

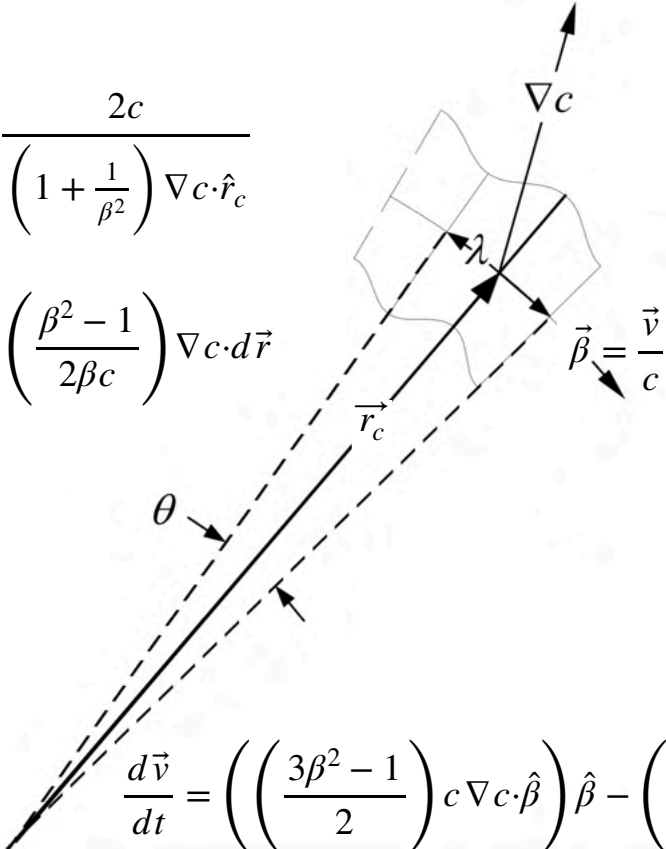
N-Body Dynamics via Curvature of Wave Propagation: A Flat-Space Framework Matching GR Solar-System and EHT Constraints

Morris G. Anderson
morrisganderson@gmail.com

Abstract

The curvature of wave propagation method (CWPM) models gravity in flat three-dimensional Euclidean space, capturing both weak- and strong-field regimes. Gravitational effects arise solely from the spatial gradient of the speed of light, wave-particle duality, and the resulting curvature of wave propagation, which together yield an explicit, closed-form, velocity-dependent acceleration vector that is analytically more compact and natively simpler to implement than the complex Parameterized Post-Newtonian (PPN) expansion series. CWPM N-body simulations reproduce the classical solar-system tests of general relativity—including Mercury’s perihelion advance, light deflection, and Shapiro delay—while executing up to five times faster than PPN integrations. A 15-body simulation confirms Mercury’s precession and demonstrates convergence under time-step refinement. Independent implementation and testing by Kuijpers (2025) in his pre-existing Solar System Simulator confirms CWPM matches GR-based PPN DE405-referenced deviations to within <30 km RMS over 600 years, while running $\sim 5\times$ faster. Although CWPM places the photon orbital radius at the Schwarzschild radius $r_s = 2GM/c^2$ (rather than $1.5 r_s$ in GR), it predicts a black-hole shadow diameter of $2er_s$, only $\sim 4.6\%$ larger than the GR value—well within Event Horizon Telescope uncertainties for both M87* and Sgr A*. Included MATLAB scripts enable students and researchers to compute solar-system orbits, light propagation, black-hole shadows, star-cluster dynamics, and deep-space trajectories using a unified framework based on a single explicit equation of motion, offering an accessible, computationally efficient learning tool for physics education and independent research.

This manuscript includes minor rewording to the Author Declarations and corrects typographical errors from the earlier ResearchGate preprint uploaded on January 16, 2026 (<https://doi.org/10.13140/RG.2.2.30735.04004>).



