Theoretical Detection Limits and Error Reduction for Radial Velocity Observations of an Earth-Like Exoplanet

by

Allison L. Moberger

Submitted to the Department of Earth, Atmospheric, and Planetary Sciences
in Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Earth, Atmospheric, and Planetary Sciences
at the Massachusetts Institute of Technology

May 8, 2009
[June 2009]

Author Signature redacted
Department of Earth, Atmospheric, and Planetary Sciences
May 8, 2009

Certified by Signature redacted
Sara Seager
Thesis Supervisor

Accepted by Signature redacted
Samuel Bowring
Chair, Committee on Undergraduate Program

Abstract

The intent of this project was to determine the relationship between the number of radial velocity observations of an Earth-twin exoplanet and the error in the mass calculated from the detected signal. If the planet's period is known through prior transit observations, the mass may be measured by radial velocity more accurately; this project tested and measured the conditions for this error reduction.

Simulated sets of radial velocity data taken by HARPS (accurate to $1\,^{m}/_{s}$) for an Earth-mass planet in a circular, edge-on, 1 AU orbit around a Sun-like star were used with a least-squares fit to measure the amplitude of the sinusoidal radial velocity curve. The three conditions in which the mass fit was compared were: evenly-spaced observations with the period unknown; evenly-spaced observations with the period known; and an unevenly-spaced observation method in which observation times are chosen to be very frequent and clustered around the peaks of the radial velocity curve.

For evenly-spaced observations, knowledge of the period did not reduce the error in the mass measurement compared to the period-unknown case, though it did allow for the elimination of the false-negative detection case. When observations were evenly spaced, the percent error in the detected mass had a power law relationship with the number of observations of $\sigma_{\%\,error} = 1250N^{-0.5}$.

However, when using the knowledge of the period from transits to choose clustered observation times near the peaks of the curve, the error in the mass was reduced by about 20% for the same number of total observations, and was thus approximated by the power law $\sigma_{\%\,error} = 1030N^{-0.5}$. This indicates that if the period of a low-mass planet is known through transits, the use of clustered observations allows its mass to be measured more accurately with the same number of radial velocity observations than if the period were unknown.

Contents

List of Figures

1 Background and Motivation

Since the 1995 discovery of 51 Pegasi b, the holy grail of exoplanet research has been to find an extrasolar Earth-twin. Believed to be the first step to finding habitable planets outside our solar system, the discovery of an extrasolar Earth might in turn eventually lead to the discovery of extraterrestrial life. Yet the hypothetical Earth-twins orbiting other stars are still out of our reach with current exoplanetary discovery methods. As current detection methods are refined, however, the lowest mass of planets detected becomes closer and closer to Earth's mass; using a combination of methods may allow the likelihood and accuracy of Earth-mass planet detections to increase substantially.

1.1 Discovery methods

1.1.1 Radial velocity

Of the 347 exoplanets currently known (as of May 3, 2009), 321 have been discovered or observed using stellar radial velocity by Doppler spectroscopy [Schneider 2009], which is currently the most prolific exoplanetary discovery method. This method works by looking for small shifts in the spectral lines of a star; caused by the Doppler effect, the periodic red- and blue-shifting of the lines indicates that the star is moving toward and away from the observer. This small wobble is due to a companion star or planet in the system[1], as both the star and companion orbit their common center of mass. From the magnitude of the spectral shift a radial velocity can be calculated, and when plotted over time the velocity curve can be seen to be sinusoidal (though a companion in an eccentric orbit, or multiple companions, will cause more complex shapes). The semi-amplitude of the sinusoid – equivalent to the radial velocity of the star when the planet is at phase 0 or phase 0.5 – is given by

$$K = \frac{m_p \sin i}{m_*^{2/3}} \left(\frac{2\pi G}{P} \right)^{1/3}, \tag{1}$$

where m_p is the mass of the planet, m_* is the mass of the star, i is the inclination of the planet's orbit (defined as 90 degrees for an edge-on orbit and 0 degrees for a face-on orbit), P is the planet's orbital period, and G is the gravitational constant (this equation is derived fully in the Method section).

Because the magnitude of the signal is directly proportional to the planetary mass[2], low-mass planets are very difficult to detect using this method unless they are in a multi-planet system with other higher-mass planets; the smallest planet discovered by radial velocity is GJ 581 e, with a mass of 1.9 Earth-masses, though it is a system with three other planets of 5 Earth-masses or greater [Mayor et al. 2009a, preprint]; Figure 1 shows the discovery radial velocity curve of GJ 581 e as an example of the data from which the smallest signal yet was detected.

[1] It should be noted that this method also detects the radial velocity shifts in a star due to a companion star, so the companion mass must be calculated to differentiate between signals due to planets and signals due to binary stars.

[2] The inclination cannot be determined from radial velocity alone, in which case the mass parameter is usually given as $m_p \sin i$, which serves as a lower bound on the mass; later transit observations may determine that the inclination is less than 90 degrees, which leads to a higher planet mass.

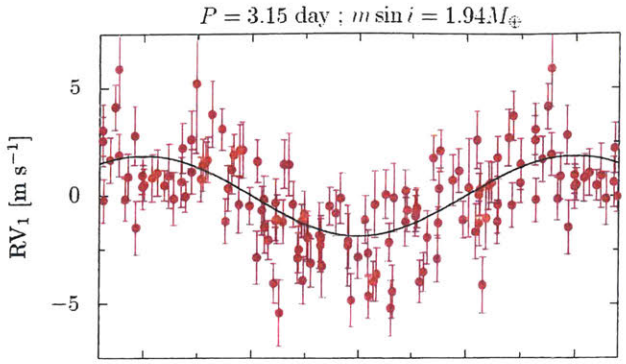


Figure 1: Discovery radial velocity curve for GJ 581 e
Shown is the residual radial velocity data, after the subtraction of the signals due to the other three planets in the system, from which GJ 581 e was discovered and its mass calculated to be 1.9 M_{Earth}. Note that, because GJ 581 e is so close to its star (it orbits with a period of only 3.15 days), its radial velocity semi-amplitude is approximately 2 m/s, allowing its mass to be measured accurately with the ± 1 m/s data obtained by HARPS. Earth's semi-amplitude is only 0.09 m/s.

Previous to this recent discovery, the lowest-mass planet known was HD 40307 b, which has a mass 4.2 times that of Earth, though it too is in a system with at least two other larger planets [Mayor et al. 2009b]. The smallest planet discovered by radial velocity in a system by itself is GJ 176 b, which is 8.42 Earth-masses [Forveille et al. 2009].

1.1.2 Transits

The next most productive exoplanetary discovery method is photometric transit detection, which can detect 58 of the 347 known planets [Schneider 2009]. Transits are observed when the flux from a star drops slightly as a planet passes in front of the star, and then returns to normal later when the planet is no longer along a direct line of sight to the star[3]. From the duration and depth of the transit, the relative size of the planet to the star and its distance to the star may be calculated; spectroscopic observations may be needed in addition to precisely determine the star's size so that the planet's radius can be determined. For a planet of radius R_p in an orbit of semi-major axis a, inclination i, and orbital period P orbiting a star of radius R_*, the duration of the transit is

[3]While a transit is also observed with some binary stars, the magnitude of the flux drop may be used to calculate the relative size of the transiting object, which often helps to differentiate transiting planets from transiting binary stars. However, this method still may produce false positives in cases of grazing binaries (in which the orbit is inclined enough that one star does not fully pass in front of the other), blends (in which the large flux drop due to transiting binaries is diluted by a third unrelated star close to the line of sight), and secondary eclipses (in which the occultation of the dimmer member of the binary by the brighter one is observed, which may have a flux drop much lower than the primary eclipse/transit of the two stars and hence comparable to that of a planet transiting a star). Other methods, such as radial velocity, are often used to check systems that a promising transit marks as a good candidate, to help eliminate these possibilities.

$$t_T = \frac{PR_*}{\pi a}\sqrt{\left(1+\frac{R_p}{R_*}\right)^2 - \left(\frac{a\cos i}{R_*}\right)^2}. \quad (2)$$

Likewise, the transit depth is given by

$$\Delta F = \left(\frac{R_p}{R_*}\right)^2. \quad (3)$$

From these above equations and the size of the star, the size of the planet can be calculated; however, it should be noted that transits give an observer no information about the planet's mass. Because the depth of the transit scales with the radius of the planet relative to the radius of the star, large planets orbiting small stars are preferentially detected using this method; so far, the smallest planet discovered by transit observations is CoRoT-7 b, with a radius of 1.8 Earth-radii and a mass of 11.1 Earth-masses [Rouan et al. 2009, in preparation].

1.1.3 Other methods

Other exoplanetary discovery methods besides these two exist. Gravitational microlensing utilizes the bending of light in gravitational fields, and can detect planets orbiting stars, when both are too dim to observe directly. As the planet's parent star passes in front of an observable star, the light from the bright background star is bent gravitationally as if the parent star were a lens, and tiny variations in the brightened signal from the background star can prove the presence of planets. However, microlensing can only be used to take a census and not to learn anything more about the planets as they are undetectable after the individual lensing event.

Direct imaging is the most straightforward and least scientifically-complex method, as well as one of the best for public relations from the exoplanetary field. Observers simply take images of stars and block out the light from the star itself, allowing the much-dimmer planet to be seen. This method favors very large planets or brown dwarfs orbiting at great distances from their parent stars, though, which are not favorable conditions for the habitability of those planets.

Pulsar timing is the detection of tiny variations in the very regular periodic pulses from pulsars due to a planet altering the pulsar's rotation rate. Pulsars are, however, not at all sun-like, and planets orbiting them would be uninhabitable due to the high levels of ionizing radiation they emit. In total these methods account for only 26 of the 347 known planets [Schneider 2009], and are not discussed further in this paper.

1.2 Project introduction

Focusing on the radial velocity and transit methods of exoplanet discovery, it is clear that neither method on its own is particularly well-suited to detect Earth-mass planets orbiting Sun-like stars. For example, if we were to observe our solar system from the outside using radial velocity (assuming an edge-on orbit), the radial velocity signal due to the Earth would have a semi-amplitude of about 9 cm/s, while the signal due to Jupiter would have a semi-amplitude of over 13 m/s: Jupiter produces about 150 times the signal Earth does. Even GJ 176 b, the lowest-mass planet in a single-planet system detected by radial velocity, produces a signal of 4.4 m/s, still nearly 50 times the signal Earth would produce. The Keck telescope is cited as able to measure radial velocities as low as 3 m/s [NASA 2008], but this is still over 30 times greater than the resolution needed to constitute a discovery

of an Earth-mass planet. Even HARPS (the High Accuracy Radial velocity Planet Searcher), the ESO instrument used to detect GJ 581 e (the recently-discovered 1.9 m_{Earth} planet), can measure radial velocity to only 1 m/s accuracy [ESO 2003]. This is low enough error to detect GJ 581 e, which orbits extremely close to its star with a period of just over 3 days [Mayor et al. 2009a, preprint] (meaning its radial velocity semi-amplitude is nearly 2 m/s – certainly detectable with HARPS, though many observations are still required), but not nearly low enough to detect a planet of similar mass with a longer period like that of Earth. It is clear that Earth-mass planets at Earth distances from their stars are still undetectable by radial velocity, as their 10 cm/s signals are likely to be entirely lost in the noise of the data unless we obtain a large number of radial velocity observations.

