - A Novel Monte Carlo Technique

Because we are concentrating on computational aspects of astrophysics we have chosen to skip consideration of the very involved process of exploring higher order Taylor

series expansions. Instead we explore some Monte Carlo approaches to non-linear error analysis. The philosophy for doing a Monte Carlo investigation of the errors is a bit different from standard propagation. Instead of getting the prediction errors from an expansion of the first order random or systematic errors, we determine the errors directly through perturbations around the mean values. These "random" sample deviations are then averaged by the standard deviation "rule" to produce an estimate of the average prediction error for the whole curve or integral. First we do the minimum curve and then the maximum curve. Secondly we will do the integrals of those curves. Rather than trying to do a random perturbation of the observations themselves and doing repeated least squares solutions, we prefer to just perturb the **covariance matrix** values themselves and combine them with random fluctuations. The mm is the number of MC values created, tfluct = the time flucuation in days, and cfluct = the allowed coefficient fluctuation in fractions of the covariance value.

Monte Carlo Errors for Minimum Curve Expansion

```
errMCUmin[mm_, tfluct_, cfluct_] :=
 Module[{}, dev = Table[0, {ii, 1, mm}]; dev1 = Table[0, {ii, 1, mm}]; minU = 8000;
  minL = 2800; Δτ = tfluct; Clear[i, j]; trial := RandomReal[{minL, minU}];
  Δtime := RandomReal[{-tfluct, tfluct}]; ns = Length[minparams]; Do[time = trial;
   dev[[ii]] = (mincurve /. t → (time + Δtime)) - (mincurve /. t → time);, {ii, 1, mm}];
  s = Table[{j - 1}, {j, 1, ns - 1}];
  Do[dev1[[ii]] = Module[{}, aminmatrix = Table[Table[mincovariance[[1, i, j]]
          (1 + RandomReal[{-cfluct, cfluct}]) , {i, 1, ns - 1}], {j, 1, ns - 1}];
     vector = Table[((trial - minparams[[9]])^s[[i]]), {i, 1, (ns - 1)}];
     bmatrix = Flatten[aminmatrix.vector];
     Transpose[vector].bmatrix] - Module[{}, aminmatrix =
      Table[Table[mincovariance[[1, i, j]] , {i, 1, ns - 1}], {j, 1, ns - 1}];
     vector = Table[((trial - minparams[[9]])^s[[i]]), {i, 1, (ns - 1)}];
     bmatrix = Flatten[aminmatrix.vector]; Transpose[vector].bmatrix],
   {ii, 1, mm}]; sd = 0; sd1 = 0; Do[sd = sd + dev[[ii]]^2;
   sd1 = sd1 + Abs[dev1[[ii]]];,
   {ii, 1, mm}]; Sqrt[(sd + sd1)/(2 (mm - 1))]]
```

Monte Carlo Errors for Maximum Curve Expansion

```
errMCUmax[mm_, tfluct_, cfluct_] :=
 Module[{}, dev = Table[0, {ii, 1, mm}]; dev1 = Table[0, {ii, 1, mm}]; maxU = 11500;
  maxL = 7500; Δτ = tfluct; Clear[i, j]; trial := RandomReal[{maxL, maxU}];
  Δtime := RandomReal[{-tfluct, tfluct}]; ns1 = Length[maxerrors5]; Do[time = trial;
   dev[[ii]] = (maxcurve4 /. t → (time + Δtime)) - (maxcurve4 /. t → time);,
   {ii, 1, mm}]; s = Table[{j - 1}, {j, 1, ns1}];
  Do[dev1[[ii]] = Module[{}, aminmatrix = Table[Table[maxcovariance[[1, i, j]]
          (1 + RandomReal[{-cfluct, cfluct}]) , {i, 1, ns1}], {j, 1, ns1}];
     vector = Table[((trial - tmax1)^s[[i]]), {i, 1, ns1}];
     bmatrix = Flatten[aminmatrix.vector]; Transpose[vector].bmatrix] - Module[{},
     aminmatrix = Table[Table[maxcovariance[[1, i, j]] , {i, 1, ns1}], {j, 1, ns1}];
     vector = Table[((trial - tmax1)^s[[i]]), {i, 1, ns1}];
     bmatrix = Flatten[aminmatrix.vector]],
   {ii, 1, mm}]; sd = 0; sd1 = 0; Do[sd = sd + dev[[ii]]^2;
   sd1 = sd1 + Abs[dev1[[ii]]];, {ii, 1, mm}]; Sqrt[(sd + sd1)/(2 (mm - 1))]]
```