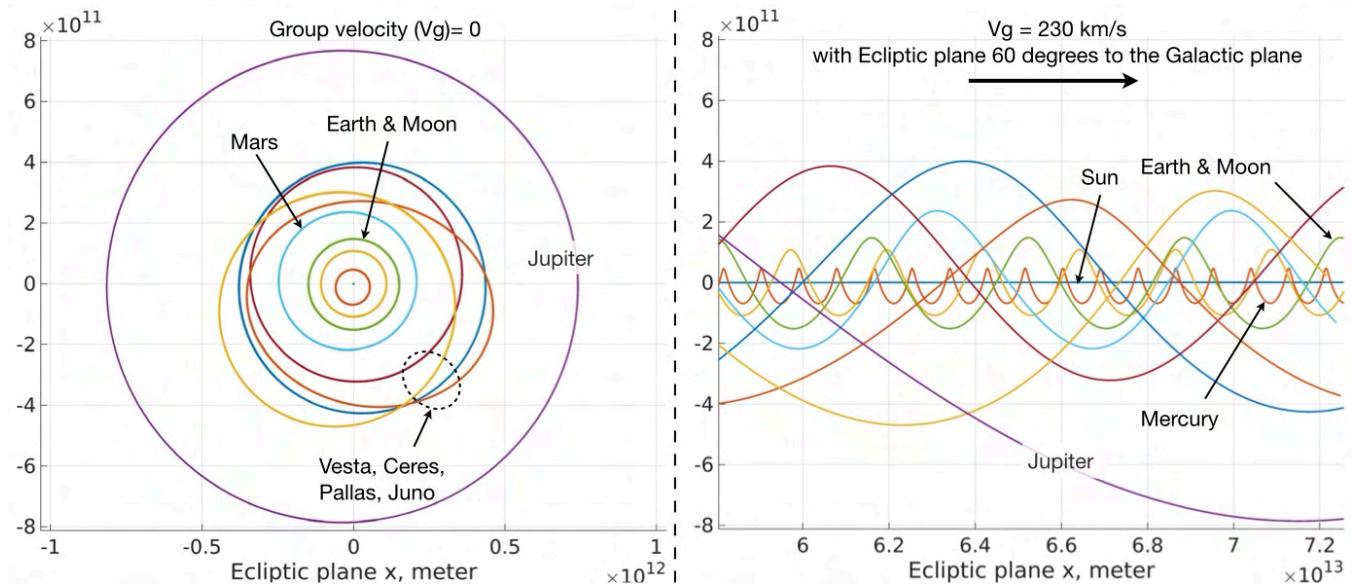
$$r_c = \frac{2c}{\left(1 + \frac{1}{\beta^2}\right) \nabla c \cdot \hat{r}_c}$$

$$d\beta = \left(\frac{\beta^2 - 1}{2\beta c}\right) \nabla c \cdot d\vec{r}$$

$$\frac{d\vec{v}}{dt} = \left(\left(\frac{3\beta^2 - 1}{2}\right) c \nabla c \cdot \hat{\beta}\right) \hat{\beta} - \left(\left(\frac{\beta^2 + 1}{2}\right) c \nabla c \cdot \hat{r}_c\right) \hat{r}_c$$

The curvature of wave propagation (top-left) and resulting acceleration vector (bottom)

For definitions of all symbols, see Section 5 (Symbols and Constants)



Typical N-body Simulations Generated using CWPM MATLAB code (see Example 1 on page 17)

1. Introduction

In astronomy, N-body simulations use numerical methods to integrate the motion of two or more gravitational bodies. There is significant interest in simplifying and improving the accuracy of these methods to understand the nature of gravity and the dynamics of our Solar System. The works by Park, R. S. *et al.* (2017, 2021), Pogossian, S. P. (2022), and Clemence, G. M. (1947) illustrate the associated data, numerical methods, and challenges.