However, promising developments are being made in the technology used to detect transits of small planets. NASA's new Kepler Mission launched a wide-field photometric telescope into an Earth-trailing orbit in March 2009 and saw first light in April. Kepler will conduct a four-year survey of about 100,000 stars in the region of Cygnus to look for transits by Earth-size planets without the interference of the Sun or the Earth's atmosphere [NASA 2008]. However, while the transit of an Earthlike planet may be detectable by Kepler, the mass remains an unknown quantity when planetary candidates are observed by transits alone. Ideally, once a planet is discovered transiting its star, it will be observed with radial velocity to obtain an $m_p \sin i$ parameter for the new planet. Any planet as small as Earth, though, has such a small radial velocity semi-amplitude that the value of $m_p \sin i$ is not very accurate.

Nonetheless, observation of transits gives us one other important piece of information about the planet besides its size: from transit timing, we can determine a planet's orbital period. The period is one of the two variables in the radial velocity semi-amplitude equation, the other being the $m_p \sin i$ parameter, which is effectively equivalent to m_p for transiting planets, because transits can only occur when i is close to 90 degrees. It is possible that knowing the period already from the discovery transit observations could improve the radial velocity fit enough that, with many observations, the low-mass planet's signal is detectable in the noise. The radial velocity observations would then allow its mass to be calculated relatively accurately from the fit of the signal.

The intent of this project is to determine the relationship between the number of observations and the error in the mass detection, and how it varies when the period is known versus when it is unknown.

2 Method

2.1 Fitting

The first part of the investigation was to create code which could find the mass and period of the best-fit curve to supplied radial velocity data. To accomplish this, I first used the C programming language to create data and artificially add random Gaussian noise to it. The data was created using Jupiter's mass and period initially to test the fitting algorithm, and calculated using equation (1) as the amplitude of a sine wave with its period changed to that of Jupiter's orbit:

```
K = (1./3600) * (mJupiter/pow(mSun,2./3)) * pow((2*pi*Ghr/pJupiter),(1./3));
Vtest[index] = K * sin((2*pi/pJupiter)*i);
```

Here the factor of $1/3600$ is to convert the value of `K` into meters per second, as `pJupiter` is in hours and `Ghr` is the universal gravitational constant converted into $m^3/kg \cdot hr^2$. The value of `index` is a simple counter that advances through the members of the array holding the values, while `i` is 100 times this and tells the observation time in hours; here we assume one data point for every 100 hours, because it is impractical (and unnecessary) to manage more data points than one every 100 hours of Jupiter's 12-year orbit.

Then Gaussian noise was calculated by using the central limit theorem, which states that the average of N uniformly-distributed random variables is itself a random variable with an approximately Gaussian distribution for large N. I used 10 random integers uniformly distributed between -5 and 5, multiplied by an arbitrary value and averaged, and this value is added to `Vtest[i]` to create `Vobs[i]`. The arbitrary factor in the noise allows the signal-to-noise ratio (SNR) to be varied. The SNR is defined in the context of this paper as

$$SNR = \frac{\sqrt{N} \cdot K}{\sqrt{2} \cdot \sigma} \tag{4}$$

for σ being the standard deviation of the noise, N the number of measurements made, and K and V_{test} as defined in the code above. The factor of $\sqrt{2}$ comes from the definition of SNR as relative to the root mean square of the signal. For a sinusoidal signal with amplitude A, the root mean square is given by $rms = \frac{A}{\sqrt{2}}$. Equation 4 is derived from equations 23 and 24 in [Gaudi & Winn 2007].

2.1.1 Chi squared versus least-squares

Once the data had been created, I designed a subroutine which would take several arguments (a mass, a period, the array containing the points to be fit, and the length of that array) and return the value of the fit parameter, χ^2. The definition of this parameter, which is minimized by the best fit, is given as $\chi^2 = \sum_{all\, i} \frac{(V_{obs}[i] - V_{fit}[i])^2}{V_{fit}[i]}$ for the Pearson chi-square test. Because I was fitting sinusoidal data that could have either positive or negative values, in order to properly calculate χ^2 (which is always positive), I added an absolute-value operator to the denominator.

However, it was soon discovered that while in general the minimization of this parameter gave a good fit for high-SNR data, the fits were poor for medium to low SNRs. I think that this is because the calculation of χ^2 has an inherent bias – because $V_{fit}[i]$ is in the denominator, it returns lower values for fits with higher values, even if the higher-valued curve is not as good a fit as a lower-valued one might be. This was a recurring problem even with SNRs as large as 2, for which the fit returned a mass 1.5 times the correct value. This error was most apparent when, as a troubleshooting technique, I changed the value of K to 0 (thereby giving an SNR of 0), and the chi squared fitting algorithm returned a $1.4\, m_{Jupiter}$ signal. At this point I decided that χ^2 was simply not the best fit parameter for the form and SNRs of my data, and began investigating different fit parameters instead.

I settled on using the least-squares test instead because it was similar enough to χ^2 that it would not be difficult to implement in the code I had already built, but would eliminate the bias inherent in χ^2. The fit parameter in least-squares, called $error^2$ here, is calculated by $error^2 = \sum_{all\, i} (V_{obs}[i] - V_{fit}[i])^2$, and minimizing this parameter gave fits accurate to 1% even for an SNR of 10^{-2}.

I later found that it would have been more appropriate to use a different form of χ^2, which is instead given by $\chi^2 = \sum_{all\, i} \left(\frac{V_{obs}[i] - V_{fit}[i]}{\sigma} \right)^2$, which gives a chi-square

distribution; this would have solved the problem I was having with using the chi-squared fit parameter, but by the time I learned of this mistake I had already switched to using least-squares.

2.1.2 Minimization of fit parameter

Initially on constructing the code to fit the data, I had written the fit-parameter-minimization part of the routine using the simplistic construct of nested for loops. This program would loop through the possible values of m_p in a given range for a given step size, and for each value loop through all of the possible values of P in a given range for a given step size, calculating the fit parameter for each pair. A simple test was all that was required to remember the pair of values that gave the lowest possible value for the fit parameter.

While very easy to implement, this "grid" method also has several drawbacks. First, it requires the user to input the minimum and maximum values for the mass and period and does not search outside this range for better fits. A wider range, however, has a tradeoff with time: checking for a single planet between 0.1 and $5\,m_{Jupiter}$ regularly took three to five minutes regardless of SNR, because the algorithm needed to complete a check of all the values in the table whether or not it had already found the minimum. The inefficient nature of the nested loops made testing even small changes to the code frustrating, and I decided I needed to find a new minimization method that would make the search for a minimum more efficient.

When I learned about the downhill simplex method for minimization, I hoped that it would solve the problems I was having. This works by creating a triangle from three points in the period-mass plane and calculating the value of the fit parameter at each vertex. It then makes one of several moves to improve these values: it first tries reflecting the point with the highest $error^2$ across the opposite edge of the triangle. If this is an improvement to the error of the point, it tries reflecting the point farther to expand the triangle and checks again if this is an even better value of $error^2$. If the reflection of the worst point is not an improvement, the minimum must be inside the triangle, so it is contracted toward the point with the lowest value of $error^2$. The process is repeated until the triangle has contracted far enough that its vertices are within a given "tolerance" distance of one another, and then the vertex with the best $error^2$ is returned as the minimum. This process is also called the amoeba method because it can be thought of as an amoeba which flips its way downhill on a three-dimensional function (the plane being the period and mass, and the height above the plane being the value of the fit parameter) and, once it discovers that it has reached a valley, contracts downward into the bottommost point in the valley.

However, the Nelder-Mead simplex method has a known problem as well: the amoeba cannot distinguish between a small local minimum and a larger absolute minimum, meaning it can get stuck in a rut and find a point that is overall a poor fit but happens to be closer to the initial conditions than the true minimum value. I discovered this problem only after becoming rather reliant on the simplex method, so finding a way to resolve this issue was at first a daunting task. However, I realized eventually that I had the tools I needed already in my initial fitting method! I combined the grid method with the simplex method, making the grid loose enough that it took a relatively short time, yet found an absolute minimum in the possible region (I set the range to 0.5 to 5 $M_{Jupiter}$) to serve as a starting point in the simplex method. This finally eliminated the sources of error inherent to the simplex method relative to the grid method, while retaining the relatively fast performance of the simplex method relative to the grid method.

Using this combination of fitting and minimization methods, the new algorithm was relatively involved to translate into code, but quickly proved it was worth it when my tests of data sets that were giving erroneous results with one or the other fit methods before returned extremely accurate answers (within 0.1% in both mass and period) in just a few seconds. Finally, with an effective fit parameter and an efficient minimization algorithm, I could begin answering my original research question.

2.2 Simulation of Earth-mass planets

When scaling the data down to Earth-mass planets, we must recall that not only is the mass (and therefore the signal) lessened by several orders of magnitude, but also because the period is about twelve times shorter, a data point every 100 hours gives us far fewer data points to fit, meaning the fits will be less accurate than the Jupiter data for comparable SNRs. We also must remember that realistic SNRs for Jupiter-mass and Earth-mass planets are very different, and thus comparable SNRs are unlikely.

In this case, we assume that all data is taken to an accuracy of ± 1 m/s, a reasonable level of noise for our current best low-mass planet detection techniques. (± 1 m/s is the noise level of HARPS, used to detect GJ 581 e – the detection curve shown in Figure 1 – so this is not an unreasonable lower limit on the noise inherent in observations.) We also know the semi-amplitude to be expected from Earth is about 9 cm/s. As SNR is defined as in equation (4), from these two values we are able to calculate the SNR at which our observations will be fixed, which is a function of N:

$$SNR_{Earth} = \frac{\sqrt{N} \cdot K}{\sqrt{2} \cdot \sigma} = \frac{\sqrt{N} \cdot 0.09\,\mathrm{m/s}}{\sqrt{2} \cdot 1\,\mathrm{m/s}} = 0.0636 \cdot \sqrt{N}. \tag{5}$$

We can also derive what the standard deviation of the Gaussian noise we are adding to the data to create `Vobs[]`. This value will then be multipled by a value called `noiseFactor` in the noise creation algorithm to give the correct SNR for our data. The product of `noiseFactor` and the standard deviation of the basic noise creation algorithm should produce the value of σ in the SNR equation. From knowing that the standard deviation of equally-spaced integers between -5 and 5 is $\sqrt{10}$ and according to the central limit theorem, the standard deviation of the average of those points goes as $\sigma = \frac{\sigma_i}{\sqrt{N}} = 1$ for $N = 10$, I initially estimated an initial (completely serendipitous) value of 1 for `noiseFactor`. Experimentally, however, after substituting m_{Earth} and P_{Earth} for the Jupiter values in my code, I found that the SNR was coming out to about $0.058 \cdot \sqrt{N}$, so I changed the value of noiseFactor to $\frac{0.0636}{0.0580} \approx 1.097 = \sqrt{1.2}$, which corrected the SNR to within less than 1% of the correct value.