Monte Carlo Errors for Minimum Curve Integrals

```
errMCUintmin[mm_, tfluct_, cfluct_, tU0_, tL0_] :=
 Module[{}, dev1 = Table[0, {ii, 1, mm}]; dev2 = Table[0, {ii, 1, mm}];
  integmin = ∫_tL1^tU1 (rv0 - mincurve) dt; Δtime1 := RandomReal[{-tfluct, tfluct}];
  Δtime2 := RandomReal[{-tfluct, tfluct}];
  ns = Length[minparams]; s = Table[{j - 1}, {j, 1, ns - 1}];
  minU = 8000; minL = 2800; trial := RandomReal[{minL, minU}];
  Do[dev1[[ii]] = (integmin /. {tU1 → (tU0 + Δtime1), tL1 → tL0 + Δtime2}) -
      (integmin /. {tU1 → tU0, tL1 → tL0});, {ii, 1, mm}];
  s = Table[{j - 1}, {j, 1, ns - 1}]; Do[dev2[[ii]] =
    Module[{}, aminmatrix = Table[Table[mincovariance[[1, i, j]]
          (1 + RandomReal[{-cfluct, cfluct}]) , {i, 1, ns - 1}], {j, 1, ns - 1}];
     vector = Table[((trial - minparams[[9]])^s[[i]]), {i, 1, (ns - 1)}];
     bmatrix = Flatten[aminmatrix.vector];
     Transpose[vector].bmatrix] - Module[{}, aminmatrix =
      Table[Table[mincovariance[[1, i, j]] , {i, 1, ns - 1}], {j, 1, ns - 1}];
     vector = Table[((trial - minparams[[9]])^s[[i]]), {i, 1, (ns - 1)}];
     bmatrix = Flatten[aminmatrix.vector];
     Transpose[vector].bmatrix], {ii, 1, mm}]; sd = 0;
  sd1 = 0; Do[sd1 = sd1 + dev1[[ii]]^2; sd = sd + Abs[dev2[[ii]]];, {ii, 1, mm}];
  Sqrt[(sd + sd1)/(2 (mm - 1)) + Δrv0^2 + 2 tfluct^2]]
```

Monte Carlo Errors for Maximum Curve Integrals

```
errMCUintmax[mm_, tfluct_, cfluct_, tU0_, tL0_] :=
 Module[{}, dev1 = Table[0, {ii, 1, mm}];
  dev2 = Table[0, {ii, 1, mm}]; integmax = ∫_tL2^tU2 (maxcurve4 - rv0) 𝕕t;
  Δtime1 := RandomReal[{-tfluct, tfluct}]; Δtime2 := RandomReal[{-tfluct, tfluct}];
  ns1 = Length[maxerrors5]; s = Table[{j - 1}, {j, 1, ns1}];
  maxU = 11500; maxL = 7500; trial := RandomReal[{maxL, maxU}];
  Do[dev2[[ii]] = (integmax /. {tU2 → (tU0 + Δtime1), tL2 → (tL0 + Δtime2)}) -
      (integmax /. {tU2 → tU0, tL2 → tL0});, {ii, 1, mm}];
  Do[dev1[[ii]] = Module[{}, aminmatrix = Table[Table[maxcovariance[[1, i, j]]
          (1 + RandomReal[{-cfluct, cfluct}]) , {i, 1, ns1}], {j, 1, ns1}];
      vector = Table[((trial - tmax1)^s[[i]]), {i, 1, ns1}];
      bmatrix = Flatten[aminmatrix.vector]; Transpose[vector].bmatrix] - Module[{},
      aminmatrix = Table[Table[maxcovariance[[1, i, j]] , {i, 1, ns1}], {j, 1, ns1}];
      vector = Table[((trial - tmax1)^s[[i]]), {i, 1, ns1}]; bmatrix =
       Flatten[aminmatrix.vector]; Transpose[vector].bmatrix], {ii, 1, mm}];
  sd = 0; sd2 = 0; Do[sd2 = sd2 + dev2[[ii]]^2; sd = sd + Abs[dev1[[ii]]];,
   {ii, 1, mm}]; √((sd + sd2)/(2 (mm - 1)) + Δrv0^2 + 2 tfluct^2) ]
```

We now conclude the digression concerning the theory of errors. We hope the student realizes the importance of rigorous error analysis to modern astronomy and astrophysics. It takes a long time to gain a good grasp of the statistics and probability and so being able to adapt the *Mathematica* routines presented here is a way around a lack of statistical experience. For example, students should be able to substitute appropriate Gaussian (or some other error model) pseudorandom generators for the uniform ones used above in the Monte Carlo calculations. One final point is that although a big fuss about errors is made during the data analysis stage, students are disappointed to learn that when it comes to reporting the results and the errors, all this effort is given only passing notice (often one line at most) in the actual research paper. That is because unless there were special problems encountered, the correctness of the error analysis is taken for granted.

