Draft version April 30, 2022
Preprint typeset using LaTeX style emulateapj v. 12/16/11

A FAST CODE FOR TRANSIT-TIMING USING REBOUND

Ethan Hargrove & Hanno Rein
University of Toronto, Toronto, Ontario, Canada
Draft version April 30, 2022

ABSTRACT

Fitting a model planetary system to a star's observed transit-timing data is an effective way to determine the parameters of the observed system. The fitting process requires determining the transit-times for many model systems; this can be time-consuming. Using the multi-purpose N-body code REBOUND (Rein & Liu 2012), we examined the speed and accuracy of code for computing transit-times. This testing was done using the WHFast, IAS15, and Bulirsch-Stoer integrators with Newton-Raphson, Bisection, Ridders', Kou-Wang-Ostrowski, and Brent's root-finding algorithms. Also tested were two root-bracketing methods: the default, which checks each planet for a transit after every timestep, and fast-mode, which places a wide bracket around analytically computed Keplerian transit-times. Results were compared with an existing N-body code, NbodyGradient.jl (Agol, Hernandez, & Langford 2021). The most favourable integrator was WHFast, with runtimes that were approximately one-eighth of NbodyGradient's runtimes and approximately one-third of the runtimes using IAS15 and Bulirsch-Stoer for systems with five or more planets. When using a low root-finding tolerance or the WHFast integrator, the Newton-Raphson method performed best, with Brent's method being only slightly slower. When using the IAS15 and Bulirsch-Stoer integrators with a more reasonable root-finding tolerance Kou-Wang-Ostrowski's was the fastest, however the differences in runtimes among root-finding algorithms were relatively small. The fast-mode of root-bracketing performed 2.5 times faster than the default with WHFast, however its usefulness is constrained to well-behaving systems. The fast runtime of the code shows its promise as a tool for fitting a model planetary system to observed transit-times.

1. INTRODUCTION

For the first time, in 1992, astronomers Aleksander Wolszczan and Dale Frail confirmed the existence of planets orbiting a star other than Earth. Such planets are referred to as exoplanets, and their study remains a significant field of research within astronomy, with new exoplanets confirmed regularly. When we discover and confirm the existence of a new exoplanet, there is a strong desire to classify it based on its mass, orbit, and composition. Grouping exoplanets with similar parameters allows us to broadly apply information obtained about one planet to the entire group, increasing the efficiency of each discovery. Most exoplanet detections come from the transit method, when we observe periodic decreases in the brightness of a star as a planet passes in front of it (transits). While this method is effective at detecting exoplanets, it must be used alongside another method to determine all parameters of the planetary system.

A recent increase in photometric precision has allowed us to detect changes in the period of planets over time. These changes are caused by gravitational interactions between planets and are referred to as transit-timing variations (TTVs). If we observe TTVs, then we can determine the parameters with the aid of planetary system simulations.

A system with more than two bodies does not undergo Keplerian motion because all bodies in the system are interacting gravitationally. Due to all these interactions we must use an N-body simulation for our modelling. REBOUND is a multi-purpose N-body code that integrates over the equations of motion to evolve the system forwards or backwards in time (Rein & Liu 2012). We will examine three of REBOUND's integrators. The first being WHFast, an implementation of the symplectic Wisdom-Holman integrator that works best when perturbations to the Keplerian orbits are small (Rein & Tamiyo 2015). The second being IAS15, a very high order, non-symplectic integrator (Rein & Speigel 2015). The third being Bulirsch-Stoer, an adaptive integrator that is best used for short integrations with medium accuracy (Hairer, Wanner, & Norsett 1993). Each of the integrators were utilized with the default setting in REBOUND and an initial timestep 1/20 the shortest orbital period in the system. Using REBOUND, we can calculate the transit-times of a simulated planetary system using root-finding methods. These transit-times can then be used to fit a model planetary system to the observed transit-times. The parameter fitting process will require calculating the transit-times for many simulated systems, so improving speed while maintaining accuracy is a top priority.