A primary test for verifying the accuracy of an N-body simulation is to compare predicted results with observational precession data for Mercury's orbit (the typical way to represent Mercury's precession is by $\dot{\varpi} = \dot{\omega} + \dot{\Omega} \cos i$ as described in Appendix A on page 42). Derived precession results from published information for a best fit of Keplerian elements to observational data include:

$\dot{\varpi} = 574.10''/\text{century}$ *The Relativity Effect in Planetary Motions* (Clemence, G. M., 1947)

$\dot{\varpi} = 576.90''/\text{century}$ *Explanatory Supplement to the Astronomical Almanac* by Seidelmann, P. K. (1992 and 2006, page 316, Table 5.8.1)

$\dot{\varpi} = 577.12''/\text{century}$ *Orbital Ephemerides of the Sun, Moon, and Planets* (Standish and Williams, 2006 Table 8.10.3) and *The Explanatory Supplement to the Astronomical Almanac*, 3rd edition, page 340 (Urban S. E. and Seidelmann P. K., 2012)

By contrast, Solar System N-body simulations based on Newtonian gravitational theory predict Mercury's orbit to precess at approximately 532.1''/century (Pogossian, S. P., 2022). According to Einstein's theory of general relativity, the discrepancy between the Newtonian prediction and the observational results can be explained by the curvature of spacetime. However, simulating the Solar System with a full N-body model based directly on the local properties of space as described by general relativity is not yet computationally feasible. Consequently, NASA's Jet Propulsion Laboratory (JPL) uses a complex Parameterized Post-Newtonian (PPN) method to simulate the Solar System. This approach, which approximates general relativity in weak gravitational fields (Folkner, W. M. *et al.*, 2014), estimates Mercury's total perihelion precession at $\dot{\varpi} = 575.31''/\text{century}$ (Park, R. S. *et al.*, 2017).

2. Objective

Develop a simple and accurate N-body gravity simulation method based on the curvature of wave propagation.

3. Gravity N-body Simulation Challenges

- Classical Newtonian mechanics fails to accurately predict several gravitational phenomena, including the natural frequency shift of matter, the redshift and deflection of light, the Shapiro time delay, and planetary precession.
- The NASA JPL PPN code, which achieves high precision in ephemeris calculations, is not publicly available. The PPN framework itself, as implemented in DE430 ([Folkner, W. M. et al., 2014](#)) and DE440 ([Park, R. S. et al., 2021](#)), is technically demanding and often requires simplification for practical use ([Pogossian S. P., 2025](#)).
- Nico Kuijpers has developed a publicly available Solar System Simulator ([GitHub repository](#)) that incorporates an option to apply PPN methodology to approximate general relativity. The simulator demonstrates accuracy comparable to the JPL PPN code but also highlights a significant drawback: PPN equations demand substantially greater computational resources than Newtonian mechanics.
- Full general relativity N-body simulations that directly solve Einstein's field equations are extremely challenging. They are limited to only a few compact objects—such as black holes ([Ficarra and Lousto, 2024](#)) or neutron stars—and require supercomputer-scale numerical relativity codes. Examples applicable to solar-system-type scenarios are not found.

4. Solution

The curvature of wave propagation method (CWPM), based on wave-particle duality, provides a framework for determining the influence of the local properties of space on the motion of each body. This is accomplished by extending the work documented in *The Curvature of Motion in a Gravitational Field* ([Anderson, M. G., 2021](#)), from a single governing body to an N-body simulation.

Recent independent full-Solar-System benchmarks by [Kuijpers, N., \(2025\)](#) show that CWPM closely matches GR-based PPN trajectories, with <30 km RMS difference in their DE405-referenced deviations over 600 years, while executing $\sim 5\times$ faster.