Returning to the Determination of Orbital Elements

Here are the errors for the integral of the (rv0-minimum curve) with assumed time errors

The timing of such observations is generally very good, so we chose 0.1 day = 2.4 hours as a tenative choice. But notice that there is not much difference between 0.5 or 0.1 day in the results.

```
Δtu𝕃 = 0.1; Δtl𝕃 = 0.1; tfluct = 0.1;
```

The traditional error calculations appear to give fairly large which often leads to any error

analysis being given at all.

```
integ1
```

```
11 872.2
```

```
σmin = errintmin[tuL, ΔtuL, tlL, ΔtlL]
```

```
19 307.1
```

The trailing component notations [[ ]] are needed to remove the braces that *Mathematica* leaves in the function results. They occur because of the Module[] use in the functions. In this case the failure of convergenece leaves the answer on level deeper than expected so we use Flatten[] and then [[1]] This is an example of a failure of first order theory with correlated errors (it diverges to a large number) while the Monte Carlo value is stable and stays small.

```
σmins = serrintmin[tuL, tlL]
```

```
427 201.
```

```
σminMC = errMCUintmin[10 000, 0.5, 0.5, tuL, tlL][[1]]
```

```
1.55402
```

Here are the errors of the integral of the (maximum curve-rv0) with assumed time errors

```
ΔtuU = 0.1; ΔtlU = 0.1; tfluct = 0.1;
```

The traditional error calculations appear somewhat smaller because the maximum is more “peaked” while the minimum is very broad.

```
integ2
```

```
11 875.8
```

```
σmax = errintmax[tuU, ΔtuU, tlU, ΔtlU]
```

```
71.8082
```

The trailing component notations [[ ]] are needed to remove the braces that *Mathematica* leaves in the function results. They occur because of the Module[] use in the functions. Notice that the correlated variable computation has led to the error being larger than the integral. This occurs because of an instability in the least-squares solution.

```
σmaxs = serrintmax[tuU, tlU]
```

```
37.0122
```

```
σmaxMC = errMCUintmax[10 000, 0.5, 0.5, tuU, tlU][[1]]
```

```
0.731293
```

Now we reintegrate the fits to get the areas and amplitudes from max to the γ velocity and from the min to the γ velocity as required by the orbit solution shown by Smart.

Here is where the propagated errors start to become of the same order as the answer as was mentioned above. Thus we carry along the three error estimates nearly to the end as the final element error estimates so that each type of error estimate may be seen in context.

The maximum curve area Δ1

```
tuL0 = max2[[5, 1, 2]];
```

Here we use the actual functional maximum time for the maximum time

```
tmax0
```

```
10062.3
```

```
tlL0 = tmax0;
```

Here are the earlier assumed time errors.

```
Δtu L0 = 0.1; Δtl L0 = 0.1;
```

```
lims2
```

```
25.4269
```

$$\Delta 1 = \int_{tlL0}^{tuL0} (\text{maxcurve4} - \text{rv0})\, dt$$

```
5250.2
```

Here are the error estimates

```
Δ01a = errintmax[tuL0, Δtu L0, tlL0, Δtl L0]
```

```
23.8758
```

```
Δ01b = serrintmax[tuL0, tlL0]
```

```
12.8894
```

```
Δ01c = errMCUintmax[10000, 0.5, 0.5, tuL0, tlL0][[1]]
```

```
1.36536
```

The minimum curve area Δ2

```
tuU = min2[[5, 1, 2]];
```

```
tlU = tmin0; Δtmin0 = Δtph1;
```

```
Δtu = 0.1; Δtl = Δtmin0;
```

$$\Delta 2 = \int_{tlU}^{tuU} (\text{rv0} - \text{mincurve})\, dt;$$

Here are the error estimates of the minimum curve integral

```
Δ02a = errintmin[tuU, Δtu U, tlU, Δtl U]
```

```
19386.
```

```
Δ02b = serrintmin[tuU, tlU]
```

```
438 493.
```

```
Δ02c = errMCUintmin[10 000, 0.5, 0.5, tuU, tlU][[1]]
```

```
1.71642
```

The α and β parameters are the amplitudes of the maximum and minimum portions of the radial velocity curves. The errors in α and β follow the error analysis function for the polynomial expansion itself derived above. The minimum curve amplitude is given first

```
β = rv0 - (mincurve /. t → tmin0)
```

```
3.41067
```

```
Δββ1 = errmin[tmin0, Δtmin0]
```

```
1.33588
```

```
Δββ2 = serrmin[tmin0][[1]]
```

```
44.7062
```

```
Δββ3 = errMCUmin[10 000, 0.5, 0.5][[1]]
```

```
1.39372
```

Here is the error for the amplitude of representing the maximum curve

```
α = (maxcurve4 /. t → tmax0) - rv0
```

```
5.68085
```

Here is the error for the integral representing the minimum curve integral

```
Δαα1 = errmax[tmax0, Δtmin0]
```

```
6.2969 × 10^-11
```

```
Δαα2 = serrmin[tmax0][[1]]
```

```
200.914
```

```
Δαα3 = errMCUmin[10 000, 0.5, 0.5][[1]]
```

```
1.40473
```

The binary velocity amplitude is K=kay. Griffin gives 4.85 km/s

```
kay = (α + β) / 2
```

```
4.54576
```