2. TRANSIT-TIMES OF SIMULATED SYSTEM

For our REBOUND planetary simulations our reference plane is the x-y plane and our reference direction is the positive x axis. When performing simulations using REBOUND, it is best to use the center-of-mass reference frame to decrease the numerical errors due to finite floating point precision as the center-of-mass and planets will slowly drift away from the origin. Setting the observer's line of sight as the reference direction tells us that a transit occurs when a planet crosses the positive x axis. In our center-of-mass frame a transit occurs when $y_{\mathrm{planet}} - y_{\mathrm{star}} = 0$ coming from below (see Figure 1). This subtraction comes from the need to consider the position of the planets in a heliocentric prospective for transit detection. Future references to the y-position will refer to $y_{\rm planet} - y_{\rm star}$ in the center-of-mass frame, and future references to the y-velocity will refer to $v_{y_{\rm planet}} - v_{y_{\rm star}}$ in the center-of-mass frame. The transit-times of our simulated planets are the roots of their y-position as a function of time as the function goes from negative to positive.

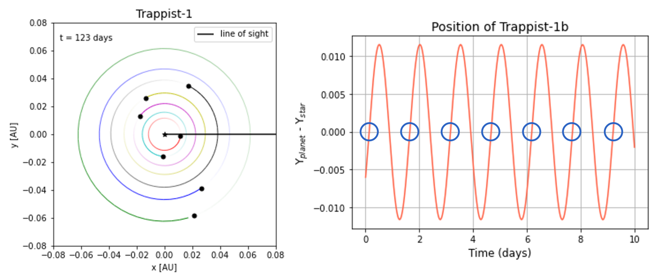

Figure 1. Left: Sample system Trappist-1 with planets orbiting counterclockwise in the x-y plane, periodically transiting the observer's line of sight. Right: The center-of-mass $y_{\rm planet} - y_{\rm star}$ of Trappist-1b as a function of time. Circled are the roots of the function that correspond to a transit.

3. BRACKETING THE ROOTS

Due to the periodic nature of planetary motion, we must carefully determine an interval where exactly one root exists. A simple and thorough way to go about this is to determine the position of each planet after every timestep, if a planet's previous y-position was negative and its current y-position is positive, then we know a transit has occurred at some point during this interval. This method will be referred to as default. It only works if the timestep used is less than half of the shortest orbital period in the system, else it is possible that a planet's previous and current y-position could have the same sign while a root exists between them.
Checking the y-position of each planet at every timestep can be computationally expensive, especially for the WHFast integrator, which utilizes smaller timesteps. For well-behaved systems we can avoid this by analytically determining the transit-times for each planet as though they were undergoing Keplerian motion. We can then create a large bracket around these transit-times and then use a root-finding algorithm to find each true transit-time. This works for most systems, however it will have issues when perturbations to the Keplerian orbits are very large. This method will be referred to as fast-mode. The derivation of the Keplerian transit-times can be found in Appendix A.

4. FINDING THE ROOTS

After bracketing a root we can converge upon it using a root-finding algorithm. There are two main types of root-finding algorithms, closed and open. Closed algorithms work by reducing the distance between the upper and lower bracket with each iteration, the error is the distance between the two brackets. Open algorithms start with an initial guess and each iteration provides a new guess that should be closer to the true value, the error is the distance between subsequent guesses. For open methods our initial guess will be the midpoint of the original bracket. When using open methods for periodic functions with many roots it is important that we find the root that we have bracketed and not some other root. If any iteration's guess is outside the original bracketed interval, we must to adjust the brackets before resuming with the open method, using the midpoint of the new bracketed interval as our initial guess. We know we have converged when the error is less than or equal to the uncertainty of our observed transit-times. A variable root-finding tolerance is a feature not found in existing transit-timing programs and helps improve the modelling efficiency.

4.1. *Newton-Raphson Method*

The Newton-Raphson method is an open method that uses the derivative of the function to generate guesses. We can use this method because REBOUND keeps track of the instantaneous velocity of each celestial object. The method works by using the derivative to find the intercept of the tangent line, and setting that as the new guess. This method is very efficient when the initial guess in near the true value, with the number of significant digits approximately doubling each iteration, while requiring one position calculation and one velocity calculation (Press et al. 2007).

4.2. *Bisection Method*

The y-position of our brackets have opposite signs and the intermediate value theorem states that if two points on a continuous function have opposite , then there exists a point between them that is equal to zero. The bisection method is a closed method that uses this fact to converge to a solution. It works by finding the y-position of the midpoint of the bracketed interval and use the midpoint to replace the bracket with the same y-position sign. This halves the error each iteration and while requiring only one position calculation (Press et al. 2007).

