McGill University

Faculty of Science

Final Project
Calculating Transit Timing Variation Curves using N-Body Simulations

PHYS 512

Laurie Amen - ID 260907757
Dhvani Doshi - ID 261060689
Jennifer Glover - ID 261153731

December 5, 2023

1 Introduction

Exoplanets, planets outside of our solar system, have been sought-after objects of interest since the first exoplanet discovery in 1992. While the number of detected exoplanets has been increasing greatly in recent years, with over 5000 exoplanets having been discovered, many exoplanet systems have not been studied in detail. The study of exoplanets can be uniquely challenging given they are, by nature, much fainter and smaller than their host stars.

The most popular method to study exoplanets involves observing their host star's light curve. A light curve is the time series of total flux received from a star. In a system with a single non-variable star and no planets, the light curve will be constant. If the system has a planet, as the planet transits (passes in front of its host star), there will be a small decrease in light observed from the star. Thus, a periodic dip in a light curve is indicative of a planet. The orbital period of the planet can be directly measured by timing these dips in brightness. From the orbital period and Keplarian dynamics, the mass ratio of the star and planet can be determined.

However, light curves can only give information about *transiting* planets. Other planets in the system which do not transit will not directly appear in the light curve. However, in a system with many planets, the gravitational interactions between the planets can lead to small, but measurable, deviations in the orbital periods of the transiting planets. Due to deviations in the orbital periods, the transit of a planet does not occur at a constant periodic rate. Instead, there are variations to the expected time a planet transits its star. These transit timing variations (TTVs) can be used to detect additional planets in a system and to characterize planetary systems. Specifically, they yield information about mass ratios and orbital eccentricities. For example, Gillon et al. [2017] used TTVs to infer the mass ratios of the Trappist-1 planets. Kaye et al. [2022] used TTVs to constrain the masses and eccentricities of two planets in the TOI-270 system.

TTVs cannot be calculated analytically, since the N-body problem cannot be solved analytically. Thus, all TTV analyses rely on numerically solving the N-body problem. Specifically, a numerical integrator must be used to determine the orbits of the planets.

2 Problem Description

This project aims to recreate the TTVs observed by Kaye et al. [2022] using N-body simulations to generate light curves.

We used the Rebound N-body simulation code to numerically integrate the orbits of the planets. Specifically, we solved the classical N-body problem (where only the classical gravitational force is considered). The effects of general relativity, such as the curvature of space-time, were not considered in this work. The curvature of space-time is known to cause observable orbital variations, such as the precession of the perihelion of Mercury. However, these orbital variations are typically very small when considering time scales of less than 100 years and thus should not be significant in this context.

From the N-body orbital information, we computed the light curve of the system. We then extracted the TTV information from the light curve, partly replicating the observational process.

In addition, when calculating the light curves, we assumed that the planetary systems were viewed edge-on (inclination angle of $i = 90^\circ$) and the planetary surfaces were perfect absorbers. That is, we assumed the planets in the simulation did not reflect any light from their host stars. In reality, planets will reflect a small amount of light from their host stars; an average planetary albedo is $\sim 20\%$. Furthermore, we assumed that the planets did not emit any thermal radiation. Physically, planets will emit some thermal radiation as they are not at absolute zero. However, the amount of thermal radiation emitted by a planet will be small compared to the host star.

3 Code Description

3.1 Rebound: N-body Code

The N-body problem is essentially a set of ordinary differential equations in which the time evolution of each particle depends on its position and velocity and the positions of all other particles.

Rebound keeps track of the mass, position, and velocity of each particle in the simulation individually. Importantly for our use, Rebound allows the user to assign a label to each particle so that it can be tracked uniquely throughout the simulation (see Figure (1)). In our simulations, we set the reference frame as the center-of-mass frame. This ensures that the center-of-mass remains at the origin for the entire simulation. Without this, the system may eventually drift from the origin which can result in the accumulation of numerical errors.

Figure 1: Initialization of a Rebound simulation with two particles representing the Sun and Mercury. The parameters 'a' and 'e' represent the initial orbital semi-major axis and eccentricity, respectively. They are used to set the starting position of a particle and do not fix the orbital parameters of a particle for the entire simulation. Finally, the reference frame is set to the center-of-mass.

```
sim = rebound.Simulation()
sim.units = ('AU', 's', 'Msun')
sim.add(m=1.0, hash='sun')
sim.add(m=1.65e-7, a=0.39, e=0.205, hash="mercury")
sim.move_to_com()
```

In Rebound, the acceleration of the i-th particle is calculated by summing over the gravitational acceleration induced by all other particles (see Equation (1)) [Rein and Liu, 2012].

$$\vec{a}_i = \sum_{j=0}^{N_{\text{active}}-1} \frac{Gm_j}{(r_{ij}^2 + b)^{3/2}} \hat{r}_{ij} \tag{1}$$

where N_{active} is the number of particles in the simulation, G is the gravitational constant, m_j is the mass of the j-th particle, and r_{ji} is the separation between the i-th and j-th particles. In scenarios where the particles may experience close interactions, the gravitational softening parameter, b, can be set to some finite value to prevent the acceleration from becoming unphysical. In our simulations, the particles did not experience close encounters, and the gravitational softening parameter was set to zero. When there are few particles in the simulation, as was our case, Equation (1) can calculated directly. For simulations with a very large number of particles Equation (1) may become time-consuming to run. For this case, Rebound offers an alternate scheme that considers the effect of only nearby particles on acceleration.