We combine the errors strictly according to rms average. Sometimes if the chain rule is applied the factor of two will show up in the error calculation

```
Δkay1 = √(Δαα1^2 + Δββ1^2)
```

```
1.33588
```

$$\Delta\text{kay2} = \sqrt{\Delta\alpha\alpha 2^2 + \Delta\beta\beta 2^2}$$

205.828

$$\Delta\text{kay3} = \sqrt{\Delta\alpha\alpha 3^2 + \Delta\beta\beta 3^2}$$

1.97882

$$\text{ecos}\omega = \frac{\alpha - \beta}{\alpha + \beta};\ \text{esin}\omega = \frac{2\sqrt{\alpha\,\beta}}{\alpha + \beta}\,\frac{\Delta 2 - \Delta 1}{\Delta 2 + \Delta 1};$$

$$\omega = \text{ArcTan}[\text{ecos}\omega, \text{esin}\omega]$$

0.769602

Argument of periapsis is ω. Griffin gives 28.1 degrees

$$\omega \,/\, \text{Degree}$$

44.095

$$\text{ecos}\omega 1 = \frac{\alpha 1 - \beta 1}{\alpha 1 + \beta 1};\ \text{esin}\omega 1 = \frac{2\sqrt{\alpha 1\,\beta 1}}{\alpha 1 + \beta 1}\,\frac{\Delta 02 - \Delta 01}{\Delta 02 + \Delta 01};$$

$$\text{w} = \text{ArcTan}\left[\left(\frac{\text{esin}\omega 1}{\text{ecos}\omega 1}\right)\right];$$

Here are the three error types propagated now as random errors because we no longer know the correlations.

$$\Delta\text{wa} = \sqrt{(\partial_{\alpha 1}\text{w}\ \Delta\alpha 1)^2 + (\partial_{\beta 1}\text{w}\ \Delta\beta 1)^2 + (\partial_{\Delta 01}\text{w}\ \Delta\Delta 01)^2 + (\partial_{\Delta 02}\text{w}\ \Delta\Delta 02)^2}\ /.$$
$$\{\alpha 1 \to \alpha,\ \beta 1 \to \beta,\ \Delta\alpha 1 \to \Delta\alpha\alpha 1,\ \Delta\beta 1 \to \Delta\beta\beta 1,\ \Delta 01 \to \Delta 1,\ \Delta\Delta 01 \to \Delta 01\text{a},\ \Delta 02 \to \Delta 2,\ \Delta\Delta 02 \to \Delta 02\text{a}\}$$

2.11466

$$\Delta\text{wa} \,/\, \text{Degree}$$

121.161

$$\Delta\text{wb} = \sqrt{(\partial_{\alpha 1}\text{w}\ \Delta\alpha 1)^2 + (\partial_{\beta 1}\text{w}\ \Delta\beta 1)^2 + (\partial_{\Delta 01}\text{w}\ \Delta\Delta 01)^2 + (\partial_{\Delta 02}\text{w}\ \Delta\Delta 02)^2}\ /.$$
$$\{\alpha 1 \to \alpha,\ \beta 1 \to \beta,\ \Delta\alpha 1 \to \Delta\alpha\alpha 2,\ \Delta\beta 1 \to \Delta\beta\beta 2,\ \Delta 01 \to \Delta 1,\ \Delta\Delta 01 \to \Delta 01\text{b},\ \Delta 02 \to \Delta 2,\ \Delta\Delta 02 \to \Delta 02\text{b}\}$$

60.2815

$$\Delta\text{wb} \,/\, \text{Degree}$$

3453.87

$$\Delta\text{wc} = \sqrt{(\partial_{\alpha 1}\text{w}\ \Delta\alpha 1)^2 + (\partial_{\beta 1}\text{w}\ \Delta\beta 1)^2 + (\partial_{\Delta 01}\text{w}\ \Delta\Delta 01)^2 + (\partial_{\Delta 02}\text{w}\ \Delta\Delta 02)^2}\ /.$$
$$\{\alpha 1 \to \alpha,\ \beta 1 \to \beta,\ \Delta\alpha 1 \to \Delta\alpha\alpha 3,\ \Delta\beta 1 \to \Delta\beta\beta 3,\ \Delta 01 \to \Delta 1,\ \Delta\Delta 01 \to \Delta 01\text{c},\ \Delta 02 \to \Delta 2,\ \Delta\Delta 02 \to \Delta 02\text{c}\}$$

0.477958

```
Δwc / Degree
```

```
27.385
```

The eccentricity ecc given by Griffin is 0.328

$$\text{ecc1} = \frac{\text{ecos}\omega}{\text{Cos}[\omega]}$$

```
0.347685
```

$$\text{ecc2} = \frac{\text{esin}\omega}{\text{Sin}[\omega]}$$

```
0.347685
```

Here is a symbolic version that can be used to derive the errors.