4.3. *Ridders' Method*

If a function is smooth, then the secant line of points close together will be a good approximation. Ridders' is an open method that works by using the midpoint and endpoints of the bracket to fit and factor out an exponential function; this attempts to straighten out the function. The intercept of the secant line between the bracketed endpoints of our straightened out function becomes the new guess, and replaces the bracket with the same y-position sign. When the size of the bracket is small the number of significant digits approximately doubles each iteration, while requiring two position calculations (Press et al. 2007).

4.4. *Kou-Wang-Ostrowski Method*

Ostrowski's method treats Newton's method as an interim solution and adds an extra step to obtain a better estimate. In 2010, Dr. Jishang Kou and Dr. Xiuhua Wang presented a higher-order variant of Ostrowski's method containing three interim solutions. The variant used was equation (26) in Kou & Wang 2010. The idea of this method is to obtain an accuracy similar to multiple Newton's method iterations, while performing fewer derivative calculations. Each iteration increases the significant digits by a factor of eight while requiring three position calculations and one derivative calculation. This high convergence rate actually hindered its usefulness because floating point numbers on a 64-bit computer can only have 16 digits of precision. When finding roots using a very low tolerance a second iteration is sometimes required, in such cases a divide by zero error typically occurs due to identical subsequent interim solutions. To remedy this we exit an iteration early if the difference between subsequent guesses is less than the tolerance. Exiting an iteration early this way also helps prevent computing roots with a greater precision than is needed.

4.5. *Brent's Method*

The higher the order of the polynomial we fit to the function, the better the estimate it provides. So fitting a quadratic function using the bracket points and the midpoint should provide a good estimate. However, not every quadratic function has real roots, so instead we can fit a sideways parabola (inverse quadratic) to the data. Brent's is a closed method that uses the root of a fitted inverse quadratic, determines the position at this point and replaces the bracket with the same y-position sign. If an iteration's fitted root is outside the original interval or is too close to an existing bracket, a bisection step is done instead. Brent's method is often used as the standard root-finding method, especially when the function's derivative is not available (Valladio & McClain 1997).

5. ACCURACY TESTING

We can use our analytical Keplerian transit-times to test the accuracy of our transit-time finder. Two-body systems undergo Keplerian motion, so the computed transit-times of a single planet orbiting a star should match the analytical values. This testing was performed on the WHFast, IAS15, and Bulirsch-Stoer integrators using both default and fast-mode with each of the five root-finding methods for target accuracies (tolerances) of 10^{-2}, 10^{-4}, 10^{-6}, 10^{-8}, 10^{-10}, and 10^{-12} days (see Figure 2 in Appendix B). For each instance we tested 1000 randomly generated systems, where the mass of the star was uniformly sampled from $[0.1, 10)$ solar masses, the mass of the planet from $[1.66 \times 10^{-7}, 9.55 \times 10^{-4})$ solar masses, the planet's semi-major axis from $[0.387, 1.11)$ AU, the planet's eccentricity from $[0.007, 0.206)$, the planet's longitude of ascending node from $[0, 2\pi)$ radians, the planet's true anomaly from $[0, 2\pi)$ radians, the planet's argument of periapsis from $[0, 2\pi)$ radians, and the planet's inclination from $[-0.1, 0.1)$ radians. The analytical and computed transit-times for each system were determined over a period of 1500 days. The absolute difference between the analytical and computed transit-times were then computed and the largest difference for each system was recorded, as you are only as accurate as your worst calculation.

For WHFast with default and fast-mode for each root-finding algorithm, all absolute difference values were less than the tolerance up until a tolerance of 10^{-10} days, afterwards there was no further decrease in absolute difference. The IAS15 integrator had absolute difference values that were slightly less than that of WHFast but were still less than the tolerance up until a tolerance of 10^{-10} days. For Bulirsch-Stoer with default and fast-mode for each root-finding algorithm, all absolute difference values were less than the tolerance up until a tolerance of 10^{-4} days, afterwards there was no further decrease in absolute difference. This was expected as the integrator is known to be most useful for situations where only medium accuracy is required. These determined accuracies are satisfactory, as the transit-times for Trappist-1 in Grimm et al. 2017 were calculated with uncertainties on the order of 10^{-4} days.

