Using the Yoshida Integration Method for Exoplanetary Motions around τ Ceti

Ryan Brady

05/04/2023

1 Abstract

Astrophysical programming serves as a powerful computational tool, enabling researchers to model, simulate, and analyze complex celestial phenomena, thus paving the way for groundbreaking discoveries that deepen our understanding of the universe and its underlying mechanisms. In this paper, I present a Python-based numerical model that simulates the orbital motion of the seven exoplanets in the τ Ceti system, a Sun-like star system that is of particular interest to researchers due to its proximity and potential habitability. Using the Yoshida integration method, an orbital simulation individually for each planet is conducted. The simulation results are visualized in a two-dimensional plot, which displays the planetary orbits in the τ Ceti system alongside the position of the central star. Furthermore, the error in the final positions of the planets compared to their initial positions is computed, providing a measure of the simulation's accuracy. The findings reveal that the fourth-order Yoshida integration method provides a low error range for the planetary orbits, with errors ranging from approximately 1.46×10^{-4} AU to 5.75×10^{-4} AU. Additionally, several notable patterns in the planetary orbits are observed, including eccentricities and semi-major axes, which could inform our understanding of planetary dynamics and habitability. The simulation results provide a valuable foundation for future research in exoplanetary systems, informing the search for habitable exoplanets and deepening our understanding of the processes that govern planetary formation and evolution, as well as proves that the Yoshida method is capable of solving complex differential equations.

2 Introduction

τ Ceti, a G-type main-sequence star, is located approximately 11.9 light-years away from our solar system in the constellation Cetus. With a spectral classification of G8V, it is similar in size and temperature to our Sun, albeit slightly smaller and cooler [4]. Owing to its proximity and sun-like characteristics, τ

Ceti has been an object of interest for astronomers because of its 4 confirmed and 3 theoretical exoplanets [1].

I present a Python-based numerical model that simulates the orbital motion of these exoplanets using the Yoshida integration method. The model takes into account the gravitational force exerted by the central star, as well as by companion exoplanets, and the initial orbital conditions of each planet.

The Python code provided demonstrates the implementation of the numerical simulation using the NumPy library for efficient numerical computations and the Matplotlib library for visualizing the results. A simulation is conducted for each planet individually, employing a custom *Planet* class to store and manage relevant information such as eccentricity, semi-major axis, and orbital period. The initial conditions with respect to each planet are then calculated as follows:

$$\begin{aligned} x_0 &= r(1-e), \\ y_0 &= 0, \\ v_{x_0} &= 0, \\ v_{y_0} &= \sqrt{\frac{GM}{r}\frac{1+e}{1-e}} \end{aligned}$$

where r is the semi-major axis of the orbit, e is the eccentricity of the orbit, M is the mass of τ Ceti, and G is Newton's gravitational constant. The initial conditions for τ Ceti and its exoplanets are provided by *Color Difference Makes a Difference: Four Planet Candidates around τ Ceti* by Feng et al. [1] and the *NASA Exoplanet Archive* [4].

The Yoshida integration method, a fourth-order symplectic integrator, is employed to maintain numerical stability and accuracy over long time spans. It calculates the change in x and y coordinates, as well as the acceleration in the x and y directions, and then calculates the new position and velocity at some instance $t + \Delta t$ via a function called 'rhs' (see *Section 3* and *Section 7* for more details).

The function receives two input parameters: a 4-element array called 'vector,' which contains the current position (x, y) and velocity (u, v) of the planet; and a dictionary called 'planet_positions,' which contains the positions of the other planets in the system.

The rhs function first extracts the position (x, y) and velocity (u, v) components from the input 'vector' array. It then calculates the radial distance (r) between the planet and the central star using the Pythagorean theorem. Next, it initializes the time derivatives of position ($\frac{dx}{dt}$, $\frac{dy}{dt}$) to the corresponding velocity components (u, v). Finally, a 4-element array containing the time derivatives of position ($\frac{dx}{dt}$, $\frac{dy}{dt}$) and velocity ($\frac{du}{dt}$, $\frac{dv}{dt}$) for the given planet is returned.