Rebound contains a number of integrators for solving the N-body problem. For all integrators, Rebound employs the Drift-Kick-Drift scheme where the position of each particle is updated using the Hamiltonian. The Hamiltonian is considered to be the sum of the kinetic term and the potential term (ie. $H = H_1 + H_2$). For example, the kinetic term of the Hamiltonian may be $H_1 = \frac{1}{2}p^2$ and the potential term may be the gravitational potential $H_2 = \Phi(q)$, where p and q are momentum and position. This means that during the first drift step, each particle is evolved using the kinetic term of the Hamiltonian (H_1) for half a time-step. During the kick step, each particle is evolved using the potential term of the Hamiltonian (H_2) for a full time-step. Finally, during the last drift step, each particle is evolved again using the kinetic term of the Hamiltonian (H_1) for another half time-step. At each step in the integration, all particles evolve the same amount in time, however, the time-step to be varied over the course of an integration.

We employ the Integrator with Adaptive Step-size control, 15th order (IAS15) [Rein and Spiegel, 2015]. This integrator was designed to simulate gravitational systems, such as planetary orbits, down to machine precision over 10^9 orbits. While IAS15 has an automatic adaptive step-size routine, this routine was designed for close encounters between particles. In our simulations, the particles did not have close encounters, however, we chose to manually adjust the time step to coincide with transits (see Section 3.2).

3.2 Variable Timestep

A variable time-step was introduced into the simulation code to increase the accuracy of the mid-transit time (MTT) calculations without sacrificing computational efficiency. While testing for the One-Planet example seen in Section 4.2, it was noticed that the transit timing variations (TTVs) did not match theoretical expectations. According to theory, the TTVs for one planet should be a constant line with the TTV for each transit at 0 as there are no additional gravitational interactions with other planets. However, our preliminary results showed high variance about 0. This was attributed to the fact that our MTT calculations used either the center index or the average of the two endpoints of a transit to determine the MTT. As a result, if there were large timesteps during the transit, the middle index or the average of the two endpoints may not be indicative of the true MTT.

Therefore, a variable timestep was implemented where the simulation uses a predefined smaller timestep if any of the planets in the simulation are in a defined "transit region". This transit region is determined to be in the $x > 0$ regime where $-R_* - R_{maxp} - b \leqslant y \leqslant R_* + R_{maxp} + b$. Here, R_* is the radius of the star, R_{maxp} is the largest planetary radius in the system, and b is a buffer such that the simulation uses the smaller timestep before and after the true transit of the planet. If no planet is within this transit region then the simulation will use a predefined larger timestep.

3.3 Light Curve Code

The code for calculating the light curve of the host star is available in the file `lightcurve.py`. This code utilizes the positions and radii of various planets and the host star to determine the flux obstructed by a planet during its transit across the host star. The code operates under the assumption that all objects are situated on the $z = 0$ plane, and the observer viewing the transit is positioned at $x = \infty, y = 0$. Consequently, the planet is considered to transit the star when it is in the $x > 0$ region. The program tracks both the front and back edges of the planet's surface, along with those of the star. The computation of the surface area blocked by the planet occurs when the planet transits in front of the star, and this area is directly proportional to the amount of flux obstructed. The code independently calculates the flux blocked for each individual planet. The following code snippet shows the computation of the transit location and the corresponding amount of blocked flux, with the variables "planet" and "star" representing a `pandas` DataFrame for the respective objects (see Figure (2)). The "X" and "Y" columns of the data represent the X and Y positions of the objects.

Figure 2: This figure shows the code that defines when a planet is in transit and how much of the star's flux is blocked during the transit.

```
# First consider full transit scenario where the entirety of the planet is blocking the star
full_transit = (planet["X"] > 0) & (planet_front<=star_front) & (planet_back>=star_back)
planet.loc[full_transit,"BlockedFlux"] = (planetR[i]/starR)**2

# Next consider partial transits where the planet is entering full transit or exiting full transit
entering = ((planet_front>=star_back) & (planet_back<=star_back))
exiting = ((planet_front>=star_front) & (planet_back<=star_front))
partial_transit = (planet["X"] > 0) & (entering ^ exiting)

# Calculate distance between centers
distance = np.abs(planet["Y"].loc[partial_transit]-star["Y"].loc[partial_transit]).values

# Find the blocked area
A_intersect = self.intersection_area(distance, starR, planetR)
planet.loc[partial_transit,"BlockedFlux"] = (A_intersect/(4*np.pi*starR**2))
```

3.4 TTV Code

In the absence of other perturbing bodies, the 'ephemeris' of a planet's orbit around its star, or its transit times as a function of transit number, called 'epoch', is a linear relationship given by

$$C = T_0 + P \times E \tag{2}$$

where T_0 is the time of the first measured transit, P is a constant orbital period, and E is the epoch (Agol and Fabrycky [2018]). C stands for 'calculated transit times', as this is the theoretical model of an orbiting body's ephemeris. The presence of other perturbing bodies can bring about instantaneous changes in orbital parameters such as eccentricity e, semi-major axis a, or longitude of perisatron ω, which in turn result in a non-constant orbital period P. TTVs are computed by measuring the deviation of a planet's observed ('O') transit times from a theoretical linear ephemeris, such that $TTV = O - C$.