6. SPEED COMPARISON

We compared the speed of each of the three integrators with each of the five root-finding methods with both default and fast-mode to NbodyGradient (Agol, Hernandez, & Langford 2021). For the comparison we found the transit-times of a system analogous to Trappist-1 over a period of 1500 days. To test performance with fewer planets, the necessary number of outermost planets were removed. In Agol, Hernandez & Langford 2021, transit-times were computed with a tolerance of 10^{-12} days, and there does not appear to be a way to change the root-finding tolerance of NbodyGradient. In order to get a somewhat fair comparison for our transit-timing we used a tolerance of 10^{-12} days (see Figure 3 in Appendix C), despite our accuracy testing showing that we are unable to achieve such an accuracy. We also performed a speed comparison using a tolerance of 10^{-4} days (see Figure 4 in Appendix C), as this is likely to be the tolerance used when implemented. Tolerance aside, NbodyGradient has the ability to compute the derivatives of the times of transit with respect to the initial conditions, which are likely useful when fitting a model to observed tranit-times.

The fastest integrator was WHFast, with runtimes that were approximately one-eighth of NbodyGradient's runtimes and approximately one-third of the runtimes using IAS15 and Bulirsch-Stoer for systems with five or more planets and a tolerance of 10^{-12} days. WHFast remained the fastest with a tolerance of 10^{-4} days. When using a low root-finding tolerance (10^{-12} days) or the WHFast integrator, the Newton-Raphson method performed best, with Brent's method being only slightly slower. This was not the case when using the IAS15 and Bulirsch-Stoer integrators with a more reasonable root-finding tolerance (10^{-4} days), as Kou-Wang-Ostrowski's method was the fastest, however the differences in runtimes among root-finding algorithms were relatively small. The fast-mode of root-bracketing performed 2.5 times faster than the default with WHFast, however its usefulness is constrained to well-behaving systems.

Some other interesting observations from the speed test are that default outperformed fast-mode using the IAS15 and Bulirsch-Stoer integrators. These integrators tend to use larger timesteps, thus requiring fewer timesteps per orbit than WHFast. This makes the downside of calculating the Keplerian transit-times and having a larger bracketed interval outweigh the advantage of not having to check the position of each planet after every timestep. Also Ridders' method of root-finding performed quite poorly with fast-mode, being the slowest method in most cases. This makes sense given that bracketed intervals are larger with fast-mode and Ridders' operates on the fact that points close together are joined by a slight curve that can be fit and factored out.

7. CONCLUSIONS

The goal was to create a program to calculate the transit-times of a simulated planetary system with a focus on speed, while maintaining accuracy. Each integrator and root-finding algorithm tested were able to agree with the analytically computed transit times with at least an accuracy of 10^{-4} days. Given that the transit-times for Trappist-1 in Grimm et al. 2017 were calculated with uncertainties on the order of 10^{-4} days, we are confident in the code's accuracy.

From our speed tests we can see that our code scales well as the number of planets in the system increases. However, without deploying model fitting algorithms it is difficult to be certain whether or not the code will be an effective tool for fitting a model planetary system to observed transit-times. However, using the results of our speed test we know on a standard laptop our code would take approximately one day to determine the transit-times of 100 000 seven-planet systems simulated for 1 500 days using the WHFast integrator, the fast-mode of root-bracketing, and the Newton-Raphson root-finding method with a tolerance of 10^{-4} days. It would take NbodyGradient approximately 26 days to determine transit-times of 100 000 seven-planet systems simulated for 1 500 days. NbodyGradient does have the ability to compute the derivatives of the times of transit with respect to the initial conditions, however at this time we do not know if these derivatives will allow for accurate model fitting using 26 times fewer systems. The next step is to examine different model fitting algorithms with our transit-timing code to determine its usefulness for fitting a model planetary system to observed transit-times.

REFERENCES

Agol, E., Hernandez, D. M., & Langford, Z. 2021, MNRAS, 507(2), 1582
Grimm et al. 2017, A&A, 613, A68
Hairer, E., Wanner, G., & Norsett, S. P. 1993, Springer, 224
Kou, J., & Wang, X., 2010, Appl. Math. Lett., 23, 92
Press, W.H., Teukolsky, S. A., Vetterling, W.T., & Flannery, B.P. 2007, Cambridge, 442
Rein, H. & Liu, S. F. 2012, A&A, 537, A128
Rein, H. & Speigel, D. S. 2015, MNRAS, 446(2), 1424
Rein, H. & Tamiyo, D. 2015, MNRAS, 452(1), 376
Tsui, J. B. 2005, Wiley
Valladio, D. A., & McClain, W. D. 1997, McGraw-Hill
Wolszczan, A. & Frail, D. A. 1992, Nature, 355, 145