To compute the time derivatives of velocity ($\frac{du}{dt}$, $\frac{dv}{dt}$), the function initially considers the gravitational force exerted by the central star. It then iterates through the 'planet_positions' dictionary to include the gravitational forces exerted by the other planets in the system, returning the computed time derivatives of position and velocity as a 4-element array.

The differential equations of motion are as follows:

$$\begin{aligned} \frac{dx}{dt} &= u, \\ \frac{dy}{dt} &= v, \\ a_x = \frac{du}{dt} &= -\frac{GMx}{r^3}, \\ a_y = \frac{dv}{dt} &= -\frac{GMy}{r^3} \end{aligned}$$

The results of the simulation are visualized in a two-dimensional plot displaying the orbits of the exoplanets in the τ Ceti system, along with the position of the central star. Additionally, the model computes the error in the final positions of the planets compared to their initial positions, providing a measure of the accuracy of the simulation.

This paper aims to provide a comprehensive analysis of the orbital dynamics of the τ Ceti exoplanetary system, as well as a detailed description of the numerical model and its implementation in Python. The results obtained through the simulation can be used to further understand the dynamical behavior of exoplanetary systems and provide a foundation for future studies in this field.

3 The Fourth-Order Yoshida Integration Method

The fourth-order Yoshida integration method is a symplectic integration technique used for solving Hamiltonian systems, such as those encountered in celestial mechanics and molecular dynamics. This method is part of a family of higher-order composition methods that offer increased accuracy and stability compared to traditional numerical integration schemes, such as the Euler or Verlet methods. In this section, an overview of the fourth-order Yoshida integration method and a discussion of its key features and benefits is provided.

Symplectic integrators are designed to preserve the geometric properties of Hamiltonian systems, thereby ensuring the long-term stability of the numerical solution. The fourth-order Yoshida integrator achieves this by combining lower-order symplectic integrators in a manner that results in a higher-order method [2]. The Yoshida method derives its name from its inventor, Haruo Yoshida, who developed it in 1990.

The fourth-order Yoshida method is a composition of coefficients c_i and d_i, as defined below:

$$
\begin{aligned}
c_1 &= \frac{w_1}{2}, \\
c_4 &= \frac{w_1}{2}, \\
c_2 &= \frac{w_0 + w_1}{2}, \\
c_3 &= \frac{w_0 + w_1}{2}, \\
d_1 &= w_1, \\
d_3 &= w_1, \\
d_2 &= w_0,
\end{aligned}
$$

where

$$
\begin{aligned}
w_0 &= -\frac{\sqrt[3]{2}}{2 - \sqrt[3]{2}}, \\
w_1 &= \frac{1}{2 - \sqrt[3]{2}}.
\end{aligned}
$$

Given an initial state, the fourth-order Yoshida method computes the updated state by applying a desired sequence of operations. In this method, a higher-order symplectic integrator is constructed by composing simpler symplectic integrators (e.g., 2^{nd} order). The method uses this series of coefficients to weight each of the simpler integrators in order to construct a more accurate overall solution. In this 4th order Yoshida method, there are four coefficients for the time steps (c_1, c_2, c_3, and c_4) and three corresponding coefficients for the order in which the Hamiltonian components are integrated (d_1, d_2, and d_3).

These coefficients are designed to cancel out errors of up to 4th order in the Taylor series expansion of the solution [2], leading to a more accurate and stable numerical integration scheme when compared to second-order methods, such as the Verlet method. Further benefits of the fourth-order Yoshida integration method include:

- Long-term stability: Symplectic integrators, such as the Yoshida method, preserve the energy and phase-space volume of the system, ensuring that the numerical solution remains stable over long time periods.
- Applicability: The Yoshida method is applicable to a wide range of Hamiltonian systems, including celestial mechanics and molecular dynamics problems, making it a versatile tool for numerical simulations.

In summary, the fourth-order Yoshida integration method provides an efficient and accurate approach for solving Hamiltonian systems, making it well-suited for the study of celestial mechanics and other applications. Its higher-order accuracy and long-term stability make it a valuable tool for understanding the dynamics of complex systems.