In practice, extracting TTVs requires fitting measured light curves to a model and extracting the mid-transit times (MTTs) of each individual transit, fitting them to a linear ephemeris model, and computing residuals (Kaye et al. [2022]). In the simulation for this project, some simplifications were made to the full observational process, including using individual planet light curves, instead of a combined light curve as would be measured in practice. This allowed us to skip the complex multi-step process of fitting a combined light curve and each individual transit, in favour of focusing on extracting TTV measurements.

The code computing first MTTs, then TTVs can be found in the file `ttv.py`, which takes light curve `pandas` DataFrames from `lightcurve.py` as input. First, transits are identified by isolating the data indices where the BlockedFlux is non zero, meaning the planet is in transit. Each group of consecutive indices is identified as an individual transit. For each transit, the MTT is estimated either by the middle index, or by interpolation. Although there is little difference in accuracy between these two methods for small timesteps, the interpolation method is recommended for larger timesteps. The interpolation scheme is shown in Figure (3), where `final` is a DataFrame containing the transit indices and flux of each identified transit separately. `self.time` is the array of times corresponding to each datapoint which comes with the light curve DataFrame, and is necessary for a variable timestep

Figure 3: This code snippet shows how mid-transit times are estimated by interpolation.

```
if method == "interpolate":
    # Alternate method, interpolation
    for i in range(0,m,2):
        dip_inds = final.iloc[:,i] # get transit indices

        if i==0:
            dipstart = dip_inds.iloc[0]
        else:
            dipstart = dip_inds.iloc[0]-1 # index of start of transit (take point right before the
    flux starts to be blocked)

        endind = -1
        dipend = dip_inds.iloc[endind] # index of end of transit
        while np.isnan(dipend):
            # accounts for transits that are a couple datapoints shorter than others
            endind -= 1
            dipend = dip_inds.iloc[endind]

        dipend += 1 # (take point right after the last point where there is blocked flux)
        midtime = (self.time[dipend]+self.time[dipstart])/2 # MTT, in *seconds*
        MTTs[i//2] = midtime
```

routine. The first MTT is usually removed from the data, as it is possible that the simulation started in the middle of a transit, rendering the first MTT estimation inaccurate.

After computing MTTs, the MTT vs. epoch data is fit to a linear model for P, the mean orbital transit time, and T_0 according to Equation (2). The fit is done by `scipy.optimize.curve_fit`, which uses a non-linear least squares method. The TTVs are then simply computed from the residuals of the fit, as show in Figure (4).

Figure 4: This code snippet shows how transit timing variations are obtained from mid-transit times using a linear fit.

```
tnum = range(len(self.MTTs)) # transit number array

# Linear model to fit MTT data
    def linmodel(x,m,b):
        return m*x + b

# Perform the linear fit to MTT data
popt, pcov = curve_fit(linmodel,tnum,self.MTTs,sigma=[self.dt for mtt in self.MTTs]) # take
    timestep as the error
model_rslt = linmodel(tnum,popt[0],popt[1])
print('slope= ',popt[0],' intercept= ',popt[1])
print('Errors=',np.sqrt(np.diag(pcov)))

TTVs = self.MTTs - model_rslt # TTV is basically the residual of the MTTs linear fit (Observed-
    Calculated)
```

The `computeTTVs()` function outputs the MTT data and its linear fit, as well as the $C - O$ TTV plot. An example of the MTT linear fit is shown in Figure (5), and corresponds to planet TOI-270d, see Section 5.

4 Code Tests

4.1 Testing Rebound and light curves with the Solar System

Firstly, we test Rebound and the light curve code using the solar system as it is familiar and well-understood.

To begin, we test Rebound with a two-body system composed of one particle representing the Sun and one particle representing the Earth. The two-body problem can be solved analytically, unlike the N-body problem which makes it ideal for testing the accuracy of the numerical integration. The simulation is initialized by setting the mass of the Earth to $3.0 \times 10^{-6} M_{\odot}$, the orbital semi-major axis to 1 au, and the orbital eccentricity to 0.017. The simulation was then run for 20 years (see Figure (6)). The period of Earth's orbit was found to be 365.272 days when fitting a sinusoid to its x-position over time, and 365.259 days when fitting to its y-position over time. The discrepancy