The governing equations are straightforward to implement, and the included MATLAB scripts show strong agreement with observational data for orbits, light deflection, Shapiro time delay, planetary precession, and black hole shadow diameter. These scripts can be adapted to study gravitational and motion effects on matter's natural frequency and redshift, as derived in *Time, Matter, and Gravity* ([Anderson, M. G., 2004](#)) and demonstrated by Anderson ([2021](#), Example 2) with GPS satellite data.

Author's Note

General relativity has passed numerous precision tests across Solar System dynamics, binary pulsars, and gravitational wave observations. The curvature of wave propagation method (CWPM) is presented here as a simpler, flat-space framework that already reproduces several key observational results and may, with further theoretical development, be extendable to the full range of gravitational phenomena.

The purpose of this work is not to discredit any existing theory of gravitation. Rather, the aim is to provide a clear and accessible description of gravitational motion based on the curvature of wave propagation, and to demonstrate how this framework can be applied to practical N-body simulations. Comparisons with general relativity are included solely to indicate where CWPM aligns with, or departs from, established predictions. All equations are presented in a readable form suitable for students and researchers with a basic background in calculus and physics, and the accompanying MATLAB scripts are provided to support independent exploration and verification.

5. Symbols and Constants

Symbol	Units	Description
$\vec{\beta}$	none	Velocity / local speed of light
$\hat{}$	none	The hat over a variable, such as $\hat{\beta}$, indicates a unit vector
c	m/s	Speed of light at the point of evaluation
∇c	1/s	Gradient in the speed of light
λ	meters	Wavelength
m	kg	Total Matter
m_o	kg	Rest Matter, which includes all internal energy, is usually referred to as rest mass.
M	kg	Gravitational Matter. This is equal to the rest mass of the body.
$\vec{r}$	meters	Position vector
r_c	meters	Radius of curvature
$\vec{v}$	m/s	Velocity vector

Constant	Description
$c_s = 299\,792\,458 \frac{m}{s}$	Speed of light at the standard location
$G = 6.6743 \times 10^{-11} \frac{m^3}{kg \cdot s^2}$	Newtons universal gravitational constant

6. Theory

The CWPM derivation is documented in *Time, Matter, and Gravity* ([Anderson, M. G., 2004](#)), hereafter referred to as [TMG](#). CWPM reformulates motion, inertia, and gravitation in terms of wave propagation rather than forces or spacetime curvature. The method treats TIME as an axiomatic concept, with all measured quantities arising from changes or separations in TIME relative to a defined standard. Within this framework, gravitational and inertial behavior, including equivalence-principle behavior, emerges naturally from spatial variations in wave propagation.

Throughout this paper, the term CWPM refers, depending on context, either to the underlying theoretical framework developed in TMG or to the practical curvature of wave propagation method used for N-body simulations.

6.1 Position Definition of TIME

In this framework, **TIME (capitalized)** is defined as **position**, corresponding to where things are. TIME is not defined operationally by clocks or measurement procedures and does not flow or evolve; it serves as a coordinate-like quantity. TIME has the following characteristics:

- **TIME** is equal to position and corresponds to where things are.
- **TIME Past** is where you have been and the way things were.
- **TIME Present** is where you are and the way things are.
- **TIME Future** is where you will be and the way things will be.

A change in **TIME** (commonly referred to as *time* in lowercase) corresponds to a change in position. Time, therefore, is a relational quantity, not a fundamental entity. A separation in **TIME** (commonly referred to as *length*) corresponds to a separation in position.

This definition of TIME does not involve a flow of time, is independent of measurement, and is not tied to any operational procedure. TIME is a **non-operational** positional quantity, while SI standards are **operational realizations** that specify how clocks, wavelengths, and light-based measurements are

implemented in practice. Section 2 of TMG demonstrates that these operational SI definitions for time, length, and mass arise naturally within the relational structure of TIME.

In the context of SI standards, the **clock rate** is defined as the ratio of wavelength to wave velocity for light emitted by the cesium atom.