4 Sequence of Operations

The following table presents the sequence of operations for the fourth-order Yoshida integration method in more detail:

$$
\begin{aligned}
x_i^1 &= x_i + c_1\, v_i\, \Delta t, \\
v_i^1 &= v_i + d_1\, a(x_i^1)\, \Delta t, \\
x_i^2 &= x_i^1 + c_2\, v_i^1\, \Delta t, \\
v_i^2 &= v_i^1 + d_2\, a(x_i^2)\, \Delta t, \\
x_i^3 &= x_i^2 + c_3\, v_i^2\, \Delta t, \\
v_i^3 &= v_i^2 + d_3\, a(x_i^3)\, \Delta t, \\
x_{i+1} \equiv x_i^4 &= x_i^3 + c_4\, v_i^3\, \Delta t, \\
v_{i+1} \equiv v_i^4 &= v_i^3.
\end{aligned}
$$

In this table, x_i and v_i represent the position and velocity of the system at time step i, while Δt is the time step size. The function $a(x)$ calculates the acceleration at position x. The superscripts denote intermediate states in the integration process, with the final updated state given by x_{i+1} and v_{i+1} [3].

5 Solar System Testing

To validate the performance and accuracy of this numerical model, a series of simulations for planets in our solar system, including Venus, Earth, Mars, Jupiter, and Saturn, was conducted. By comparing the numerical results with the known orbital parameters of these planets, an assessment of the effectiveness of the Yoshida integration method in solving the equations of motion for celestial bodies is made.

Figure 1 shows the orbits of Venus, Earth, Mars, Jupiter, and Saturn obtained from the simulation. The central yellow dot represents the Sun, and the orbital paths of the planets are plotted around it. Table 1 summarizes the error in the final positions of the planets compared to their initial positions, providing a measure of the accuracy of the simulation.

Planet	**Step Size (s)**	**Error (AU)**
Venus	100	0.00036741
Earth	100	0.00027533
Mars	100	0.00023688
Jupiter	1000	0.0010195
Saturn	1000	0.00077095

Table 1: Error in the final positions of solar system planets.

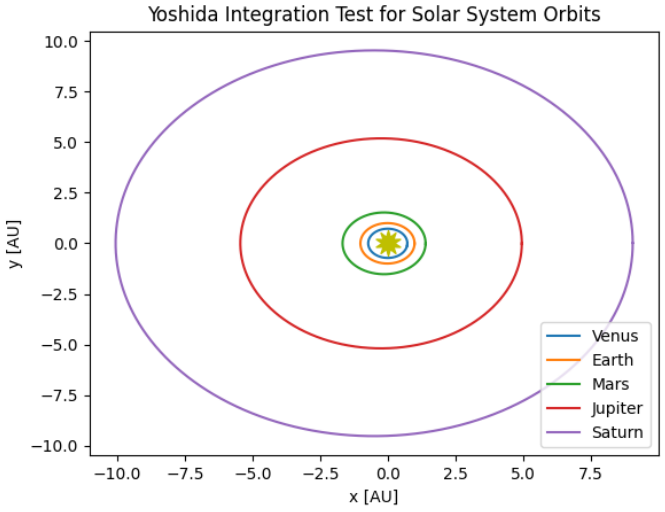

Figure 1: Yoshida Integration Test for Venus, Earth, Mars, Jupiter, and Saturn Orbits.

The results obtained from the solar system testing demonstrate the ability of the Python-based numerical model to accurately simulate the motion of celestial bodies using the Yoshida integration method. The low error values indicate that the model is capable of maintaining numerical stability and accuracy over long time spans, making it suitable for studying the orbital dynamics of exoplanetary systems such as the τ Ceti system.