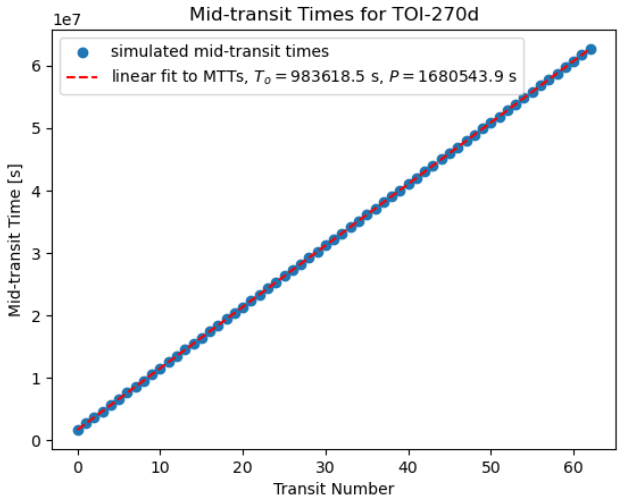

Figure 5: Example of a MTT fit to a linear ephemeris model for planet TOI-270d, yielding a mean orbital period P. The residuals of the fit constitute the TTVs.

between these results is due to the slight eccentricity of Earth's orbit; the difference disappears if Earth's orbital eccentricity is set to zero at the beginning of the simulation. From Kepler's Third Law, the orbital period of Earth is predicted to be 365.257 days. These results are consistent within ~ 0.02 days or ~ 30 minutes. In addition, the percent difference in kinetic energy of the Earth after the first orbit and after 18 additional orbits is on the order of 10^{-14}. These results indicate that the Rebound code is working as intended and conserving energy well.

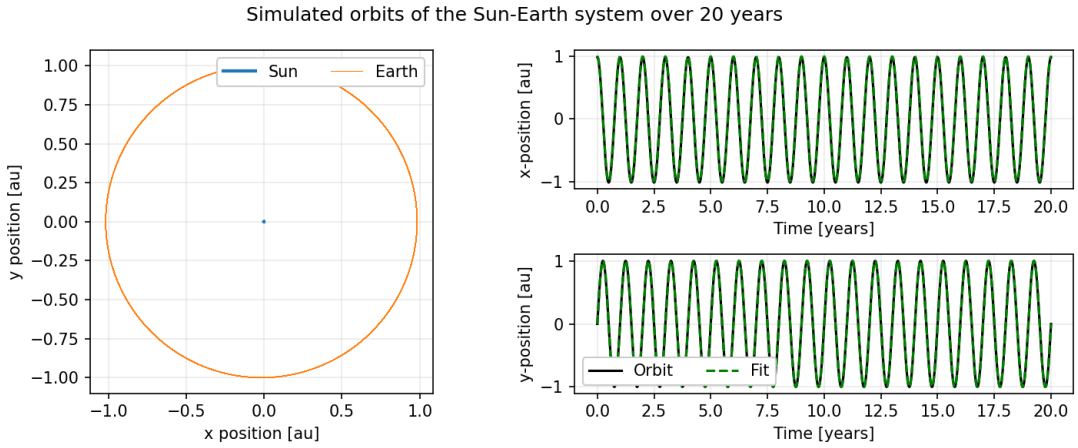

Figure 6: The left panel shows the simulated orbits of the Sun-Earth system over 20 years. The top right panel shows the Earth's x-position over time, and the bottom right panel shows the Earth's y-position over time. Earth's orbit is demonstrating a consistent period of ~ 1 year and is not changing in amplitude. This indicates that the simulation is working as intended.

Secondly, we test that the light curve is being generated as expected. To accomplish this, we simulated the evolution of the solar system for 2 years and produced an animation of the light curve as a function of time[1].

4.2 Testing TTV with One Planet

In a system comprised of a single planet and its host star, the orbital period of the planet is expected to remain constant, due to the absence of other objects exerting gravitational influences that might either accelerate or decelerate the planet in its orbit. Therefore, the TTV of the planet should theoretically be close to zero throughout its