I decided at this point that in order to better explore the relationship between the number of observations and the error in the mass, the next step would be to implement a system for running many trials of creating and fitting data, averaging the relevant quantities, and graphing them to extrapolate their relationships. In order to do this, I turned the entire fitting and grid/simplex algorithm into a subroutine of my C program which took three arguments. The first was `nPds`, the number of periods for which we observe every so often. I decreased the frequency of observations to every 25 hours, which may be somewhat unrealistic but increased the number of points without having to calculate many many periods' worth of data. The other two arguments were the coefficients of P_{Earth} in the initial conditions of the period

coordinate in the simplex method; this allowed me to set the values to serve as the upper and lower bounds (0.5 to 2 gave good results) of the starting grid when I wanted the period to be a fit variable, and to set them both to 1 when I wanted the period to be a known quantity. This subroutine, called `fitter`, returned void, but instead put the key values it had calculated into a static array which could be read in the main routine.

Next I created a new, relatively simple main routine consisting of two loops (one with the period as a fit variable and one with it known, using the two arguments as discussed above) which would start from an initial value of `nPds` and increase it in steps, running the `fitter` subroutine to calculate the fit, SNR, number of observations, and mass error many times at each step, and printing the results array each time. I planned to then calculate the number of observations for each different `nPds` and to plot this against the standard deviation of the percent error in the mass produced by each `nPds` value. This would give me a basic understanding of the relationship between number of observations and mass error and how it changed if the period was known.

2.2.1 The false-negative problem

The problems that I discovered were at first frustrating, but not difficult to resolve. First I found that I needed to institute a cutoff for the simplex algorithm to break out of its loop if it did not converge after some number of steps (as this was likely an indication that the triangle was stuck flipping back and forth between two equally-bad values, implying that the minimum happened to lie along the opposite edge of the triangle) and used 500 steps as a cutoff. This resolved most of the problems I was having with the program getting stuck; I also had the program write the `nPds` and counter values to the terminal so that I could track how much progress it had made, as several hundred trials each for 10 values of `nPds` (3, 6, 9... 27, 30) was making the program take nearly an hour to run. Then I ran the program for conditions with the period as a fit variable and with the period known.

I tested this process first by running it 100 times at each step, and tested several `nPds` initial, maximum, and step values. At first I ran the program with `nPds`=1 periods up to 15 periods and noticed that the graph did not show enough to characterize the curves' full behavior, so I raised the maximum to 30 periods' worth of observations and, to save on computation time, calculated data points for every 3 periods of observing rather than each additional period. I then increased the number of trials of each `nPds` value to 300, in order to ensure that theaverages calculated from the data were sufficiently well-sampled, which would (I hoped) make the relationship I would graph smoother.

When the program had finished running, I opened the resulting data in Excel and calculated the data points I wanted to graph. I calculated N, the number of observations corresponding to each `nPds`, by multiplying `nPds` by the number of hours in a year (87600) and divided by the frequency of observations (25 hours). I also discarded any runs in which the percent error on the mass was over 10000, then took the standard deviation of the percent mass error in the remaining runs for each `nPds`. This gave the value of the mass error which 68% of the runs had error less than or equal to. I then plotted these points on a graph with N on the x-axis and $\sigma_{\%\,error}$ on the y-axis (Figure 2).

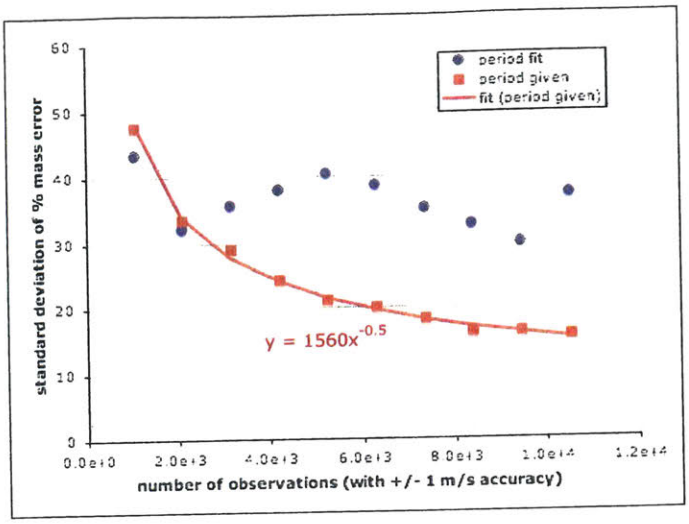


Figure 2: Comparison of mass error between period-known and period-fit cases for 3 to 30 periods of data
This graph resulted from the first running of my fitting algorithm. Each point represents the standard deviation of 100 trials of fitting data taken every 25 hours for 3, 6, 9, ... 30 periods, with the period known (red) versus the period as a fit variable (blue). Mass error is well-behaved with known period, but not when period is a fit variable.

Immediately it was obvious that there was a significant difference in the mass errors achieved with the period known versus the period as a fit variable. But more intriguing, however, was the different shapes of the curves. The data for the known period was well-fit by the power law relationship $\sigma = 1560N^{-0.5}$, but the data for the period as a fit variable was a much stranger shape. Why would the error first decrease and then begin to increase again for an increasing number of observations? At first I thought this could be an artifact of my somewhat unconventional grid/simplex fitting algorithm, but a change in the grid step size returned no change in the data, so the origin had to come from somewhere else. I then thought that perhaps the number of points was too low for the algorithm to correctly detect the low-SNR signal, and ran it again for 40, 50, 60, 70, and 80 periods' worth of data (still observing every 25 hours) and added these to Figure 2 to get Figure 3.

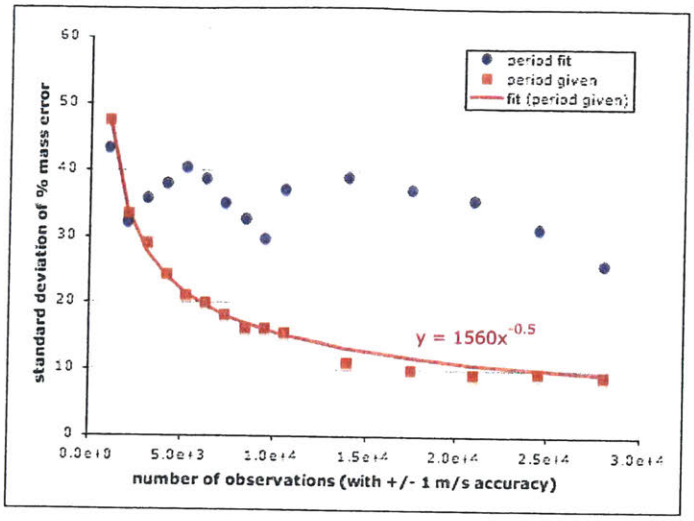


Figure 3: Comparison of mass error between period-known and period-fit cases with the addition of 40 to 80 periods of data

Shown is the same data from Figure 2, with points calculated for 40, 50, ... 80 periods added. These points continue to show the same behavior and imply that the addition of many more observations does not significantly improve the mass calculated by radial velocity.

The points in the period known case continued to fit the same power law curve of $\sigma = 1560N^{-0.5}$, but the new points in the period fit case showed the mass error again increasing and then decreasing too slowly for it to be practical to simulate out further. Clearly there was a problem endemic to the period fit case that was causing it to take a strange shape. In order to investigate this further, I made a histogram of the percent mass errors returned by the algorithm for the period fit case, and found that instead of the Gaussian curve I expected, I had a different situation (Figure 4).

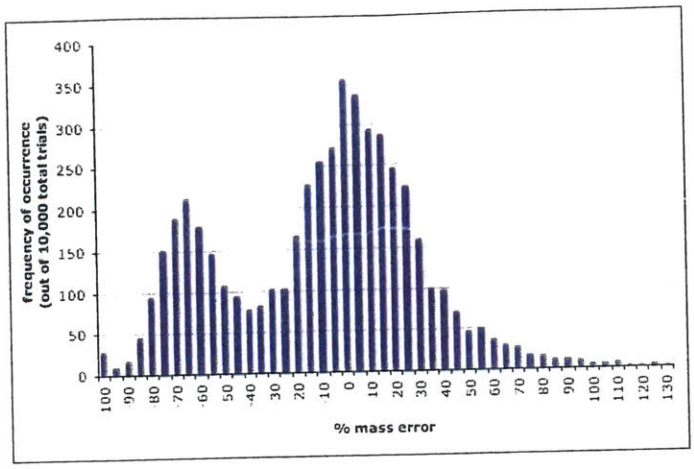


Figure 4: Percent mass error histogram for 3 to 80 periods of data in the period-fit case only

This histogram of the mass errors in the period-fit case showed, instead of a basic normal distribution, a secondary peak centered on very low-mass measurements, implying that the fit algorithm was sometimes finding no signal in the simulated data.

Noting that the secondary peak was in the direction of negative mass errors, I checked how this translated into actual mass numbers, plotting the mass found against the trial number, and immediately saw the problem (Figure 5).

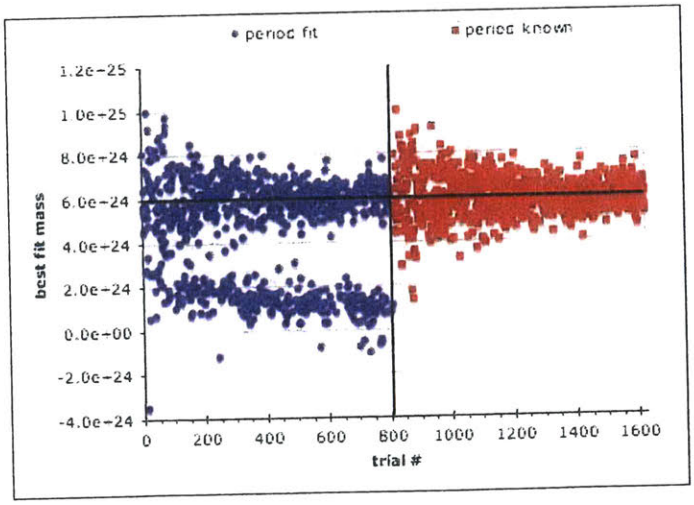


Figure 5: Distribution of masses detected in each trial of the period-fit and period-given cases

The distribution of detected masses shows, in the period-fit case only, a secondary cluster of masses much lower than the correct value, implying that the fitting algorithm found a spurious low-mass signal – or no signal at all – in the data.

From this I realized that the period fit case must be occasionally not detecting the signal in the data (not a surprising result, with an SNR as low as this) and returning a mass close to zero, or finding a spurious low-amplitude signal caused by the noise rather than the planet's signal. Effectively, even with a large number of observations, the period fit was occasionally returning a false negative, which was throwing off the data in the curve in Figures 2 and 3.

I also noticed that in Figure 5 there was a gap around $m_p = 3 \times 10^{24}$ kg, which given an Earth mass of 6×10^{24} kg, corresponds to the "valley" in Figure 4 around -50%. I decided that the best way to compare the relationship between the number of points and the percent error in the mass across the two period conditions, I needed to eliminate these false negative cases when the fitting did not detect the correct signal. I did this by having Excel remove the values of percent mass error that were less than -50% from the calculation of the standard deviation, and ran the whole program again for `nPds` = 10, 20, 30, ... 80, with 100 trials for each value of `nPds`, and achieved Figure 6.