6 Simulation Results

Figure 2 displays the orbits of the exoplanets, where the dotted plots represent the unconfirmed and the solid plots represent the confirmed. In the simulation results, it can be observed that the fourth-order Yoshida integration method provides a relatively low error for the planetary orbits. With a time step of 100s for each planet the errors range from approximately 1.46×10^{-4} AU to 5.75×10^{-4} AU. Among the planets, planet f has the lowest error value of 1.46×10^{-4} AU, while planet b has the highest error value of 5.75×10^{-4} AU. These results suggest that the numerical method used in the simulation is effective in maintaining accuracy throughout the simulated time period. Although the errors are not identical for all planets, they remain small in this case, indicating a reliable representation of the planetary orbits.

Furthermore, a graph of error as a function of time-step size Δt is included. It can clearly be seen that as the size of time steps increases, the error also increases. The rate at which the error increases is different for each planet

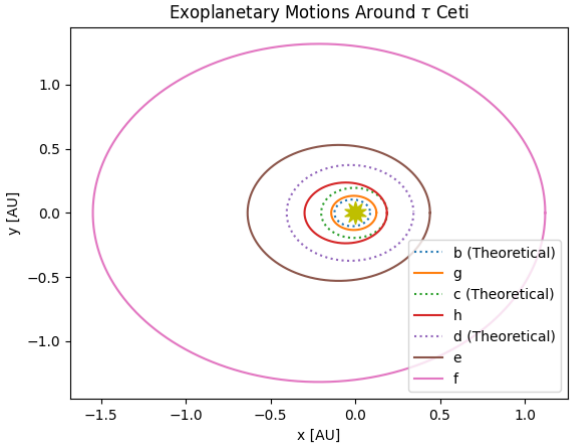

Figure 2: Simulation of the planetary orbits.

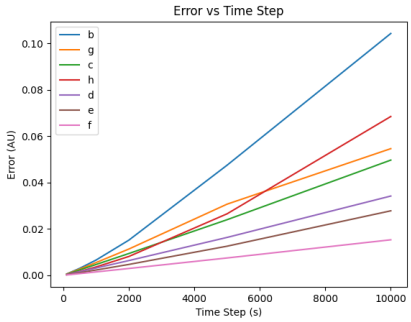

Figure 3: Error of planets at various time steps.

because each planet has a different periodicity. For example, Planet b has a period of 13.965 days and Planet f has a period of 636 days, so a time step of $\Delta t = 1000$s for Planet b leads to approximately only 1,200 data points, while the same time step for Planet f leads to about 55,000 data points for one respective orbit. It is further significant to note that all errors converges towards 0 with smaller time-steps. Future improvements to the simulation could focus on further refining the integration method or adjusting parameters to reduce these errors, but the current results already provide valuable insights into the behavior of the planetary orbits in this stellar system.

7 Comparison Against the Euler Method

The simulation was repeated using the Euler Method of Integration. A table of errors at a time step of 500s is shown below:

Method	Euler	Yoshida	
Planet	Error (AU)	Error (AU)	Difference (AU)
b	0.006649	0.006606	0.000042
g	0.005595	0.005510	0.000085
c	0.004794	0.004704	0.000091
h	0.003687	0.003614	0.000073
d	0.003228	0.003156	0.000072
e	0.002351	0.002295	0.000056
f	0.001507	0.001469	0.000038

Table 2: Comparison of errors between Euler and Yoshida methods and their differences

The results, as shown in Table 2, indicate that the Yoshida method consistently yields lower errors compared to the Euler method across all examined planets. The difference in errors ranges from 0.000038 AU for planet f to 0.000091 AU for planet c, demonstrating a notable improvement in the accuracy of the Yoshida method. This finding supports the hypothesis that higher-order numerical integrators, such as the Yoshida method, are more effective in capturing the dynamics of celestial bodies. Consequently, researchers and astronomers aiming for increased precision in their planetary motion simulations should consider employing the Yoshida method over the simpler Euler method.

8 Limitations of the Method

While the fourth-order Yoshida Method is an effective numerical integration scheme for simulating dynamical systems, it does have its limitations. One of the most significant drawbacks is the method's sensitivity to the choice of time step. When the time step is too large, the method can become unstable, leading to inaccurate or divergent results (as seen by Figure 3). Conversely, choosing a time step that is too small can lead to excessive computational cost and slow convergence rates. Therefore, selecting an optimal time step for a given problem can be a challenging task, requiring thorough testing and possibly adaptive time step control.