[1]This animation is available in the GitHub as solarSystemLightCurve.gif

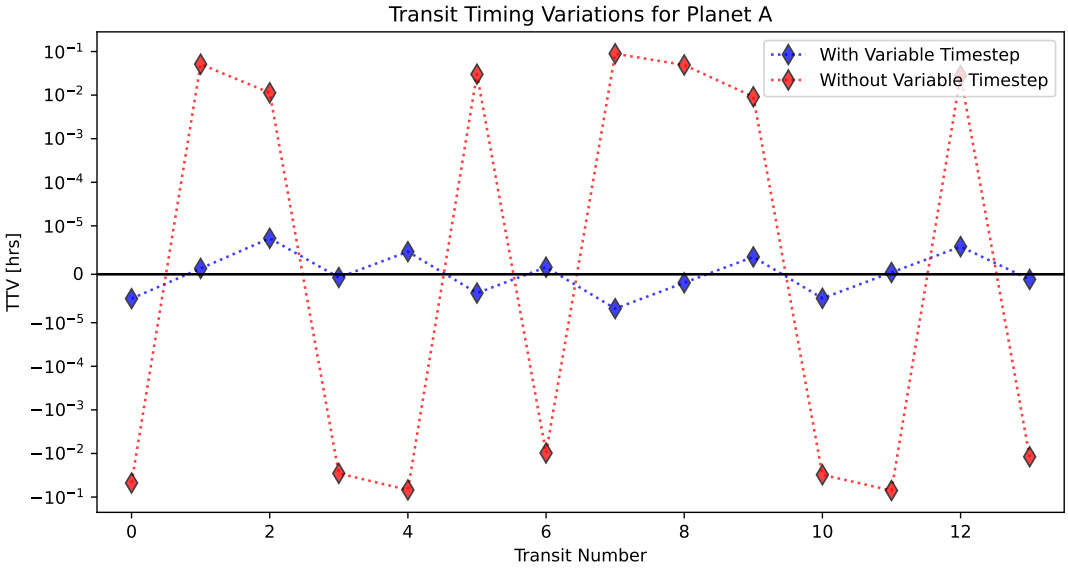

Figure 7: This figure shows the one planet code test for simulations with and without incorporating the variable timestep. This figure demonstrates that the use of a variable timestep improves the performance of our TTV calculations, providing results that more closely align with theoretical expectations.

transits. This one-planet scenario serves as a critical test case, allowing us to assess how errors propagate within our simulation and gain insights into the precision of our TTV calculations.

To evaluate this scenario, we constructed a simulation featuring a singular star and planet, employing parameters corresponding to those of the Sun and Venus. The mass, radius, or semi-major axis parameters in this code test are not important as the aforementioned TTV behaviour should be expected from any one-planet system. Initially, the simulation was executed without a variable timestep, therefore an overall timestep of $1000s$ was used. The simulation spans a duration of 10 years and then the TTVs are calculated. The resulting TTVs, depicted in Figure (7) in red, matched theoretical expectations as they remained constant throughout the transits. However, they exhibited a notable variance with values much larger than zero. In response to this variability, we introduced a variable timestep, described in Section 3.2, into the simulation to mitigate these fluctuations.

A subsequent simulation, with the same object parameters, implemented the variable timestep with the large timestep set to $1000s$ and the small timestep set to $0.1s$. This simulation also ran for 10 years and the resulting TTVs are shown in Figure (7) in blue. The adoption of a variable timestep significantly reduced the variance by a factor of 10^4. As a result, the variable timestep was utilized for further experiments to ensure the appropriate level of accuracy in our TTV values.

4.3 Testing TTV with Two Planets

After testing with one planet, it was crucial to validate the accuracy of our code for N bodies. Consequently, we conducted a simulation involving two planets in resonance, aiming to assess whether the computed TTVs aligned with theoretical expectations. TTV curves for a system with two bodies in close commensurability are anticipated to exhibit simple sinusoidal behaviour [Lithwick et al., 2012, Deck and Agol, 2016]. This implies that for planets with an orbital period ratio of 3:2 or 2:1, one should expect sinusoidal TTVs for both planets. Importantly, these curves should share the same periodicity while demonstrating anti-correlation. In other words, the maxima of one TTV curve should coincide with the minima of the other.

To test this scenario, the two-planet system, Kepler-128, was simulated where the planets Kepler-128b and Kepler-128c are Sup-Neptune pairs with the 3:2 orbital period resonance. The star and planetary parameters are provided in Table (1) and were taken from Hadden and Lithwick [2014]. The variable time step was implemented with the large timestep set to $1000s$ and the small timestep set to $10s$. The simulation ran for 4 years and the resulting TTVs were calculated for each planet and plotted in Figure (8). Sinusoidal curves were fit to the data points, using `scipy.optimize.curve_fit`, to demonstrate that the TTVs match sinusoids and share the same general period while remaining anti-correlated. These results match Figure (2) from Choksi and Chiang [2023], where the magnitude of