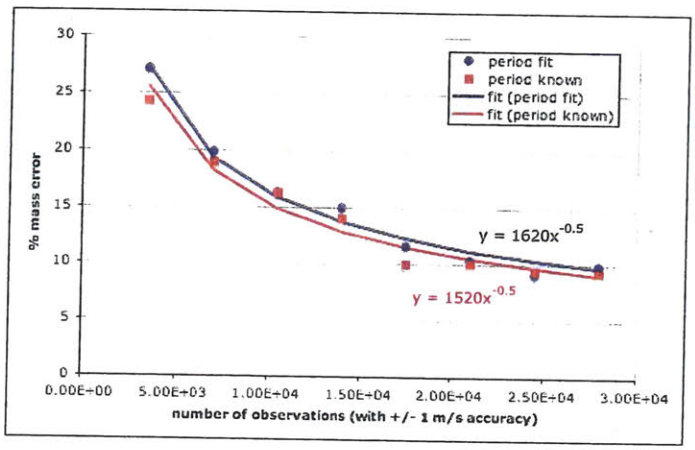


Figure 6: First characterization of mass error relationship with N as a power law curve

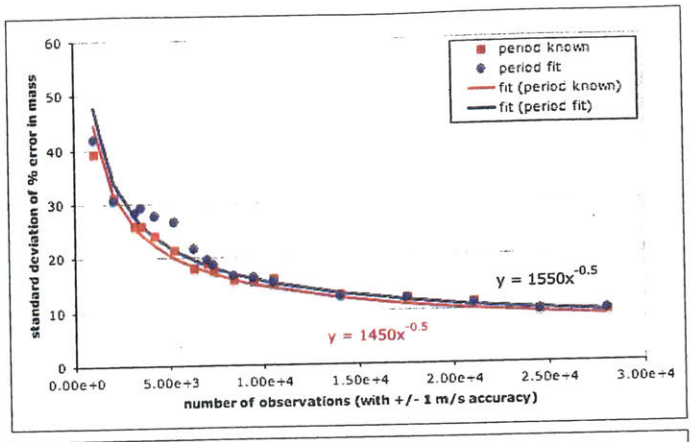
After correction for the spurious low-mass signals, the relationship between standard deviation of percent mass error and the number of observations for 10, 20, ... 80 periods of data are fit by power-law curves.

From Figure 6 I saw that the problematic behavior from Figures 2 and 3 was finally gone.

2.2.2 Preliminary results

Finally, with a working fitting algorithm, I ran the program again for `nPds` = 10, 20, 30, ... 80, this time with 500 trials for each. I calculated the percent error in mass and the number of observations for each, and generated a graph showing these values for both period known and unknown. Curious as to whether the curves diverged for lower numbers of observations, I repeated this process for `nPds` = 3, 6, 9, ... 30 and superimposed these points on the other graph. This result is shown in Figure 7a; only the `nPds` = 3, 6, 9, ... 30 points are shown in Figure 7b.

a.)



b.)

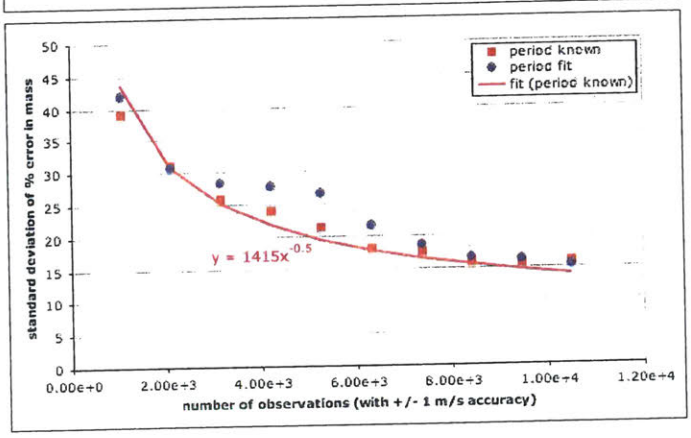


Figure 7: 500 trials of each period case introduces a new unusual behavior in the period-fit case

Graph (a) shows a divergence of the period-fit case from the period-known case between 9 and 21 periods. This region of the graph is more clearly shown in graph (b), which shows fits for 3, 6, ... 30 periods.

What I found was actually more interesting than I'd expected: though both curves were well-fit by power laws that differed only slightly ($\sigma = 1550N^{-0.5}$ for the period fit case, and $\sigma = 1450N^{-0.5}$ for the period known case), there appeared to be a divergence from the power law fit in the period fit case between 9 periods and 21 periods. I added a graph of only the period fit points outside these boundary values and found that the power-law fit to these points was exactly identical to the fit of the period known case ($\sigma = 1450N^{-0.5}$), as shown in Figure 8 (the black curve fully overlaps the red curve, so both cannot be seen). This meant that the points between these boundary values were the only reason the period fit case's power-law fit differed from the fit of the period known case. I then needed to investigate just what was causing these points to diverge.

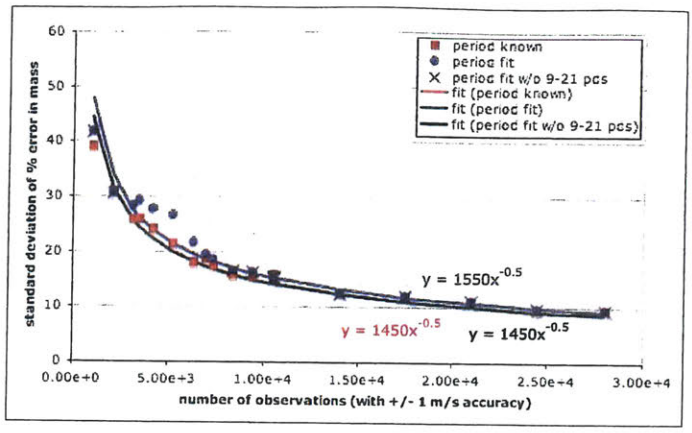


Figure 8: Comparison of mass error without divergent points included

When the same data from Figure 7a is again graphed but with the exclusion of period-fit points between 9 and 21 periods, the fit curve exactly matches the period-known curve. Shown are the period-fit case with divergent points included (blue), with divergent points removed (black), and the period-known case (red; completely overlaid by black curve). This suggests that the divergence between 9 and 21 is not a significant difference between the two period conditions.

I discovered that in the histogram of the percent errors from each trial in the divergent range, the "false negative" secondary peak phenomenon was still present, but not being corrected for by the -50% cutoff I previously imposed on the data, because the secondary peak had migrated and had in fact grown even larger than the primary peak centered at 0, as seen in Figure 9. This meant the algorithm was finding spurious low-mass signals in this range of the data far more often than in other ranges.

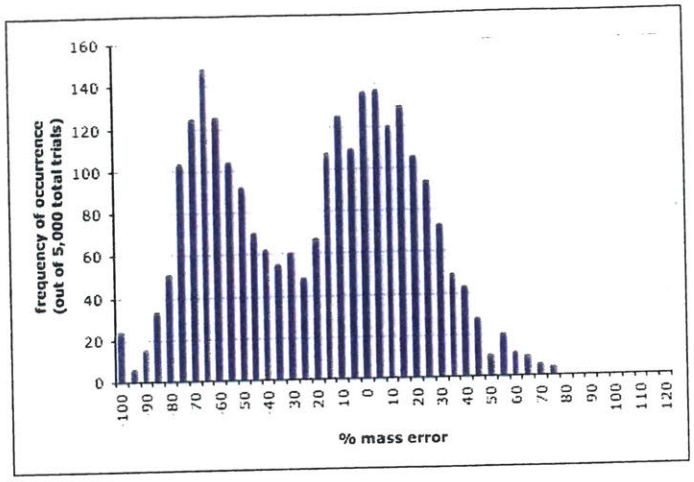


Figure 9: Percent mass error histogram for divergent points between 9 and 21 periods worth of data
This histogram shows the occurrences of each mass error for the trials involving 9 through 21 periods of data only. In this realm the fitting algorithm detects a spurious low-mass signal almost as often as it detects the correct mass, causing the divergence in the standard deviation of the mass error.

Based on the information shown in this histogram, I tried lowering the cutoff for inclusion from -50% to -25%, which produced Figure 10. It is clear that this completely corrects for the error in all but the `nPds` $= 12$ and 15 cases, which it lowers substantially, and also lowers the values of σ at higher numbers of observations.

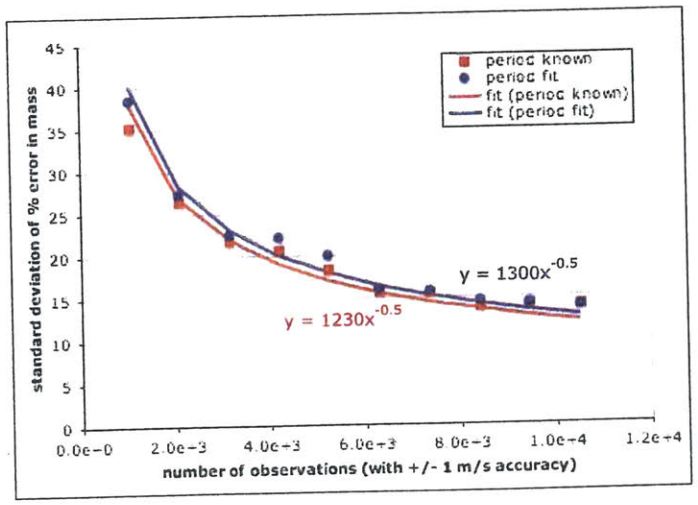


Figure 10: The divergence is removed in the period-fit case
When the lower cutoff for the inclusion of a trial is revised from -50% to -25%, the points from 9 through 21 periods' worth of data are no longer divergent.

At this point, I had a result that clearly showed that just knowing the period did not actually improve the error significantly in a mass fit when the observations are evenly-spaced, every 25 hours throughout many periods. This may in part be due to the quality of the fitting algorithm; the simplex is an efficient and effective 2-dimensional minimization algorithm, and combined with the grid to eliminate the possibility of finding a small local minimum near the initial conditions, it is reasonable for a well-constructed 2-dimensional fit to be just as good as a 1-dimensional fit. The dependence of the error on the number of observations follows a power law of the form $\sigma_{\%\,error} = \frac{A}{\sqrt{N}}$ for some constant A, which is unsurprising as SNR is itself a function of $\sqrt{N}$. Effectively, the standard deviation of the mass error is dependent on SNR, whether or not the period is known. The relationship between SNR and $\sigma_{\%\,error}$ is shown in Figure 11.

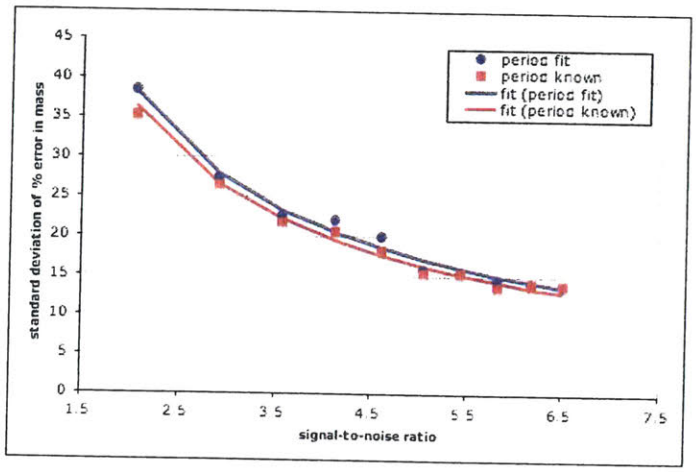


Figure 11: Percent mass error has a power-law dependence on SNR

The power-law dependence of mass error on SNR is fit by an equation of the form $\sigma = A \cdot SNR^{-0.8}$.