$$\text{ecc3} = \frac{\text{ecos}\omega 1}{\text{Cos}[w]};$$

$$\Delta\text{eccA} = \sqrt{\left((\partial_{\alpha 1}\,\text{ecc3}\,\Delta\alpha 1)^2 + (\partial_{\beta 1}\text{ecc3}\,\Delta\beta 1)^2 + (\partial_{\Delta 01}\text{ecc3}\,\Delta\Delta 01)^2 + (\partial_{\Delta 02}\text{ecc3}\,\Delta\Delta 02)^2\right)} /.$$
$$\{\alpha 1 \to \alpha,\ \beta 1 \to \beta,\ \Delta\alpha 1 \to \Delta\alpha\alpha 3,\ \Delta\beta 1 \to \Delta\beta\beta 3,\ \Delta 01 \to \Delta 1,\ \Delta\Delta 01 \to \Delta 01a,\ \Delta 02 \to \Delta 2,\ \Delta\Delta 02 \to \Delta 02a\}$$

```
0.716078
```

$$\Delta\text{eccB} = \sqrt{\left((\partial_{\alpha 1}\,\text{ecc3}\,\Delta\alpha 1)^2 + (\partial_{\beta 1}\text{ecc3}\,\Delta\beta 1)^2 + (\partial_{\Delta 01}\text{ecc3}\,\Delta\Delta 01)^2 + (\partial_{\Delta 02}\text{ecc3}\,\Delta\Delta 02)^2\right)} /.$$
$$\{\alpha 1 \to \alpha,\ \beta 1 \to \beta,\ \Delta\alpha 1 \to \Delta\alpha\alpha 3,\ \Delta\beta 1 \to \Delta\beta\beta 3,\ \Delta 01 \to \Delta 1,\ \Delta\Delta 01 \to \Delta 01b,\ \Delta 02 \to \Delta 2,\ \Delta\Delta 02 \to \Delta 02b\}$$

```
15.8346
```

$$\Delta\text{eccC} = \sqrt{\left((\partial_{\alpha 1}\,\text{ecc3}\,\Delta\alpha 1)^2 + (\partial_{\beta 1}\text{ecc3}\,\Delta\beta 1)^2 + (\partial_{\Delta 01}\text{ecc3}\,\Delta\Delta 01)^2 + (\partial_{\Delta 02}\text{ecc3}\,\Delta\Delta 02)^2\right)} /.$$
$$\{\alpha 1 \to \alpha,\ \beta 1 \to \beta,\ \Delta\alpha 1 \to \Delta\alpha\alpha 3,\ \Delta\beta 1 \to \Delta\beta\beta 3,\ \Delta 01 \to \Delta 1,\ \Delta\Delta 01 \to \Delta 01c,\ \Delta 02 \to \Delta 2,\ \Delta\Delta 02 \to \Delta 02c\}$$

```
0.15078
```

Finding the periapsis time

The periapsis occurs when the radial velocity is above the γ velocity by the amplitude

$$\text{zee1} = \text{kay}\,(1 + \text{ecc1})\,\text{Cos}[\omega]$$

```
4.3998
```

We treat the errors here as uncorrelated as we have no way to determine the covariance. We first construct the function in dummy variables

$$\text{zee0} = \text{kay0}\,(1 + \text{ecc0})\,\text{Cos}[\omega 9];$$

$$\Delta\text{zeeA} = \sqrt{\left(\partial_{\text{kay0}}\,\text{zee0}\,\Delta\text{kay1}\right)^2 + (\partial_{\text{ecc0}}\,\text{zee0}\,\Delta\text{eccA})^2 + (\partial_{\omega 9}\,\text{zee0}\,\Delta\text{wa})^2} /.$$
$$\{\text{ecc0} \to \text{ecc1},\ \text{kay0} \to \text{kay},\ \omega 9 \to \omega\}$$

```
9.40221
```

$$\Delta\text{zeeB} = \sqrt{\left(\partial_{\text{kay0}}\ \text{zee0}\ \Delta\text{kay2}\right)^2 + \left(\partial_{\text{ecc0}}\,\text{zee0}\ \Delta\text{eccB}\right)^2 + \left(\partial_{\omega 9}\,\text{zee0}\ \Delta\text{wb}\right)^2}\ /.$$

$$\{\text{ecc0} \to \text{ecc1},\ \text{kay0} \to \text{kay},\ \omega 9 \to \omega\}$$

```
329.238
```

$$\Delta\text{zeeC} = \sqrt{\left(\partial_{\text{kay0}}\ \text{zee0}\ \Delta\text{kay3}\right)^2 + \left(\partial_{\text{ecc0}}\,\text{zee0}\ \Delta\text{eccC}\right)^2 + \left(\partial_{\omega 9}\,\text{zee0}\ \Delta\text{wc}\right)^2}\ /.$$

$$\{\text{ecc0} \to \text{ecc1},\ \text{kay0} \to \text{kay},\ \omega 9 \to \omega\}$$

```
2.83938
```

Or when the observed velocity is

```
rv0 + zee1
```

```
24.1458
```

Possible times of periapsis passage are given by Solve[], but there is no way to do an error analysis of this process. Instead we estimate the fractional error of the result from the fractional error of rv0+zee and assume the same fractional error for all the times. There are three real roots. The average fractional error is about 15%. And as can be seen this is too large to aid in ruling out spurious roots.

$$\left(\frac{\Delta\text{zeeA} + \Delta\text{zeeB} + \Delta\text{zeeC}}{3\ \text{zee1}}\right)$$

```
25.8709
```