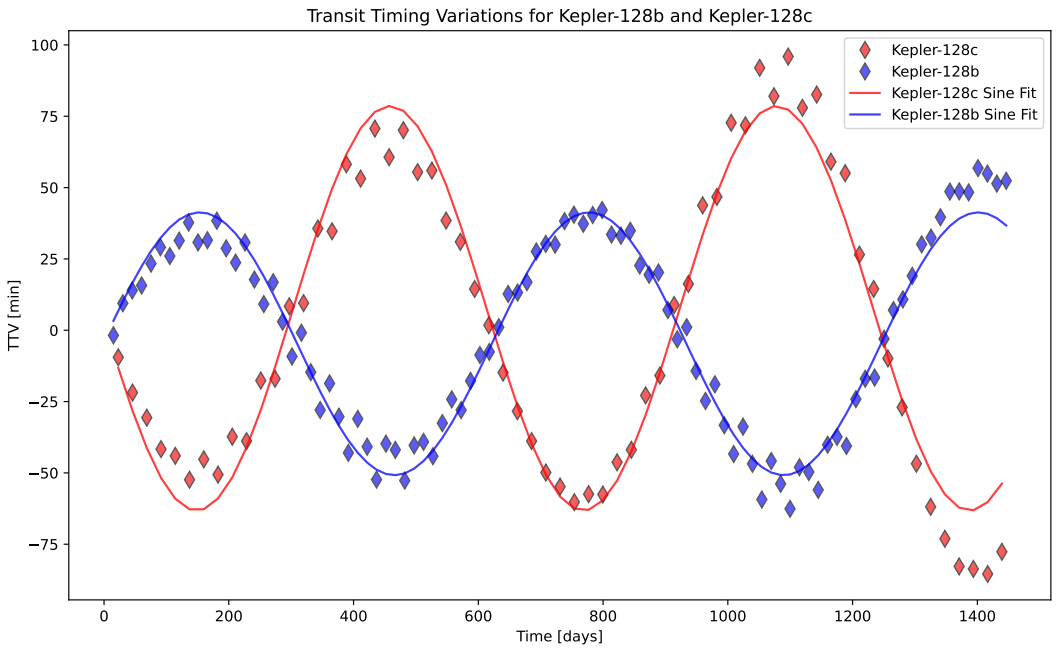

Figure 8: This figure shows the two-planet code test where two Sub-Neptunes are in orbital period resonance. This figure demonstrates that our TTV calculations closely match theoretical expectations as both TTV curves follow sinusoids.

the TTVs also ranges from -75 minutes to 75 minutes.

Table 1: This table shows the object parameters used in the simulation for the two-planet code test. Here, $M_\odot$ represents solar masses, and $R_\oplus$ represents Earth radii.

Object	Mass ($M_\odot$)	Radius ($R_\oplus$)	Period (days)
Kepler-128	1.18	180.9	NA
Kepler-128b	1.27×10^{-4}	1.13	15.090
Kepler-128c	9.86×10^{-5}	1.13	22.801

5 Results and Analysis

The main objective of this project was to attempt to reproduce the TTV trends of the Kaye et al. [2022] paper using an N-body code and light curve simulation. The parameters used as input for the simulation of the TOI-270 system are in Table 2, and are taken from the results of the paper. Different combinations of input parameters amongst those in Table 2 were tested, and the combination able to reproduce results from the paper most precisely included mass, radius, eccentricity e, mean orbital period P, and longitude of periastron ω. In particular, specifying the semi-major axis a and the mean orbital period P is redundant due to Kepler's 3rd law. We chose to specify P because its reported fractional error in Kaye et al. [2022] was much smaller than that of a. The error in P was of order 10^{-5} days and its value came directly from the TTV fit, while the error in a was of order $4R_*$ for planet b, and order $10^{-1}R_*$ for planets c and d, and the a values did not come directly from the TTV fit.

The simulation was integrated over 2 years, which is approximately the length of the survey in Kaye et al. [2022]. The large timestep was 100 seconds, and the small timestep was 10 seconds, with a total compute time of about 30 seconds. After producing light curves, MTTs and TTVs for each planet using `lightcurve.py` and `ttv.py`, the TTV curves were each fit to a sinusoidal function. Although we do not expect an exact sinusoidal model for this 3-planet system, the sinusoidal fit helps us extract certain trends from the TTV curve and compare them to the

Table 2: Best fitting stellar and planetary parameters of the TOI-270 system from Kaye et al. [2022], based on TTV and radial velocity data. Only the best-fitting values were used in our simulation. ω is the longitude of periastron. The inclination angle i was approximately 90° for all planets, so was neglected in our simulation.

Parameter	**Star**	**Planet b**	**Planet c**	**Planet d**
Mass	$M_* = 0.386 M_\odot$	$1.48 M_\oplus$	$6.20 M_\oplus$	$4.20 M_\oplus$
Radius	$R_* = 0.380 R_\odot$	$1.28 R_\oplus$	$2.33 R_\oplus$	$2.00 R_\oplus$
Semi-major axis a	-	$17.108 R_*$	$25.569 R_*$	$41.744 R_*$
Eccentricity e	-	0.0167	0.0044	0.0066
Mean orbital period P (days)	-	3.35992	5.66051	11.3819
$\sqrt{e}\cos\omega$	-	0.007	0.008	-0.021

findings of the paper. The sinusoidal fit is done using `scipy.optimize.curve_fit`, which uses a non-linear least squares method to fit a given model function to the data. The range of data used for fitting had to be limited to epochs 0 to 70, 20 to 70, and 10 to 55 for planets b, c, and d respectively, to get curves that best fit the simulated data. The poor fits when using the whole data were likely due to the data not exactly following a sinusoidal model. The results of the TTV simulations and fits for the TOI-270 system are shown in Figure (9). TTV curve periods in time units, as opposed to epochs or transit number, were computed by multiplying the epoch period from the fit by the mean orbital period of each planet.

Some general trends in Kaye et al. [2022] are recovered by the simulation. A notable result from the paper was that the TTVs for TOI-270c and d exhibited an anti-correlated trend with an amplitude of around 10 min, and a super-period of ~ 3 years. Figure (9) clearly shows the expected anti-correlation of the TTVs of the two outer planets. Additionally, the sinusoidal fits for planets c and d found amplitudes of 8.45 ± 0.07 min and 12.0 ± 0.2 min respectively, which are both of order ~ 10 min as expected. The recovered super-periods for planets c and d of 1.66 ± 0.01 years and 1.66 ± 0.04 years respectively, which is almost a factor of 2 smaller than the expected super-period. This discrepancy may be due to an underestimation of the error in the super-period from our analysis, which did not take the error in the input parameters into account. It may also be due to the sinusoidal model being inexact; in fact, the model can be seen significantly deviating from the data at the endpoints of the bottom two plots in Figure (9). The simulation was re-run for a total time of 3.5 years to test if the simulation could recover the ~ 3 year super-period if data was acquired over a longer period. Properly adjusting the fitting data range, the sinusoidal best fit gave super-periods of 3.2 ± 0.2 years and 3.13 ± 0.04 years for planets c and d respectively, which matches the results of the paper.