Objectively, the result was also somewhat to be expected. Regardless of the minimization algorithm, it is not uncommon for researchers in this field to use a Fourier transform to create a periodogram, which graphically shows the peak periodic signal of the data. For most radial velocity data, finding the period from a periodogram is basically equivalent to knowing it from transit data, though low-SNR data may decrease the effectiveness of the periodogram. The identical graphs were basically unsurprising – enough observations to find a period is enough observations to get a relatively accurate mass.

The basic limitation to finding what might be considered a "more interesting" answer to this question is basically the same as the limitation to finding Earth-mass planets: the extremely low signal-to-noise ratio of the data. It is possible that for higher SNRs the curves in Figure 10 might diverge more, in which case gaining an exoplanet's period information from a transit before measuring the mass with equally-spaced radial velocity observations could significantly improve the accuracy of the mass measurement as the number of observations increases. This result remains, though, that for evenly-spaced observations even over many periods, the

9 cm/s signal produced by Earth-mass planets orbiting at 1 AU is simply small enough compared to the 1 m/s error inherent in our best instruments that changing from a 2-variable to 1-variable fit does not significantly improve the mass detection. It does, however, reduce the likelihood of false negatives, because the false-negative secondary peaks were observed only on the histograms of the period-fit data and not in the period-known cases.

2.3 The peak-observation method

I did not feel my research question had been fully answered by the equally-spaced observations result discussed above. Surely more effective applications of the period from transit observations were possible than to simply feed it to the 2-dimensional fit algorithm. I chose to investigate one particular application of period data that seemed, intuitively, like it would be the most useful: using knowledge of the period to choose observation times judiciously. I chose to investigate by how much the mass error was reduced if one were to take radial velocity measurements with high frequency for a short window of time centered on the peaks of the radial velocity curve, and not in between: what I call the "peak-observation method".

For the sake of comparison, I decided to standardize the number of data points per period across the observation methods, so that the fit would not be better in one method over the other simply because it had more data points. As observing every 25 hours for a year gives 350 observations per period, I approximated this by using 360 observations per period in the peak-observation method (simply because it was roundly divisible by 4, for reasons that should become clear shortly). The difference between these two values is not significant in the reduction of the noise; if we assume, as in the previous section, that $\sigma_{\%\,error} = \frac{A}{\sqrt{N}}$, then the standard deviation of the percent error for 350 points per period is only $\frac{\sigma_{360}}{\sigma_{350}} = 1.014$ times the error for 360 points per period. So, a resultant change of more than 1% error would be significant and due to something other than this addition of 10 points per period.

Because each period will have two observation windows (one at the maximum of the sine curve and one at the minimum), there will therefore be 180 points in each window. I chose to make the frequency of these observations a variable called `freq`, so that I could compare the error resulting from differing values. The two windows, then, were $\frac{P}{4} - 90\cdot\texttt{freq} \leq t \leq \frac{P}{4} + 90\cdot\texttt{freq}$ and $\frac{3P}{4} - 90\cdot\texttt{freq} \leq t \leq \frac{3P}{4} + 90\cdot\texttt{freq}$, for period P and observations every `freq` hours. I then wrote the code to create noisy data points at the given frequency in these windows, and used the same least-squares and grid/simplex minimization algorithms as before but put all of this into a new subroutine called `fitterTransit`, which took two arguments: `nPds` and `freq`. Then it was a simple step to loop through running `fitterTransit` 100 times for each `nPds` = 3, 6, 9, ... 30 with `freq` = 1 and again with `freq` = 2. This translates to observing every 1 hour for 180 hours around each of the two yearly peaks of the radial velocity curve for 3, 6, 9, ... 30 years, compared to observing every 2 hours for 360 hours around each peak of the radial velocity curve for the same number of years.

3 Results

3.1 Graphs

Figure 12 illustrates the difference in the relationship between mass error and number of observations across the several cases I investigated. As discussed in the previous section, when the high-false-negative trials are removed, the period fit (blue) and period known (red) cases match exactly; similarly, the peak-observing cases with `freq` = 1 (yellow) and `freq` = 2 (green) match very closely. However, the peak-observing case has a significantly lower error ($\sim 5\%$), amounting to a 20% more accurate mass.

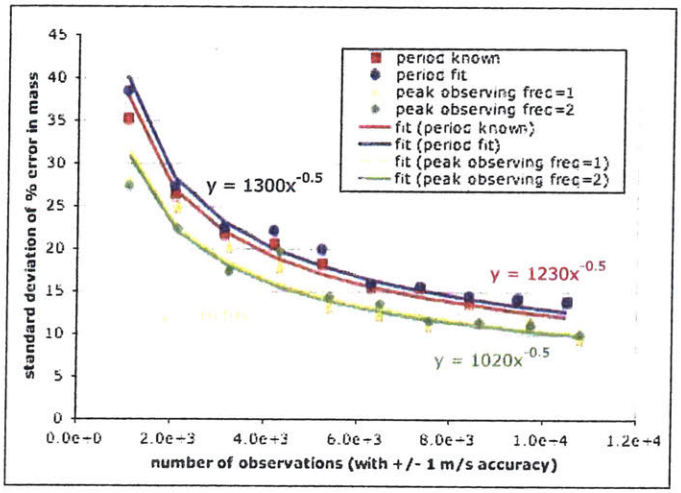


Figure 12: Comparison of the mass-error relationship with N for the different period conditions

Shown are the power-law fit curves for the different period conditions for 3, 6, 9, ... 30 periods of data. As previously shown in Figure 8, the period-fit (blue) and period-known (red) equally-spaced-observations curves are essentially equal when the divergence is fully removed. The peak-observing curves are also basically equal whether observations are made every hour (yellow) or every two hours (green). However, the peak-observing fits give a significantly lower mass error than the equally-spaced fits for the same number of observations, showing that the choice to observe close to the peak improves the accuracy of the mass.

This improved accuracy is given even though SNR is unchanged between the evenly-spaced cases and the peak-observing cases. A graph of the error's relationship with the SNR across the four cases is shown in Figure 13.

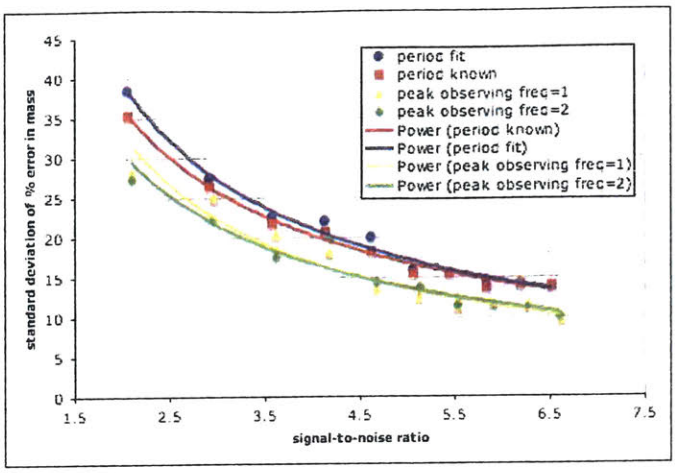


Figure 13: Comparison of error and SNR for the different period cases
The reduction of mass error from the equally-spaced case to the peak-observing case occurs even though SNR remains the same for each `nPds`.

4 Discussion and Conclusions

4.1 Discussion of results

Figures 12 and 13 make it clear that the peak-observing method has allowed the mass of the planet to be calculated more accurately than when the observations were evenly-spaced, even though SNR was unchanged. By allowing a more accurate mass to be calculated for the same SNR simply by choosing the timing of observations differently, the peak-observation method could be an extremely useful tool for exo-planetary scientists looking to lower the bound for low-mass planets detectable with radial velocity. When observations are made very frequently around the peaks of the radial velocity curve, the mass can be detected 20% more accurately than when they are evenly-spaced throughout the period.

In concrete terms, the result shown above simulated observing a star with an Earth-twin-candidate exoplanet – that is, a planet of Earth's mass in a circular orbit about a Sun-mass star at a radius of 1 AU. Taking radial velocity measurements with 1 $^{m}/_{s}$ accuracy every hour for two weeks six months apart (perhaps one week in June and one week in December) and continuing this for about 7 years would yield a mass measurement with an error of about 20%. Meanwhile, using those same 7 years to take one radial velocity measurement every 25 hours would yield about a 25% mass error.

4.2 Concluding thoughts

Observing every hour for a full week is certainly somewhat unrealistic and very idealized. I should note that I am not a stranger to the difficulties of observational astronomy, and realize that in order to put something like the peak-observation method into practice would require vast amounts of resources, extremely good luck with weather, and large blocks of time on several telescopes around the world (or,

perhaps, a radial-velocity-observing space telescope). However, this result could be extrapolated by observers or future researchers to a more realistic compromise scenario of observing which would likely produce an error intermediate between the peak-observing and equally-spaced cases. Additionally, the code and simulation techniques developed for this project may in themselves be a useful starting point for future research into increasing radial velocity accuracy in the mass measurement.

It is surprising to most, including myself, when they first learn just how difficult it would be to accurately detect an Earth-twin, and this project serves as a further demonstration of this fact; this also makes the recent discovery of low-mass planets like GJ 581 e even more exciting and impressive.

However difficult it is, though, the accurate radial velocity detection of Earth-mass exoplanets would be a huge milestone in our knowledge and understanding of the universe. Because of how groundbreaking such a result would be, any methods that have a possibility of increasing the likelihood of such a detection must be investigated. With the new Kepler telescope, which has just seen first light in the last month, the lower mass limit on detectable transiting planets is likely to change quickly in the near future, and perhaps we will soon have Earth-twin-candidate exoplanets to observe.

Dedication and acknowledgements

I would like to dedicate this thesis to the memory of my mother, Patricia Moberger, who passed away in October 2008. Though she did not get to see my work on this project, I know that she would have found it fascinating, and would have been thrilled to see me learning the same programming language she once used in her work.

I am extremely grateful to my thesis project advisor, Professor Sara Seager, for her support, help, patience, and the wealth of background knowledge on exoplanets she provided me. Jane Connor, the MIT EAPS writing tutor, was also indispensible to me, especially in preparation for the project and during the revision process, but her general support was just as much appreciated. I would also like to give my deepest thanks to Wei-Han Bobby Liu for the time and effort he spent helping me learn, write, and debug C and Excel code, as well as his endless patience in introducing and explaining the simplex method to me. I also want to acknowledge my academic advisor, Professor Rick Binzel, who provided me with the initial impetus to join the EAPS department and study planetary science, and provided me with support throughout my academic career.

Last but far from least, I would like to thank my family. If my brother and nana had not allowed me to miss their birthday celebrations, this project might not have been finished. And of course, without my father's constant encouragement and dedication, this project could never have happened at all.

Appendices

A Derivations

Derivations of the equations referenced in the introduction are provided as a supplement to the curious reader. The transit equations are not used in this paper but may be useful for future research, as the planet's radius and period can be determined from the duration and depth of its transit of a star with a known radius.