6.2 Standard Location

The standard location is a reference point where the units of time, length, and matter are defined in terms of the local properties of space and light. This location is chosen to suit the needs of the analysis and does not necessarily coincide with the center of the coordinate frame. Framing the definitions in this way allows measured quantities to vary with position, motion, or velocity while maintaining internal consistency and preserving local invariance.

6.3 Velocity and Measurement

Velocity is defined as the ratio of a change in position of matter, such as a particle or wave, to the corresponding change in position of light at the standard location, mathematically as:

$$\vec{v} = \frac{d\vec{x}}{dt}$$

where $d\vec{x}$ is the change in Euclidean position (in units of meters, vector), and dt is the corresponding change in the position of light at the standard location (in units of seconds, scalar).

Velocity is thus a measured quantity derived from wave propagation. Therefore, phenomena such as time dilation and length contraction are interpreted as changes in the wavelength, natural frequency of matter, or clock rate. Furthermore, clock synchronization reflects local physical conditions rather than defining TIME itself.

This framework is different from [Einstein's](#) operational definition of time, which defines simultaneity through synchronized clocks using light signals. In his 1905 paper on special relativity, he states:

“Thus with the help of certain imaginary physical experiments we have settled what is to be understood by synchronous stationary clocks located at different places, and have evidently obtained a definition of ‘simultaneous,’ or ‘synchronous,’ and of ‘time.’ The ‘time’ of an event is that which is given simultaneously with the event by a stationary clock located at the place of the event, this clock being synchronous, and indeed synchronous for all time determinations, with a specified stationary clock.”

6.4 The Speed of Light

Because CWPM builds upon the Position Definition of Time and the wave-like nature of matter, understanding the role of the speed of light is essential to this approach.

The behavior of light in a gravitational field can be clarified by Shapiro’s experimental tests of general relativity. These tests show that gravitational potential influences how light propagates. In his seminal paper *Fourth Test of General Relativity* ([Shapiro, I. I., 1964](#)), he states:

“... according to the general theory, the speed of a light wave depends on the strength of the gravitational potential along its path ...”

This prediction was experimentally confirmed in *Fourth Test of General Relativity: Preliminary Results*, ([Shapiro, I. I., 1968](#)), where Shapiro reported:

“The proposed experiment was designed to verify the prediction that the speed of propagation of a light ray decreases as it passes through a region of increasing gravitational potential... The most

reliable of these data agree, on average, with the excess-delay predictions of general relativity to well within the experimental uncertainty of $\pm 20\%$.”

These statements may appear contradictory at first glance, but the distinction becomes clear once the measurement method is specified. [Einstein](#) wrote:

“In the general theory of relativity, space and time cannot be defined in such a way that differences of the spatial co-ordinates can be directly measured by the unit measuring rod, or differences in the time co-ordinate by a standard clock.”

According to general relativity, the locally measured speed of light remains constant regardless of an observer’s velocity or position within a gravitational field when it is evaluated using a standard clock and measuring rod. This consistency arises because rods and clocks are themselves affected by motion and gravitational potential: rods change length, and clocks shift their natural frequency in precisely the amounts required to preserve the measured constancy of light’s speed. Thus, although the coordinate speed of light varies within a gravitational field, the time-of-flight method consistently yields the same result for the local measured speed of light in terms of (standard rod units)/(standard clock cycles). In general relativity terminology, this distinction is expressed as the constancy of the *local speed of light* versus the variation of the *coordinate speed of light* in curved spacetime. [TMG](#) shows that this distinction arises naturally from the wave-like characteristics of matter. In CWPM, the variation in c reflects a physical change in the propagation properties of space rather than a coordinate effect.

6.5 Space and Matter

CWPM rests on the concept of matter as the universal substance of existence, encompassing light, energy, fields, particles, and all physical phenomena. From this arises the postulate that *space and matter are mutually dependent: where there is space, there is matter, and where there is matter, there is space. Neither can exist independently.*

CWPM therefore aligns with a view of space as an indivisible, fluid-like medium whose properties govern electromagnetic interactions, gravitational behavior, and wave propagation. This interpretation differs conceptually from the geometric spacetime framework of general relativity, even though both frameworks preserve the same underlying equivalence-principle behavior.