The TOI-270b TTV curve roughly matches the data from Kaye et al. [2022], but the exact amplitude and period of the curve were not mentioned in the paper. The sinusoidal fit to the simulation data gave an amplitude of 0.069 ± 0.05 min, and a period of 30.30 ± 0.05 days or 5.35 ± 0.02 epochs. By visual inspection of Figure (3) in Kaye et al. [2022], the small amplitude and the period of ~ 5 epochs seem consistent with the observational data, which shows ~ 10 oscillations per 50 epochs.

In terms of orbital periods, the MTT linear fit correctly recovered the values used to setup the simulation as ~ 3.36, ~ 5.66, ~ 11.38 days for planets b, c, and d respectively.

6 Discussion and Conclusion

The simulations recovered some, but not all of the expected trends from Kaye et al. [2022]. Notably, it was unable to recover the ~ 3 year super-period of planets c and d by integrating over only 2 years. However, when the integration time was increased to 3.5 years, the simulation was able to recover the expected super-period, showing that the non-recovery for the 2-year integration was likely due to missing data to accurately represent the trend, and the data not perfectly matching a sinusoidal model. Moreover, our TTV calculator was able to replicate the theoretical results for the one and two-planet orbital systems.

Through careful analysis, we found that the performance of the calculator was highly dependent on the total time integrated as well as the timesteps that were taken during integration. It was most important to take extremely small timesteps as a planet was entering or exiting a transit as well as during a transit. This allowed our mid-transit time calculations to have better accuracy, thus resulting in a more accurate TTV curve. Furthermore, the accuracy of our TTV curves was also very dependent on the uncertainty of the orbital and planetary parameters used in the simulation. For example, using semi-major axis values that had large uncertainties would result in discrepancies between our TTV curves and the observed TTV curves for that system. Instead, we would have to use other values

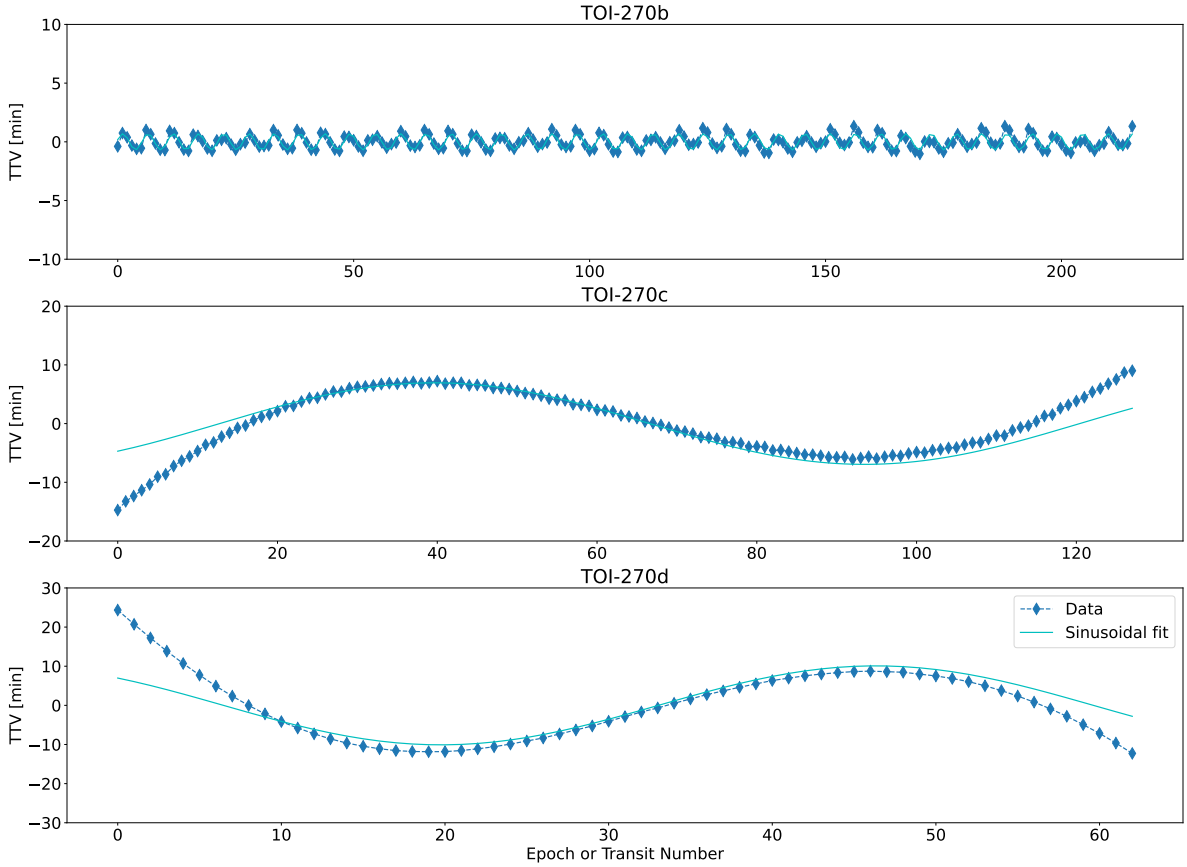

Figure 9: Results of the TTV simulation for the TOI-270 3-planet system for a 2 year integration. TTV curves are fit to a sinusoidal function to extract trends. These results are compared to Fig. 3 in Kaye et al. [2022] in the text.

such as the period of the orbit, which had better constraints to better replicate the observed TTV curves.