The maximum velocity occurs at ν ~ - 44 degrees (when (ν+ω)=0). Since we have the period, eccentricity and the true anomaly, we can estimate the time by finding the eccentric anomaly and then the time difference from the maximum until periapsis when ν = 0.

```
tperiapse1 = Solve[maxcurve == rv0 + zee1, t]
```

```
{{t → 5735.74}, {t → 7263.69 - 1320.1 i}, {t → 7263.69 + 1320.1 i}, {t → 9371.11},
 {t → 10 687.1}, {t → 12 866.5 - 780.333 i}, {t → 12 866.5 + 780.333 i}}
```

As is well-know from elementary orbit analysis, the eccentric anomaly E is found from the eccentricty and the true anomaly by inverting this equation

$$\tan\frac{v}{2} = \left(\frac{1+e}{1-e}\right)^{1/2}\tan\frac{E}{2}$$

With the mean motion given by n = 2 π/T (T = period of orbit), the time difference from perihelion (t-τ) for each observation is obtained from Kepler's equation

$$n\,(t-\tau) = E - e\sin E$$

Here is the time of rv maximum when ν = -44 degrees

```
tmax0
```

```
10 062.3
```

Here we compute the mean anomaly from ν and E. The mean anomaly divided by the "annual" motion n then gives (t-τ) =Δtp1 where τ = the periapsis time. If it turns out to be negative then the Δtp1 must be subtracted algebraically from the observed time.

```
hmax = 2 ArcTan[Sqrt[(1 - ecc1)/(1 + ecc1)] Tan[-ω / 2]]; Δtp1 = ΔT3/(2 π) (hmax - ecc1 hmax)
```

```
-502.252
```

The equivalent symbolic expressions needed for the error analysis are followed by the three cases of errors

```
hmax1 = 2 ArcTan[Sqrt[(1 - ecc0)/(1 + ecc0)] Tan[-ω9 / 2]];
```

```
ΔhmaxA = Sqrt[(∂ecc0 hmax1 ΔeccA)^2 + (∂ω9 hamx1 Δwa)^2] /. {ecc0 → ecc1, ω9 → ω}
```

```
0.425252
```

```
ΔhmaxB = Sqrt[(∂ecc0 hmax1 ΔeccB)^2 + (∂ω9 hamx1 Δwb)^2] /. {ecc0 → ecc1, ω9 → ω}
```

```
9.40357
```

```
ΔhmaxC = Sqrt[(∂ecc0 hmax1 ΔeccC)^2 + (∂ω9 hamx1 Δwc)^2] /. {ecc0 → ecc1, ω9 → ω}
```

```
0.0895425
```

```
Δtp2 = ΔT4/(2 π) (hmax3 - ecc0 hmax3);
```

```
ΔtpA = Sqrt[(∂ecc0 Δtp2 ΔeccA)^2 + (∂hmax3 Δtp2 ΔhmaxA)^2 + (∂ΔT4 Δtp2 ΔΔT)^2] /.
  {ecc0 → ecc1, ΔT4 → ΔT3, hmax3 → hmax}
```

```
674.859
```

```
ΔtpB = Sqrt[(∂ecc0 Δtp2 ΔeccB)^2 + (∂hmax3 Δtp2 ΔhmaxB)^2 + (∂ΔT4 Δtp2 ΔΔT)^2] /.
  {ecc0 → ecc1, ΔT4 → ΔT3, hmax3 → hmax}
```

```
14 919.
```

```
ΔtpC = Sqrt[(∂ecc0 Δtp2 ΔeccC)^2 + (∂hmax3 Δtp2 ΔhmaxC)^2 + (∂ΔT4 Δtp2 ΔΔT)^2] /.
  {ecc0 → ecc1, ΔT4 → ΔT3, hmax3 → hmax}
```

```
142.947
```

Since the time of maximum is less than the time of periapsis, we must subtract the difference from the time of maximum.

```
ttp = tmax0 - Δtp1
```

```
10 564.6
```

The error of tmax0 is tiny so the error of ttp is essentially that of Δtp1, 45 to 25 days. Thus these differences can be trusted to tell that ttp is indeed the correct time difference.

```
(tperiapse1[[5, 1, 2]] - ttp)
```

```
122.542
```

```
(tperiapse1[[4, 1, 2]] - ttp)
```

```
-1193.44
```

The closest root to this in the set for tperiapse1 is the 5th one in the list. To get the JD of this passage, we add the root found to the "start" time of the time series for this star.

```
perijD = start + tperiapse1[[5, 1, 2]]
```

```
59 296.3
```

Griffin gets jD 50257 for periapsis passage. But our value is just ~ one period ahead of his. Since our period has an error of ~280 days, the value below is within the our period's error of Griffin's value.

```
perijD - ΔT3
```

```
50 488.8
```

This concludes the section where the traditional orbital elements (except for a or i) have been obtained.