Another limitation of the fourth-order Yoshida Method is its order of accuracy compared to other higher-order methods. As a fourth-order method, it can provide accurate results for moderately smooth functions, but may not be suitable for problems with rapidly varying or highly oscillatory solutions. In such cases, the use of higher-order methods or specialized techniques tailored to the specific problem may be necessary to achieve accurate results. Additionally, the Yoshida Method is designed for solving Hamiltonian systems, which means

its applicability may be limited in other types of problems that do not exhibit the same structure.

9 Conclusions

In conclusion, the simulation results demonstrate that the fourth-order Yoshida integration method is effective in maintaining accuracy for the planetary orbits, with errors remaining relatively low for all planets. The variation in error values can be attributed to the different periodicities of each planet, and the analysis reveals that smaller time steps lead to a convergence of errors towards zero. These findings provide valuable insights into the behavior of planetary orbits in the solar system and establish a solid foundation for future improvements, such as refining the integration method or adjusting parameters to further reduce errors. By continuously enhancing the simulation model, researchers can deepen their understanding of star system dynamics and contribute to the field of celestial mechanics.

10 Python Code

This section contains the raw Python code used in this paper. It is encouraged to all those reading to learn from and build upon the provided code.

```
import numpy as np
import matplotlib.pyplot as plt

G = 6.67e-11
Msun = 1.989e30
Mearth = 5.972e24
M = 0.783 * Msun
AU = 1.496e11
DAY = 24*60*60

w0 = -np.cbrt(2)/(2-np.cbrt(2))
w1 = 1/(2-np.cbrt(2))

c1 = w1/2
c4 = w1/2
c2 = (w0+w1)/2
c3 = (w0+w1)/2
d1 = w1
d3 = w1
d2 = w0

class Planet:
```

```
def __init__(self,name,e,sma,T,mass):

    self.name = name
    self.e = e
    self.sma = sma
    self.T = T
    self.mass = mass
    self.history = [self.initial_conditions()]

def initial_conditions(self):
    return np.array([self.sma*(1-self.e),0,0,
        -np.sqrt((G*M)/self.sma * ((1+self.e)/(1-self.e)))])

def rhs(self, vector, planet_positions):
 x, y, u, v = vector

 r = np.sqrt(x**2 + y**2)

 dxdt = u
 dydt = v

 dudt = (-G * M * x) / r**3
 dvdt = (-G * M * y) / r**3

 for planet, position in planet_positions.items():
    if planet != self:
        dx = x - position[0]
        dy = y - position[1]
        dr = np.sqrt(dx**2 + dy**2)
        dudt -= (G * planet.mass * dx) / dr**3
        dvdt -= (G * planet.mass * dy) / dr**3

 return np.array([dxdt, dydt, dudt, dvdt])

def yoshida(self, tmax, dt, planets):
    t = 0.0
    history = [self.initial_conditions()]
    times = [t]

    while t < tmax:

     if t + dt > tmax:
        dt = tmax - t

     state_old = history[-1]
```

```
        planet_positions = {planet: planet.history[-1][:2] for planet in planets}

        dvec = self.rhs(state_old, planet_positions)

        x1 = state_old[0] + c1 * dvec[0] * dt
        y1 = state_old[1] + c1 * dvec[1] * dt
        vx1 = state_old[2] + d1 * dvec[2] * dt
        vy1 = state_old[3] + d1 * dvec[3] * dt

        mid_state = np.array([x1, y1, vx1, vy1])
        dvec_mid = self.rhs(mid_state, planet_positions)