In general, we found that employing an existing N-body code has several advantages over developing our own. N-body simulations can be computationally heavy, and thus very time-consuming, to run. The Rebound code is entirely written in C and can be compiled in Python, which greatly decreases run-time. If we were to have written our own N-body code it would have been written in Python. This would likely have resulted in an increased run-time compared to Rebound simulations. In addition, Rebound has several built-in integrators which are specialized for orbital simulations. While the IAS15 is an implicit integrator and is conceptually similar to implicit numeric integrators covered in class, it is more difficult to code than the methods in class. For example, it uses a predictor-corrector scheme to determine coefficients used to update position and velocity.

While the use of Rebound greatly improved the simulation aspect of our code, there are a couple of improvements we would make to other aspects to better the performance of our TTV calculator. In particular, we would adjust the technique we use to calculate the mid-transit times. An alternative method would include fitting the light curves to a top hat function and then interpolating the top hat function to find the mid-transit time. This would provide better accuracy for the mid-transit time than an approximate interpolation. Moreover, we would try implementing a simulation that can take errors in the orbital parameters into account to develop error bars with our TTV values as well. Lastly, we would experiment with fitting the TTVs to more complex models other than sinusoids.

The success of our TTV calculator using Rebound simulations in replicating both theoretical and observational results suggests that this code could be utilized for MCMC sampling to retrieve orbital and planetary parameters. In this case, the MCMC sampler would sample through various planetary and orbital parameters and input these into the simulation. The resulting simulated TTV points would be compared to the real data to assess if the inputted

parameters were likely to have created the true TTV observations. This extension to our project was attempted, however, the runtime was heavily dependent on how long it took for the Rebound simulations, and therefore we could not provide concrete results.

In conclusion, this work represents the successful development of a TTV code that employs Rebound to numerically integrate planetary orbits. Our program is able to replicate theoretical and observational TTV results and shows promise for future work in fitting orbital and planetary parameters.

7 Statement of Contribution

Individual contributions to the project are summarized as follows.

Laurie wrote the TTV code and computed the simulations for the TOI-270 system. She also wrote the corresponding sections of the report.

Dhvani wrote the lightcurve code as well as the variable timestep code. She also ran the simulations for the One/Two Planet code tests. Dhvani wrote the corresponding sections of the report.

Jen wrote the Introduction, Problem Description, and Rebound Code Description sections of the report. She also wrote the code to make the animations and ran the simulations for testing Rebound on the Solar System.

References

E. Agol and D. C. Fabrycky. Transit-Timing and Duration Variations for the Discovery and Characterization of Exoplanets. In H. J. Deeg and J. A. Belmonte, editors, *Handbook of Exoplanets*, page 7. 2018. doi: 10.1007/978-3-319-55333-7_7.

N. Choksi and E. Chiang. Exciting the transit timing variation phases of resonant sub-Neptunes. , 522(2):1914–1929, June 2023. doi: 10.1093/mnras/stad835.

K. M. Deck and E. Agol. Transit Timing Variations for Planets near Eccentricity-type Mean Motion Resonances. , 821(2):96, Apr. 2016. doi: 10.3847/0004-637X/821/2/96.

M. Gillon, A. H. M. J. Triaud, B.-O. Demory, et al. Seven temperate terrestrial planets around the nearby ultracool dwarf star TRAPPIST-1. , 542(7642):456–460, Feb. 2017. doi: 10.1038/nature21360.

S. Hadden and Y. Lithwick. Densities and Eccentricities of 139 Kepler Planets from Transit Time Variations. , 787 (1):80, May 2014. doi: 10.1088/0004-637X/787/1/80.

L. Kaye, S. Vissapragada, M. N. Günther, et al. Transit timings variations in the three-planet system: TOI-270. , 510(4):5464–5485, Mar. 2022. doi: 10.1093/mnras/stab3483.

Y. Lithwick, J. Xie, and Y. Wu. Extracting Planet Mass and Eccentricity from TTV Data. , 761(2):122, Dec. 2012. doi: 10.1088/0004-637X/761/2/122.

H. Rein and S. F. Liu. REBOUND: an open-source multi-purpose N-body code for collisional dynamics. , 537:A128, Jan. 2012. doi: 10.1051/0004-6361/201118085.

H. Rein and D. S. Spiegel. IAS15: a fast, adaptive, high-order integrator for gravitational dynamics, accurate to machine precision over a billion orbits. , 446(2):1424–1437, Jan. 2015. doi: 10.1093/mnras/stu2164.