APPENDIX

APPENDIX A: DERIVATION OF THE KEPLERIAN TRANSIT-TIMES

A planet's k^{th} transit can be determined using its orbital period (P) and its first transit-time (t_{transit_0}).

$$t_{\text{transit}_k} = (k \times P) + t_{\text{transit}_0} \tag{A1}$$

Now we must find the first transit-time for each planet. This can be done by dividing the difference between mean anomalies (M) by the mean motion (n) (Valladio & McClain 1997).

$$t_{\text{transit}_0} = \begin{cases} \frac{M_{\text{transit}} - M_{\text{initial}}}{n} + t_{\text{initial}} & M_{\text{transit}} \geq M_{\text{initial}} \\ \frac{M_{\text{transit}} - M_{\text{initial}} + 2\pi}{n} + t_{\text{initial}} & M_{\text{transit}} \leq M_{\text{initial}} \end{cases} \tag{A2}$$

Now we must calculate the mean anomaly at the time of transit, we can do this using the eccentricity (e), true anomaly at the time of transit (ν_{transit}), and Kepler's equation (Valladio & McClain 1997).

$$M_{\text{transit}} = (\arctan2(\frac{\sqrt{1-e^2}\sin(\nu_{\text{transit}})}{1+e\cos(\nu_{\text{transit}})}, \frac{e+\cos(\nu_{\text{transit}})}{1+e\cos(\nu_{\text{transit}})}) - e\frac{\sqrt{1-e^2}\sin(\nu_{\text{transit}})}{1+e\cos(\nu_{\text{transit}})})\, mod\, (2\pi) \tag{A3}$$

The range of the arctan function is ($-\frac{\pi}{2}$, $\frac{\pi}{2}$), since the mean anomaly is the angular distance from pericenter we want the range of our results to be on the interval $[0, 2\pi)$. By using the 2-argument arctangent (arctan2(y,x)) (Tsui 2005) we obtain results on the interval $(-\pi, \pi]$. Subsequently taking the result modulo 2π gives us results in the interval $[0, 2\pi)$.

$$\arctan2(y,x) = \begin{cases} \arctan(\frac{y}{x}) & x > 0 \\ \arctan(\frac{y}{x}) + \pi & x < 0 \text{ and } y \geq 0 \\ \arctan(\frac{y}{x}) - \pi & x < 0 \text{ and } y < 0 \\ \frac{\pi}{2} & x = 0 \text{ and } y > 0 \\ -\frac{\pi}{2} & x = 0 \text{ and } y < 0 \\ 0 & x = 0 \text{ and } y = 0 \end{cases} \tag{A4}$$

By setting the conversion of orbital elements to heliocentric elliptic y-coordinate equal to zero and solving for the true anomaly at the time of transit (Valladio & McClain 1997).

$$y = r(\sin(\Omega)\cos(\omega + \nu_{\text{transit}}) + \cos(\Omega)\sin(\omega + \nu_{\text{transit}})\cos(i)) = 0 \tag{A5}$$

$$\nu_{\text{transit}} = -\arctan2(\sin(\Omega), \cos(i)\cos(\Omega)) - \omega \tag{A6}$$