        x2 = x1 + c2 * dvec_mid[0] * dt
        y2 = y1 + c2 * dvec_mid[1] * dt
        vx2 = vx1 + d2 * dvec_mid[2] * dt
        vy2 = vy1 + d2 * dvec_mid[3] * dt

        next_state = np.array([x2, y2, vx2, vy2])
        dvec_next = self.rhs(next_state, planet_positions)

        x3 = x2 + c3 * dvec_next[0] * dt
        y3 = y2 + c3 * dvec_next[1] * dt
        vx3 = vx2 + d3 * dvec_next[2] * dt
        vy3 = vy2 + d3 * dvec_next[3] * dt

        end_state = np.array([x3, y3, vx3, vy3])
        dvec_end = self.rhs(end_state, planet_positions)

        xp1 = x3 + c4 * dvec_end[0] * dt
        yp1 = y3 + c4 * dvec_end[1] * dt
        vxp1 = vx3
        vyp1 = vy3

        state_new = np.array([xp1, yp1, vxp1, vyp1])

        t += dt

        times.append(t)
        history.append(state_new)

       return times, history


def coords(self,dt,planets):

        self.times, self.history = self.yoshida(self.T,dt,planets)
```

```
        self.x_pos = [q[0]/AU for q in self.history]
        self.y_pos = [q[1]/AU for q in self.history]

        return self.x_pos, self.y_pos

    def error(self):

     R_orig = np.sqrt(self.x_pos[0]**2 + self.y_pos[0]**2)
     R_new = np.sqrt(self.x_pos[-1]**2 + self.y_pos[-1]**2)
     e = np.abs(R_new - R_orig)

     return e


b = Planet("b",0.16,0.105*AU,13.965*DAY,2*Mearth)
g = Planet('g',0.06,0.133*AU,20*DAY,1.75*Mearth)
c = Planet('c',0.03,0.195*AU,35.362*DAY,3.1*Mearth)
h = Planet('h',0.23,0.243*AU,49.41*DAY,1.83*Mearth)
d = Planet('d',0.08,0.374*AU,94.11*DAY,3.6*Mearth)
e = Planet('e',0.18,0.538*AU,162.87*DAY,3.93*Mearth)
f = Planet('f',0.16,1.334*AU,636*DAY,3.93*Mearth)


ax = plt.subplot(111)
ax.set_title(r"Exoplanetary Motions Around $\tau$ Ceti")
ax.set_xlabel(r"x [AU]")
ax.set_ylabel(r"y [AU]")
ax.scatter([0], [0], marker=(10,1), color="y", s=250)


Planets = [b,g,c,h,d,e,f]


for p in Planets:

    x_pos, y_pos = p.coords(100,Planets)

    if p.name == "b" or p.name == "c" or p.name == "d":
        ax.plot(x_pos,y_pos,":",label = f"{p.name} (Theoretical)")
    else:
       ax.plot(x_pos,y_pos,label=p.name)

    print(f'Planet = {p.name}, Error = {p.error()}')

ax.legend()
```

```
plt.show()


time_steps = [100, 500, 1000, 2000, 5000, 10000]

errors = {planet.name: [] for planet in Planets}

for dt in time_steps:
    for p in Planets:
        x_pos, y_pos = p.coords(dt, Planets)
        error = p.error()
        errors[p.name].append(error)

fig, bx = plt.subplots()
bx.set_title(r"Error vs Time Step")
bx.set_xlabel(r"Time Step (s)")
bx.set_ylabel(r"Error (AU)")

for planet, error_list in errors.items():
    bx.plot(time_steps, error_list, label=planet)

bx.legend()
plt.show()
```

References

1. Feng et al., 2017. *Color Difference Makes a Difference: Four Planet Candidates around τ Ceti.* The Astronomical Journal, 154(4).

2. Forest et al., 1991. *Application of the Yoshida-Ruth Techniques to Implicit Integration, Multi-Maps Explicit Integration and to Taylor Series Extraction.* Lawrence Berkely National Laboratory, LBL-30616.

3. *Leapfrog Integration.* Retrieved from `https://en.wikipedia.org/wiki/Leapfrog_integration`

4. *NASA Exoplanet Archive.* Retrieved from `https://exoplanetarchive.ipac.caltech.edu/overview/tau%20cet%20f`

5. Zingale, M. *An Introduction to Computational Astrophysics.* Retrieved from `https://zingale.github.io/computational_astrophysics/intro.html`