A.1 Radial velocity semi-amplitude

Because a planet does not orbit a star, but rather both orbit the barycenter of the system (or common center of mass), both the star and the planet have some distance from this point as well as some velocity about it. The star's orbital radius, a_*, and the planet's orbital radius, a_p, are governed by the equation of force balance, which in its simplest form states

$$m_* a_* = m_p a_p \tag{6}$$

The distance of the planet from the star is

$$a = a_* + a_p \tag{7}$$

as seen in Figure 14, which marks the barycenter of the system with the X.

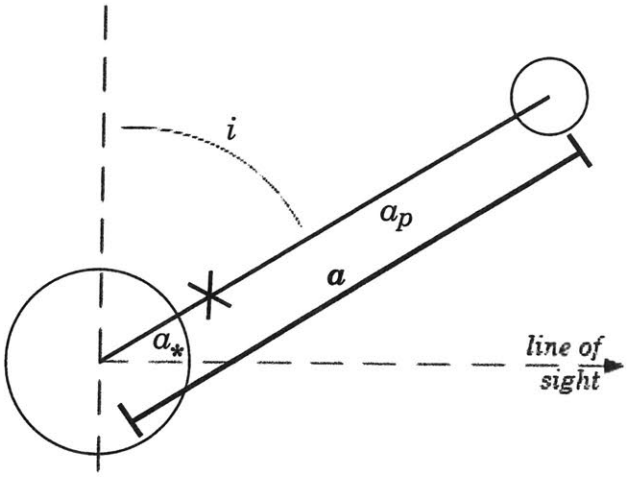


Figure 14: A star and planet orbit their common center of mass

We also assume that we are viewing the system in Figure 14 along the line of sight in the diagram, meaning the vertical dashed line represents the plane of the sky and the angle this plane makes with the plane of the orbit is the inclination i.

From (7) we derive that $a_p = \frac{m_* a_*}{m_p}$, and substituting this into (7) and simplifying gives

$$a = a_* \left(\frac{m_p + m_*}{m_p} \right) \tag{8}$$

Now we look to Kepler's third law, which states $\frac{a^3}{m_*+m_p} = \frac{P^2G}{4\pi^2}$. Replacing a with the expression in (8) gives $\frac{a_*^3\left(\frac{m_p+m_*}{m_p}\right)^3}{m_p^3} = \frac{P^2G}{4\pi^2}$, which simplifies to

$$\frac{a_*^3\left(m_p+m_*\right)^2}{m_p^3} = \frac{P^2G}{4\pi^2} \tag{9}$$

We observe that for a circular orbit, the orbital velocity of the star about the barycenter is given by a simple law of circular motion, $K_* = \frac{2\pi a_*}{P}$. When (9) is rearranged and substituted into this, we get

$$K_* = \frac{2\pi m_p}{P}\left[\frac{P^2G}{4\pi^2\left(m_p+m_*\right)^2}\right]^{1/3} \tag{10}$$

Simplifying (10) gives

$$K_* = \frac{m_p}{\left(m_p+m_*\right)^{2/3}}\left(\frac{2\pi G}{P}\right)^{1/3} \tag{11}$$

Recall that (11) is the orbital velocity of the star about the barycenter. However, when we observe this velocity to determine the planet's mass, we can only observe the velocity along the line of sight using Doppler spectroscopy. This is shown in Figure 15, from which it is clear that the line of sight velocity $K = K_* \sin i$.

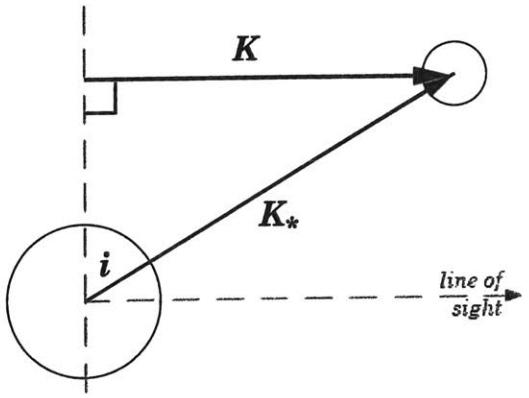


Figure 15: Line of sight velocity relative to total velocity

Therefore $K = \frac{m_p \sin i}{\left(m_p+m_*\right)^{2/3}}\left(\frac{2\pi G}{P}\right)^{1/3}$, which for $m_p \ll m_*$ simplifies to give equation (1):

$$K = \frac{m_p \sin i}{m_*^{2/3}}\left(\frac{2\pi G}{P}\right)^{1/3}$$

A.2 Transit duration

Imagine a planet transiting a star.

First contact is the time at which the limb of the planet and the limb of the star first appear to touch and ingress begins. The transit continues until egress ends at fourth contact, which is when the limb of the planet appears to last touch the limb of the star, just before the two separate and no longer appear to intersect. In order to determine the total transit duration, which corresponds to the total width of the observed transit light curve, we must relate the star and planet size and distances to the distance the planet travels to cross the star's disk, and then determine what fraction of the planet's orbital period the planet takes to traverse this distance.

First we examine the geometry of the transit event in Figure 16, which shows the planet at first contact and fourth contact with the star in the background (with planet and star not to scale).

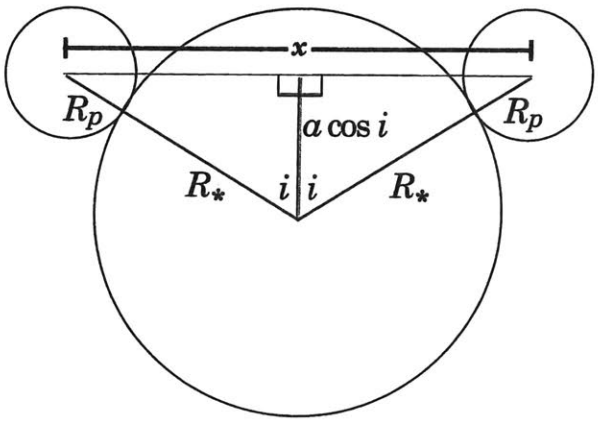


Figure 16: Schematic of a transit from first contact to fourth contact

Using the Pythagorean theorem in this figure, it is clear that the total distance that the planet travels to cross the star's disk (represented by the top horizontal line labeled x) is $x = 2\sqrt{(R_* + R_p)^2 - (a\cos i)^2}$. Multiplying through by $\frac{R_*}{R_*}$, this becomes $x = 2R_*\sqrt{\left(\frac{R_*+R_p}{R_*}\right)^2 - \left(\frac{a\cos i}{R_*}\right)^2}$, which simplifies to

$$x = 2R_*\sqrt{\left(1 + \frac{R_p}{R_*}\right)^2 - \left(\frac{a\cos i}{R_*}\right)^2} \tag{12}$$

Next we examine the geometry of the planet's orbit in Figure 17.

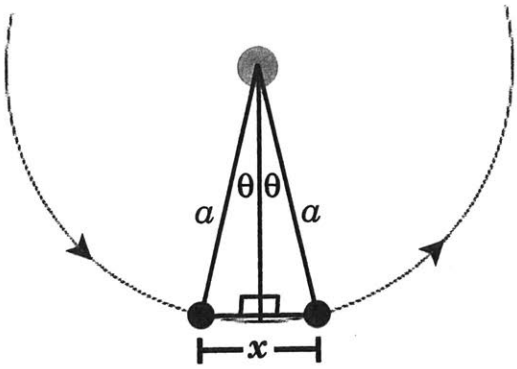


Figure 17: Overhead view of transiting planet's orbit

Here we see that $\sin\theta = \frac{x}{2a}$, which for small θ (the duration of the transit is small compared to the planet's period, which is true because the planets and stars are so far away) reduces to $\theta = \frac{x}{2a}$.

We also know that $\frac{2\theta}{2\pi} = \frac{t_T}{P}$ in a circular orbit, because the orbital velocity is constant, so the distances and times must be proportional. Solving for t_T we find that $t_T = \frac{P\theta}{\pi}$, and when we substitute in our equation for θ,

$$t_T = \frac{Px}{2\pi a} \tag{13}$$

Combining equations (12) and (13), we have derived (2):

$$t_T = \frac{PR_*}{\pi a}\sqrt{\left(1+\frac{R_p}{R_*}\right)^2 - \left(\frac{a\cos i}{R_*}\right)^2}$$

A.3 Transit depth

The transit depth is the maximum normalized change in flux from the star. As flux is simply power times unit area, the flux from the star when no transit is occurring is a function of the cross-sectional area of the star: $F_* = k\pi R_*^2$, where k is related to the total power output of the star. The amount of flux from the star which is blocked by the planet is, similarly, $F_p = k\pi R_p^2$.

The flux from the star when a transit is occurring is $F_* - F_p = k\pi R_*^2 - \pi R_p^2$. The change in flux between no transit and a transit is then $\Delta F = F_* - (F_* - F_p) = F_p$. Normalizing this (so that when no transit is occurring the flux is 1) gives $\Delta F = \frac{F_p}{F_*}$. Substituting gives $\Delta F = \frac{k\pi R_p^2}{k\pi R_*^2}$, which easily reduces to give equation (3):