APPENDIX B: ACCURACY TEST PLOTS

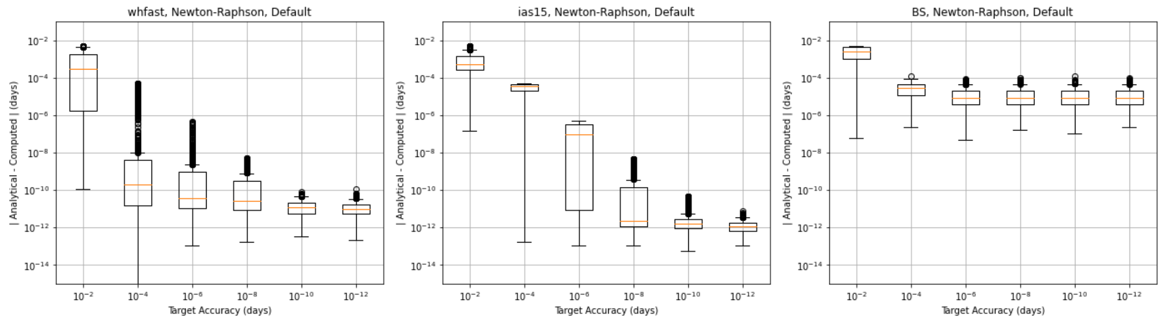

Figure 2. The largest absolute difference for each of 1000 randomly generated two-body systems simulated for 1500 days using default root-bracketing with Newton-Raphson's algorithm and the WHFast (left), IAS15 (middle), and Bulirsch-Stoer (right). The box plots indicate that for WHFast and IAS15, the absolute difference is less than or equal to the tolerance (target accuracy) up to 10^{-10} days. For Bulirsch-Stoer it is 10^{-4} days. This is a success because the current uncertainty for observing transit-times is approximately 10^{-4} days.

APPENDIX C: SPEED TEST PLOTS

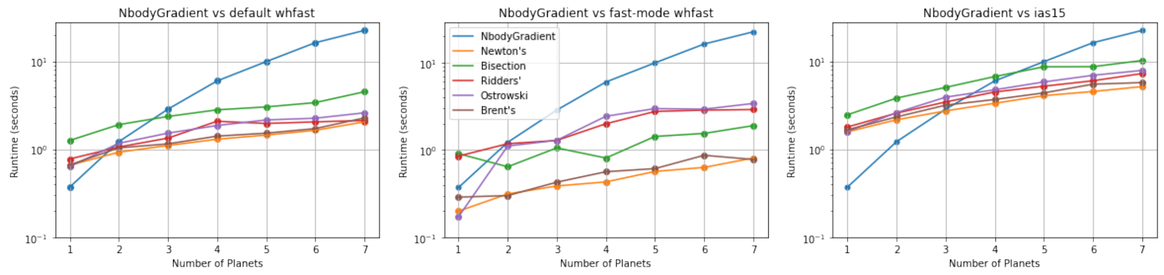

Figure 3. A speed test of the root-finding methods using: WHFast with default root-bracketing (left), WHFast with fast-mode root-bracketing (middle), and IAS15 with default bracketing (right). The root-finding tolerance used was 10^{-12} days to somewhat fairly compare to NbodyGradient (blue). This test was done by finding the transit-times of a system analogous to Trappist-1 over a period of 1500 days. To test performance with fewer planets, the necessary number of outermost planets were removed. Both WHFast and IAS15 scale well for systems with many planets. The best performing setup was the WHFast integrator with fast-mode root-bracketing and Newton-Raphson's root-finding algorithm.

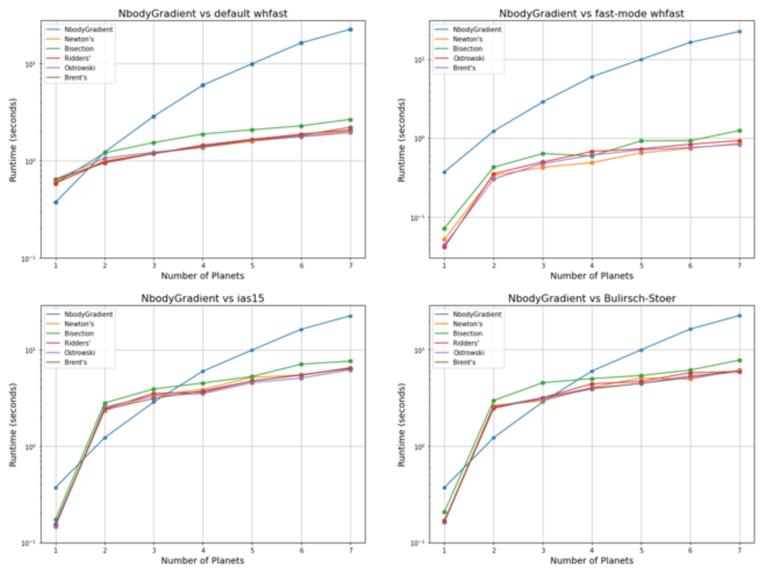

Figure 4. Speed testing with a root-finding tolerance of 10^{-4} days. We can WHFast with fast-mode root-bracketing solidly outperform the other integrators, with the exception of Ridders' root-finding algorithm. Ridders' performance with the larger bracket was worse than the other integrators and has been omitted for visibility.