Mass-related Quantities for HD 108613

At long last we return to the original objective, that of determining mass-related quantities. Of course when dealing with spectroscopic binaries even the best spectroscopic data by itself cannot obtain the inclination angle under any circumstance so that sin i always appears in the mass dependent functions. Because of Kepler's law the first mass dependent quantity is the semi-major axis a, which here is a sin i. The second quantity is the so called mass function which also involves the unknown sin i

Semi-major axis information

For an extremely long period orbit such as this one is, the a sin i value is very small indicating most likely a small inclination angle, i.e. the orbit is nearly face on where the spectroscopic information is hard to obtain. In km, the a sin i result is

$$\text{asini} = \frac{21\,600\,\Delta T3}{\pi}\,(\alpha+\beta)\,\sqrt{\left(1-\text{ecc1}^2\right)}$$

$$5.16195\times 10^{8}$$

or in AU :

$$\frac{\text{asini}}{149 \times 10^6}$$

3.4644

Here we perform the error analysis for a sin i. We start with a function that can be differentiated

$$\text{asini0} = \frac{21\,600\,\Delta\text{Ta}}{\pi}\,(\alpha 1 + \beta 1)\,\left(1 - \text{ecc3}^2\right);$$

$$\Delta\text{asiniA} = \sqrt{\left((\partial_{\alpha 1}\,\text{asini0}\,\Delta\alpha 1)^2 + (\partial_{\beta 1}\text{asini0}\,\Delta\beta 1)^2 + (\partial_{\Delta\text{Ta}}\text{asini0}\,\Delta\Delta\text{Ta})^2 + (\partial_{\Delta 01}\text{asini0}\,\Delta\Delta 01)^2 + (\partial_{\Delta 02}\text{asini0}\,\Delta\Delta 02)^2\right)} /.$$
$$\{\alpha 1 \to \alpha,\ \beta 1 \to \beta,\ \Delta\alpha 1 \to \Delta\alpha\alpha 3,\ \Delta\beta 1 \to \Delta\beta\beta 3,\ \Delta\text{Ta} \to \Delta\text{T3},\ \Delta\Delta\text{Ta} \to \Delta\Delta\text{T},$$
$$\Delta 01 \to \Delta 1,\ \Delta\Delta 01 \to \Delta 01\text{a},\ \Delta 02 \to \Delta 2,\ \Delta\Delta 02 \to \Delta 02\text{a}\}$$

2.989×10^8

$$\Delta\text{asiniB} = \sqrt{\left((\partial_{\alpha 1}\,\text{asini0}\,\Delta\alpha 1)^2 + (\partial_{\beta 1}\text{asini0}\,\Delta\beta 1)^2 + (\partial_{\Delta\text{Ta}}\text{asini0}\,\Delta\Delta\text{Ta})^2 + (\partial_{\Delta 01}\text{asini0}\,\Delta\Delta 01)^2 + (\partial_{\Delta 02}\text{asini0}\,\Delta\Delta 02)^2\right)} /.$$
$$\{\alpha 1 \to \alpha,\ \beta 1 \to \beta,\ \Delta\alpha 1 \to \Delta\alpha\alpha 3,\ \Delta\beta 1 \to \Delta\beta\beta 3,\ \Delta\text{Ta} \to \Delta\text{T3},\ \Delta\Delta\text{Ta} \to \Delta\Delta\text{T},$$
$$\Delta 01 \to \Delta 1,\ \Delta\Delta 01 \to \Delta 01\text{b},\ \Delta 02 \to \Delta 2,\ \Delta\Delta 02 \to \Delta 02\text{b}\}$$

6.06314×10^9

$$\Delta\text{asiniC} = \sqrt{\left((\partial_{\alpha 1}\,\text{asini0}\,\Delta\alpha 1)^2 + (\partial_{\beta 1}\text{asini0}\,\Delta\beta 1)^2 + (\partial_{\Delta\text{Ta}}\text{asini0}\,\Delta\Delta\text{Ta})^2 + (\partial_{\Delta 01}\text{asini0}\,\Delta\Delta 01)^2 + (\partial_{\Delta 02}\text{asini0}\,\Delta\Delta 02)^2\right)} /.$$
$$\{\alpha 1 \to \alpha,\ \beta 1 \to \beta,\ \Delta\alpha 1 \to \Delta\alpha\alpha 3,\ \Delta\beta 1 \to \Delta\beta\beta 3,\ \Delta\text{Ta} \to \Delta\text{T3},\ \Delta\Delta\text{Ta} \to \Delta\Delta\text{T},$$
$$\Delta 01 \to \Delta 1,\ \Delta\Delta 01 \to \Delta 01\text{c},\ \Delta 02 \to \Delta 2,\ \Delta\Delta 02 \to \Delta 02\text{c}\}$$

1.32372×10^8

The mass function

The mass function is defined as

$$\frac{m2^3 \sin^3 i}{(m1+m2)^2} = \frac{3.993 * 10^{-20}\,(a1 \sin i)^3}{P^2}$$

where m1 is the “visible” star mass in solar masses, m2 is the “unseen” star mass in solar masses, a1 is the semimajor axis of the visible star in kilometers and P is the period in days.

$$\text{massfunction} = \frac{3.993 \times 10^{-20}\,\text{asini}^3}{\Delta\text{T3}^2}$$

0.070801

The error analysis follows.