$$\Delta F = \left(\frac{R_p}{R_*}\right)^2$$

B Full code

```
/* Declare Header Files */
 #include <stdio.h>
 #include <math.h>
 #include <stdlib.h>

/* Declare constants */
 #define G 6.67e-11  /* Universal Gravitational Constant in MKS */
 #define Ghr (G*3600*3600) /* gravitational constant in m^3/kg hr^2 */
 #define pi 3.14159
 #define mSun 2e30 /* mass of sun in kg */

/* Declare functions */
double calcLeastSquares(double mCalc, double pCalc,
    double time[], double data[], double maxIndex);
void fitter(double nPds, double pMin, double pMax);
void fitterTransit(double nPds, double freq);

/* Declare static variables */
static double results[14];

main() {
  double noiseFactor;
  double pMinFactor;
  double pMaxFactor;
  double nPds;
  double freq;
  int i,j;

  printf("period fitted\n");
  printf("nPds pMinFactor pMaxFactor mEarth pEarth minErrorSquared mBest
      pBest errorSquared stddev SNR percMassError percPdError abs(massError)\n");
  for(nPds=3; nPds<31; nPds+=3) {
    for(j=0;j<100;j++) {
      fprintf(stderr, "period fit: nPds=%e, j=%d\n", nPds, j);
      fitter(nPds, 0.5, 5);
      for(i=0;i<14;i++){
        printf("%e ",results[i]);
      }
      printf("\n");
    }
    printf("\n");
  }

  printf("\n\n\n");
  printf("period given\n");
  printf("nPds pMinFactor pMaxFactor mEarth pEarth minErrorSquared mBest
      pBest errorSquared stddev SNR percMassError percPdError abs(massError)\n");
  for(nPds=3; nPds<31; nPds+=3){
    for(j=0;j<100;j++){
      fprintf(stderr, "period given: nPds=%e, j=%d\n", nPds, j);
      fitter(nPds, 1,1);
      for(i=0;i<14;i++){
        printf("%e ",results[i]);
      }
      printf("\n");
    }
    printf("\n");
  }

  printf("\n\n\n");
  freq=2.;
```

```
  printf("period from transit: obs every %e hr\n", freq);
  printf("nPds pMinFactor pMaxFactor mEarth pEarth minErrorSquared mBest
      pBest errorSquared stddev SNR percMassError percPdError abs(massError)\n");

  for(nPds=3; nPds<31; nPds+=3){
    for(j=0;j<100;j++){
      fprintf(stderr, "period from transit: freq=%e, nPds=%e, j=%d\n", freq, nPds, j);
      fitterTransit(nPds,2);
      for(i=0;i<14;i++){
        printf("%e ",results[i]);
      }
      printf("\n");
    }
    printf("\n");
  }

  printf("\n\n\n");
  freq=1.;

  printf("period from transit: obs every %e hr\n", freq);
  printf("nPds pMinFactor pMaxFactor mEarth pEarth minErrorSquared mBest
      pBest errorSquared stddev SNR percMassError percPdError abs(massError)\n");
  for(nPds=3; nPds<31; nPds+=3){
    for(j=0;j<100;j++){
      fprintf(stderr, "period from transit: freq=%e, nPds=%e, j=%d\n", freq, nPds, j);
      fitterTransit(nPds,1);
      for(i=0;i<14;i++){
        printf("%e ",results[i]);
      }
      printf("\n");
    }
    printf("\n");
  }
}



void fitter(double nPds, double pMinFactor, double pMaxFactor){
  /****** Declare variables ******/
  // for creating data
  int i, j, index, maxIndex;
  double mEarth = 6e24;        // kg
  double pEarth = 365*24;      // hrs
  double tobs[35000];          // hrs
  double Vobs[35000];          // m/s
  double Vtest[35000];         // m/s
  double Vnoise[35000];
  double K;
  double noise;
  double noiseFactor=1*pow(1.2, 0.5);
  double totalNoise = 0;
  double avgNoise;

  // for simplex, fitting
  double mMin, mMax, pMin, pMax;
  double best[3],mid[3],worst[3],D[3],E[3],cg[2],order[3],temp[3];
  double mBest, pBest, xBest, stddev, SNR;
  double Vfit[35000];
  double errorSquared=0;
  double minErrorSquared;
```

```
/*****************************************************************************/
/******              create data points, add noise                          ******/
/*****************************************************************************/

  // semi-amplitude for system
  K = (1./3600) * (mEarth/pow(mSun,2./3))*pow((2*pi*Ghr/pEarth),(1./3));
  index=0;
  for (i=1; i<=pEarth*nPds; i+=25) {
    tobs[index] = i;                //hours
    Vtest[index] = K*sin((2*pi/pEarth) * i); // m/s
    noise=0;

    // for each point, calculate Gaussian noise value to add
    for (j=1; j<= 10; j++) {// first sum up 10 random integers from -5 to 5
      noise+=((rand()%11 - 5)*noiseFactor);
    }
    Vnoise[index] = noise/j;    // then average by dividing by j
    Vobs[index]=Vtest[index]+Vnoise[index];  // now add noise to data array
    totalNoise +=Vnoise[index];

    index++;
  }
  maxIndex=index; //number of data points
  avgNoise = totalNoise/maxIndex;

/*****************************************************************************/
/****** institute downhill simplex method for finding minimum errorSquared. ******/
/*****************************************************************************/

  mMin = 0.5 * mEarth;
  mMax = 5. * mEarth;
  pMin = pMinFactor * pEarth;
  pMax = pMaxFactor * pEarth;
  double mIncrement = (mMax-mMin)/9.;
  double pIncrement = (pMax-pMin)/9.;

  if ((pMinFactor == pMaxFactor)&&(pMaxFactor==1)){ //period given case
    mBest = mMin;
    pBest = pMin;
    mIncrement = (mMax-mMin)/99.;
    xBest = calcLeastSquares(mMin,pMin,tobs, Vobs,maxIndex);
    for (i=0; i<100; i++){
      double x = calcLeastSquares (mMin+i*mIncrement, pMin, tobs, Vobs, maxIndex);
      if (x<xBest){
        mBest = mMin + i*mIncrement;
        xBest = x;
      }
    }
    double mOldMin = mMin;
    mMin = (mBest==mOldMin?mBest:mBest-mIncrement);
    mMax = (mBest==mMax?mBest:mBest+mIncrement);
    pMin = pBest;
    pMax = pBest;
  }
  else {  //period fit case
    mBest = mMin;
    pBest = pMin;
    xBest = calcLeastSquares(mMin,pMin,tobs, Vobs,maxIndex);
    for (i=0; i<10; i++){
      for (j=0; j<10; j++){
        double x = calcLeastSquares (mMin+i*mIncrement, pMin + j*pIncrement,
            tobs, Vobs, maxIndex);
        if (x<xBest){
```

```
        mBest = mMin + i*mIncrement;
        pBest = pMin + j*pIncrement;
        xBest = x;
      }
    }
  }
  mMin = 0.95*mBest;
  mMax = 1.05*mBest;
  pMin = 0.95*pBest;
  pMax = 1.05*pBest;
}

// initial 3 points for simplex
best[0]=mMin;
best[1]=pMin;
best[2]=calcLeastSquares(best[0],best[1],tobs,Vobs,maxIndex);
mid[0]=mMax;
mid[1]=pMin;
mid[2]=calcLeastSquares(mid[0],mid[1],tobs,Vobs,maxIndex);
worst[0]=mMax;
worst[1]=pMax;
worst[2]=calcLeastSquares(worst[0],worst[1],tobs,Vobs,maxIndex);


double tol=1e-4;
    // if values are within this fraction of each other, they are equal

int counter=0;       // number of loop iteration
int maxcounter = 500; // break out of loop if the program fails to converge

while(1) {    // infinite loop - must be broken out of by the values converging
  counter++;
  if (maxcounter<counter){
    fprintf(stderr, "This run did not converge.\n");
    break;
  }
  if (worst[2]<mid[2]){              // reorder the three values
    for (i=0; i<3; i++){              //    from smallest error (best)
      temp[i] = worst[i];              //    to largest error (worst)
      worst[i]=mid[i];
      mid[i]=temp[i];
      }
  }
  if (mid[2]<best[2]){
    for (i=0; i<3; i++){
      temp[i] = mid[i];
      mid[i]=best[i];
      best[i]=temp[i];
      }
  }
  if (worst[2]<mid[2]){
    for (i=0; i<3; i++){
      temp[i] = worst[i];
      worst[i]=mid[i];
      mid[i]=temp[i];
    }
  }

  // if the vertices of the triangle are close enough, break out of loop
  if ((2*fabs(best[0]-mid[0])/(best[0]+mid[0])<tol) &&
      (2*fabs(best[1]-mid[1])/(best[1]+mid[1])<tol) &&
      (2*fabs(best[0]-worst[0])/(best[0]+worst[0])<tol) &&
      (2*fabs(best[1]-worst[1])/(best[1]+worst[1])<tol) &&
      (2*fabs(worst[0]-mid[0])/(worst[0]+mid[0])<tol) &&
```

```
      (2*fabs(worst[1]-mid[1])/(worst[1]+mid[1])<tol)) {
    break;
  }

  cg[0] = (best[0]+mid[0])/2;          // implement Nelder-Mead simplex method
  cg[1] = (best[1]+mid[1])/2;

  D[0] = fabs(cg[0] + (1)*(cg[0] - worst[0]));
  D[1] = fabs(cg[1] + (1)*(cg[1] - worst[1]));
  D[2] = calcLeastSquares(D[0], D[1], tobs,Vobs, maxIndex);

  if (D[2]<best[2]){                // expansion case
    E[0] = cg[0] + (2)*(cg[0]-worst[0]);
    E[1] = cg[1] + (2)*(cg[1]-worst[1]);
    E[2] =   calcLeastSquares(E[0], E[1], tobs,Vobs, maxIndex);
    if (E[2]<D[2]){
      for (i=0; i<3; i++){
       worst[i] = E[i];
      }
      continue;
    }
    else {
      for (i=0; i<3; i++){
        worst[i] = D[i];
      }
      continue;
    }
  }
  else {
    if (D[2]>worst[2]){              // contraction case
      E[0] = worst[0] + (0.5)*(cg[0]-worst[0]);
      E[1] = worst[1] + (0.5)*(cg[1]-worst[1]);
      E[2] = calcLeastSquares(E[0], E[1], tobs,Vobs, maxIndex);
      if (E[2]<worst[2]){
        for (i=0; i<3; i++){
          worst[i] = E[i];
        }
        continue;
      }
      else {
        mid[0] = best[0] + 0.5*(mid[0]-best[0]);
        mid[1] = best[1] + 0.5*(mid[1]-best[1]);
        mid[2] = calcLeastSquares(mid[0], mid[1], tobs,Vobs, maxIndex);
        worst[0] = best[0] + 0.5*(worst[0]-best[0]);
        worst[1] = best[1] + 0.5*(worst[1]-best[1]);
        worst[2] = calcLeastSquares(worst[0], worst[1], tobs,Vobs, maxIndex);
        continue;
      }
    }
    else {                      // reflection case
      for (i=0; i<3; i++){
        worst[i] = D[i];
      }
      continue;
    }
  }
} // end while loop

mBest=best[0];
pBest=best[1];
xBest=best[2];
```

```
/*****************************************************************************/
/******       calculate all values in results array                         ******/
/*****************************************************************************/

  // calculate theoretical "minimum" from conditions that created the data points
  minErrorSquared = calcLeastSquares(mEarth, pEarth, tobs,Vobs, maxIndex);

  // calculate standard deviation of noise
  stddev=0;
  for (i=0; i<maxIndex; i++){
    stddev+=pow((Vnoise[i]-avgNoise),2);
  }
  stddev=pow((stddev/maxIndex),0.5);

  // calculate SNR
  SNR = pow(maxIndex,0.5)*(K/(pow(2,0.5)*stddev));

  // create points in model
  index=0;
  for (i=1; index<maxIndex; i+=100) {
    Vfit[index] =
        (1./3600) * (mBest/pow(mSun,2./3))
            * pow((2*pi*Ghr/pBest),(1./3))*sin((2*pi/pBest) * i); // m/s
    index++;
  }

  // return results array
  results[0]=nPds;
  results[1]=pMinFactor;
  results[2]=pMaxFactor;
  results[3]=mEarth;
  results[4]=pEarth;
  results[5]=minErrorSquared;

  results[6]=mBest;
  results[7]=pBest;
  results[8]=xBest;

  results[9]=stddev;
  results[10]=SNR;
  results[11]=100*((mBest/mEarth)-1);
  results[12]=100*((pBest/pEarth)-1);
  results[13]=fabs(results[11]);
}

void fitterTransit(double nPds, double freq){
  int i, j, k, index, maxIndex;
  double mEarth = 6e24;          // kg
  double pEarth = 365*24;        // hrs
  double tTransit[35000]; // hrs
  double Vtransit[35000]; // m/s
  double VtransitTest[35000]; // m/s
  double Vnoise[35000];
  double K;
  double noise;
  double noiseFactor=1*pow(1.2, 0.5);
  double totalNoise = 0;
  double avgNoise;
  // for simplex, fitting
  double mMin, mMax, pMin, pMax;
  double best[3],mid[3],worst[3],D[3],E[3],cg[2],order[3],temp[3];
  double mBest, pBest, xBest, stddev, SNR;
  double Vfit[35000];
  double errorSquared=0;
  double minErrorSquared;
```

```
/*****************************************************************************/
/******        create data points, add noise                            ******/
/*****************************************************************************/

  // semi-amplitude for system
  K = (1./3600) * (mEarth/pow(mSun,2./3))*pow((2*pi*Ghr/pEarth),(1./3));
  index=0;
  for (i=(pEarth/4)-(90*freq); i<pEarth*nPds; i+=(pEarth/2)){  // skip to 90 hrs before peak
    for (j=i; j<i+180*freq; j+=freq){  // observe every <freq> hrs until 90*freq hrs after peak
      tTransit[index]=j;  // hours
      VtransitTest[index] = K*sin((2*pi/pEarth) * j); // m/s
      noise=0;
      // for each point, calculate Gaussian noise value to add
      for (k=1; k<= 10; k++) {// first sum up 10 random integers from -5 to 5
        noise+=((rand()%11 - 5)*noiseFactor);
      }
      Vnoise[index] = noise/k;    // then average by dividing by j
      Vtransit[index]=VtransitTest[index]+Vnoise[index];  // now add noise to data array
      totalNoise +=Vnoise[index];
      index++;
    }
  }
  maxIndex=index; //number of data points
  avgNoise = totalNoise/maxIndex;

/*****************************************************************************/
/****** institute downhill simplex method for finding minimum errorSquared. ******/
/*****************************************************************************/

  mMin = 0.5 * mEarth;
  mMax = 5. * mEarth;
  pMin = pEarth;
  pMax = pEarth;
  double mIncrement = (mMax-mMin)/9.;
  double pIncrement = (pMax-pMin)/9.;


  //period given case
  mBest = mMin;
  pBest = pMin;
  mIncrement = (mMax-mMin)/99.;
  xBest = calcLeastSquares(mMin,pMin,tTransit, Vtransit,maxIndex);
  for (i=0; i<100; i++){
    double x = calcLeastSquares (mMin+i*mIncrement, pMin, tTransit, Vtransit, maxIndex);
    if (x<xBest){
      mBest = mMin + i*mIncrement;
      xBest = x;
    }
  }
  double mOldMin = mMin;
  mMin = (mBest==mOldMin?mBest:mBest-mIncrement);
  mMax = (mBest==mMax?mBest:mBest+mIncrement);
  pMin = pBest;
  pMax = pBest;



  //initial 3 points for simplex
  best[0]=mMin;
  best[1]=pMin;
  best[2]=calcLeastSquares(best[0],best[1],tTransit,Vtransit,maxIndex);
  mid[0]=mMax;
  mid[1]=pMin;
  mid[2]=calcLeastSquares(mid[0],mid[1],tTransit,Vtransit,maxIndex);
```

```
worst[0]=mMax;
worst[1]=pMax;
worst[2]=calcLeastSquares(worst[0],worst[1],tTransit,Vtransit,maxIndex);

double tol=1e-4;
    // if values are within this fraction of each other, they are equal

int counter=0;       //number of loop iteration
int maxcounter = 500; //break out of loop if the program fails to converge

while(1) {    // infinite loop - must be broken out of by the values converging
  counter++;
  if (maxcounter<counter){
    fprintf(stderr, "This run did not converge.\n");
    break;
  }
  if (worst[2]<mid[2]){              // reorder the three values
    for (i=0; i<3; i++){              //    from smallest error (best)
      temp[i] = worst[i];              //    to largest error (worst)
      worst[i]=mid[i];
      mid[i]=temp[i];
      }
  }
  if (mid[2]<best[2]){
    for (i=0; i<3; i++){
      temp[i] = mid[i];
      mid[i]=best[i];
      best[i]=temp[i];
      }
  }
  if (worst[2]<mid[2]){
    for (i=0; i<3; i++){
      temp[i] = worst[i];
      worst[i]=mid[i];
      mid[i]=temp[i];
    }
  }

  // if the vertices of the triangle are close enough, break out of loop
  if ((2*fabs(best[0]-mid[0])/(best[0]+mid[0])<tol) &&
    (2*fabs(best[1]-mid[1])/(best[1]+mid[1])<tol) &&
    (2*fabs(best[0]-worst[0])/(best[0]+worst[0])<tol) &&
    (2*fabs(best[1]-worst[1])/(best[1]+worst[1])<tol) &&
    (2*fabs(worst[0]-mid[0])/(worst[0]+mid[0])<tol) &&
    (2*fabs(worst[1]-mid[1])/(worst[1]+mid[1])<tol)) {
    break;
  }

  cg[0] = (best[0]+mid[0])/2;          // implement Nelder-Mead simplex method
  cg[1] = (best[1]+mid[1])/2;

  D[0] = fabs(cg[0] + (1)*(cg[0] - worst[0]));
  D[1] = fabs(cg[1] + (1)*(cg[1] - worst[1]));
  D[2] = calcLeastSquares(D[0], D[1], tTransit,Vtransit, maxIndex);

  if (D[2]<best[2]){               // expansion case
    E[0] = cg[0] + (2)*(cg[0]-worst[0]);
    E[1] = cg[1] + (2)*(cg[1]-worst[1]);
    E[2] =   calcLeastSquares(E[0], E[1], tTransit,Vtransit, maxIndex);
    if (E[2]<D[2]){
      for (i=0; i<3; i++){
        worst[i] = E[i];
      }
      continue;
    }
```

```
      else {
        for (i=0; i<3; i++){
          worst[i] = D[i];
        }
        continue;
      }
    }
    else {
      if (D[2]>worst[2]){              // contraction case
        E[0] = worst[0] + (0.5)*(cg[0]-worst[0]);
        E[1] = worst[1] + (0.5)*(cg[1]-worst[1]);
        E[2] =   calcLeastSquares(E[0], E[1], tTransit,Vtransit, maxIndex);
        if (E[2]<worst[2]){
          for (i=0; i<3; i++){
            worst[i] = E[i];
          }
          continue;
        }
        else {
          mid[0] = best[0] + 0.5*(mid[0]-best[0]);
          mid[1] = best[1] + 0.5*(mid[1]-best[1]);
          mid[2] = calcLeastSquares(mid[0], mid[1], tTransit,Vtransit, maxIndex);
          worst[0] = best[0] + 0.5*(worst[0]-best[0]);
          worst[1] = best[1] + 0.5*(worst[1]-best[1]);
          worst[2] = calcLeastSquares(worst[0], worst[1], tTransit,Vtransit, maxIndex);
          continue;
          }
        }
      else {                      // reflection case
        for (i=0; i<3; i++){
          worst[i] = D[i];
        }
        continue;
      }
    }
  } // end while loop

  mBest=best[0];
  pBest=best[1];
  xBest=best[2];


/*****************************************************************************/
/******       calculate all values in results array                  ******/
/*****************************************************************************/

  // calculate theoretical "minimum" from conditions that created the data points
  minErrorSquared = calcLeastSquares(mEarth, pEarth, tTransit,Vtransit, maxIndex);

  // calculate standard deviation of noise
  stddev=0;
  for (i=0; i<maxIndex; i++){
     stddev+=pow((Vnoise[i]-avgNoise),2);
  }
  stddev=pow((stddev/maxIndex),0.5);

  // calculate SNR
  SNR = pow(maxIndex,0.5)*(K/(pow(2,0.5)*stddev));

  // create points in model
  index=0;
  for (i=1; index<maxIndex; i+=100) {
    Vfit[index] =
      (1./3600) * (mBest/pow(mSun,2./3))
```

```
          * pow((2*pi*Ghr/pBest),(1./3))*sin((2*pi/pBest) * i); // m/s
    index++;
  }
  // return results array
  results[0]=nPds;
  results[1]=1;
  results[2]=1;
  results[3]=mEarth;
  results[4]=pEarth;
  results[5]=minErrorSquared;

  results[6]=mBest;
  results[7]=pBest;
  results[8]=xBest;

  results[9]=stddev;
  results[10]=SNR;
  results[11]=100*((mBest/mEarth)-1);
  results[12]=100*((pBest/pEarth)-1);
  results[13]=fabs(results[11]);
}
```

```
/*****************************************************************************/
/******         define least-squares calculation subroutine                 ******/
/*****************************************************************************/

double calcLeastSquares(double mCalc, double pCalc, double time[], double data[],
    double maxIndex){

  /* Declare variables */
  int index;
  double i;
  double Vfit[35000];
  double errorSquared=0;
  index = 0;

  for (index=0; index<maxIndex; index++){
    i = time[index];
    Vfit[index] =
        (1./3600)*(mCalc/pow(mSun,2./3))
            *  pow((2*pi*Ghr/pCalc),(1./3))*sin((2*pi/pCalc) * i); // m/s
    errorSquared +=  pow((data[index] - Vfit[index]), 2);
  }
  return errorSquared;
}
```

References

[ESO 2003] "HARPS: The Planet Hunter," ESO. Updated October 2003. http://www.eso.org/lasilla/instruments/harps/overview.html

[Forveille et al. 2009] Forveille, T., Bonfils, X., Delfosse, X., Gillon, M., Udry, S., Bouchy, F., Lovis, C., Mayor, M., Pepe, F., Perrier, C., Queloz, D., Santos, N., & Bertaux, J.-L. "The HARPS search for southern extra-solar planets: XIV. Gl 176b, a super-Earth rather than a Neptune, and at a different period," Astronomy and Astrophysics, Volume 493, Issue 2, pp. 645-650. January 2009. http://lanl.arxiv.org/abs/0809.0750v1

[Gaudi & Winn 2007] Gaudi, B.S. and Winn, J.N. "Prospects for the Characterization and Confirmation of Transiting Exoplanets via the Rossiter-McLaughlin Effect," The Astrophysical Journal, Volume 655, pp. 550-563. 20 January 2007. http://www.iop.org/EJ/article/0004-637X/655/1/550/65965.web.pdf

[Mayor et al. 2009a, preprint] Mayor, M., Bonfils, X., Forveille, T., Delfosse, X., Udry, S., Bertaux, J.-L., Beust, H., Bouchy, F., Lovis, C., Pepe, F., Perrier, C., Queloz, D., & Santos, N.C. "The HARPS search for southern extra-solar planets: XVIII. An Earth-mass planet in the GJ 581 planetary system," Astronomy and Astrophysics, 21 April 2009, in press. http://obswww.unige.ch/~udry/Gl581_preprint.pdf

[Mayor et al. 2009b] Mayor, M., Udry, S., Lovis, C., Pepe, F., Queloz, D., Benz, W., Bertaux, J.-L., Bouchy, F., Mordasini, C., & Segransan, D. "The HARPS search for southern extra-solar planets. XIII. A planetary system with 3 super-Earths (4.2, 6.9, and 9.2 M_{Earth})," Astronomy and Astrophysics, Volume 493, Issue 2, pp. 639-644. January 2009. http://adsabs.harvard.edu/abs/2009A%26A...493..639M

[NASA 2008] "Kepler Mission: A search for habitable planets," NASA. Updated 24 July 2008. http://kepler.nasa.gov/about/

[Rouan et al. 2009, in preparation] Rouan, D., Leger, A., Schneider, J., Alonso, R., Samuel, B., Guenther, E., Deleuil, M., Deeg, H.J., Fridlund, M., et al. "Has CoRoT discovered the first transiting super-earth around a main sequence star?," 2009, in preparation. Referenced at http://exoplanet.eu/planet.php?p1=CoRoT-7&p2=b#a_publi

[Schneider 2009] Schneider, J. "Interactive Extra-solar Planets Catalog," The Extrasolar Planets Encyclopedia. CNRS-LUTH: Paris Observatory. Version 2.02, updated 21 April 2009. http://exoplanet.eu/catalog.php

Index