$$\text{massf} = \frac{3.993 \times 10^{-20}\,\text{asi}^3}{\Delta\text{T4}^2};$$

```
ΔmassfA = √((∂asi massf Δasi)^2 + (∂ΔT4 massf ΔΔT4)^2) /.
  {asi → asini, ΔT4 → ΔT3, ΔΔT4 → ΔΔT, Δasi → ΔasiniA}
```

```
0.123073
```

```
ΔmassfB = √((∂asi massf Δasi)^2 + (∂ΔT4 massf ΔΔT4)^2) /.
  {asi → asini, ΔT4 → ΔT3, ΔΔT4 → ΔΔT, Δasi → ΔasiniB}
```

```
2.49486
```

```
ΔmassfC = √((∂asi massf Δasi)^2 + (∂ΔT4 massf ΔΔT4)^2) /.
  {asi → asini, ΔT4 → ΔT3, ΔΔT4 → ΔΔT, Δasi → ΔasiniC}
```

```
0.0546522
```

Discussion

For this star (HD 108613), in addition to observed photometric variability there also seems to be a radial velocity anomaly near maximum AND minimum and perhaps also all around the radial velocity curve. In the case of the minimum, our curve fitting was able to fit the anomaly. In the case of the maximum, the curve fitting routines could not handle the anomaly. The companion cannot be detected, but we do know that we are observing the side of the more luminous star that faces toward its companion at maximum and at periapsis. At minimum we are observing the "backside" of the luminous companion and at apoapsis. The usual assumption is that the more luminous star is the primary, but we don't know that for sure. The enhanced radial velocity anomalies at maximum velocity may indicate a near edge-on orbit. But this is highly speculative. Other than the UBV magnitudes, there is little further information for this star. According to Griffin (2010), the (U-B) is between 1.85 and 2.03 while (B-V) is from 1.53 to 1.58. The fact that the (U-B) is larger than the (B-V) indicates a reddened object (i.e. hence distant). The spectrum type inferred from the (B-V) is K5 for a supergiant or K8 for a giant (Allen, 2000, p.389). The mass M1 would be 1.2 solar masses for a giant and 13 solar masses for a supergiant. The radial velocity anomalies (and indeed the scatter of radial velocity throughout its orbit) may be due to pulsations and so the identification with a red giant variable is more likely. First let us explore the supergiant scenario

```
iii = 45 °;
```

$$\text{Solve}\left[\text{massfunction} == \frac{\text{m2}^3\,\text{Sin}[\text{iii}]^3}{(13+\text{m2})^2},\ \text{m2}\right]$$

```
Solve::ratnz : Solve was unable to solve the system with inexact coefficients. The
    answer was obtained by solving a corresponding exact system and numericizing the result. ≫
{{m2 → -1.82209 - 2.34161 i}, {m2 → -1.82209 + 2.34161 i}, {m2 → 3.84443}}
```

At i= 90 degrees the resulting companion mass is 2.4 solar masses and with i even as low as 45 degrees, the result is 3.6 solar masses. Our numerical experiments thus cannot eliminate the possiblity that at the orbital inclination of 45 degrees, an invisible companion could be a 3 solar mass neutron star. That possibility makes this system merit additional

observations to see if that is indeed true. But that is a long shot. If the invisible companion is a black hole then any mass is possible and so a large mass root from the Solve[] could be considered valid. But those objects are even more unlikely.

The radial velocity anomalies (and indeed the scatter of radial velocity throughout its orbit) may be due to pulsations and so the identification with a red variable giant is more likely. The Solve[] we give next assumes a 1.2 solar mass luminous object (giant ?). That certainly is consistent with the optical data. It is also likely that M2 < M1. We show the face-on case (i = 90 degrees). You may change the inclination to establish the other possible combinations. These calculations do eliminate certain inclination values and indicate that it is possible that this object could be a very long period eclipsing variable like ϵ Aurigae and therefore a very interesting object for further study, but one that we can't pursue except computationally. At i -> 90 degrees, the invisible companion could be a main sequence star of mass less than the sun. In such a case, the primary would be over 6 or 7 magnitudes brighter than the secondary so an exotic explanation is not required. But without additional information no decision can be made about this object. It is hoped that through this somewhat extended discussion, the elements of an initial vetting of information about newly discovered objects is illustrated and that by going through this example process students will be better prepared when they are called upon to vet information on an object that they themselves discover.

```
iii = 85 °;
```

$$\text{Solve}\left[\text{massfunction} == \frac{\text{m2}^3\,\text{Sin}[\text{iii}]^3}{(1.2+\text{m2})^2}, \text{m2}\right]$$

Solve::ratnz : Solve was unable to solve the system with inexact coefficients. The answer was obtained by solving a corresponding exact system and numericizing the result. ≫

$$\{\{\text{m2} \to -0.273569 - 0.30303\,i\}, \{\text{m2} \to -0.273569 + 0.30303\,i\}, \{\text{m2} \to 0.618754\}\}$$

```
tB = SessionTime[];
(tB - tA) / 60
0.5777033
```