6.6 The Wave Nature of Matter and Rest Mass

The CWPM framework is built on equations of motion that apply uniformly to both light and particles, conserving matter and momentum even when the speed of light varies with position (see TMG Section 6). A central postulate is that all particles are composed of electromagnetic standing waves, and that interactions between particles occur through the exchange of electromagnetic momentum. Within this wave-based description, CWPM provides a clear definition of rest mass as the internal momentum of the standing waves that constitute a particle, divided by the speed of light at the standard location. From this foundation, the framework naturally yields the wavelength of interaction between particles in terms of momentum, as well as the wavelength of interaction with space—giving a direct derivation of Louis de Broglie’s wavelength relation (see TMG Section 6.3) based on Einstein’s photon concept of light.

6.7 Position-Dependent Speed of Light

In TMG, all physical quantities are referenced to the standard location, meaning that measurements of mass, wavelength, and frequency are defined relative to the properties of space at that location. As a result, the speed of light is treated as a variable even though the time-of-flight measurement method remains invariant. Within CWPM, the local value of c is not constant but depends on the physical properties of space, which in turn depend on the surrounding distribution of rest mass.

TMG shows that rest mass is formed from electromagnetic standing waves, and that the internal momentum, charge, and atomic transition energies of matter depend directly on the local value of the speed of light. These relationships allow us to derive how c must vary in order for matter to maintain consistent internal structure as it moves through space.

When this wave-based description is combined with the gravitational force expression modified to depend explicitly on rest mass, the result is a direct derivation of the natural logarithmic variation of the speed of light with respect to a governing body (TMG equation 4-11).

This spatial variation in c provides the physical mechanism that shapes both the internal characteristics of matter and the curvature of motion used in the CWPM N-body method. It forms the essential link between the wave-based structure of matter and the gravitational behavior of bodies, enabling CWPM to describe motion through wave propagation rather than force or geometric curvature.

6.8 Curvature of Motion

Motion through regions where wave-propagation properties vary spatially results in curved trajectories (see cover page and TMG Eq. 7-9). In CWPM, this curvature arises directly from the way changes in the local speed of light influence the propagation of matter's wave characteristics. In this framework, no forces or spacetime curvature are invoked.

Both particles and photons follow paths determined by these propagation gradients, leading naturally to phenomena such as orbital motion, light deflection, signal delay and equivalence-principle behavior. In regions where the gradients are weak, the resulting trajectories reduce smoothly to the Newtonian limit.

6.9 The Influence of a Moving Gravitational Field

To facilitate modeling N-body systems, the following postulate is incorporated into the TMG framework:

The influence of gravity on a body passing through a field is equivalent to the influence of a gravitational field moving through the body.

This postulate ensures that gravitational effects are treated consistently whether the motion arises from the body, the field, or both, and it aligns naturally with the wave-propagation basis of CWPM.

6.10 Acceleration Vector

The acceleration vector (see cover page and derivation in Sections 7.3–7.6) arises directly from spatial gradients in wave-propagation properties. The CWPM formulation yields a closed-form expression involving only the object's velocity, the local speed of light, and its spatial gradient, enabling efficient and stable numerical integration in N-body simulations.

This formulation reproduces Newtonian dynamics in the weak-field limit and remains continuous in strong-field regimes without requiring post-Newtonian expansions, field equations, or correction terms. The same expression applies to all forms of matter, providing a unified description of motion.

6.11 Recovery of Newtonian Gravity in the Weak-Field Limit

An important consistency check for any gravitational method is that it must reproduce classical Newtonian behavior in the weak-field, low-velocity limit. In CWPM, this limit follows directly from TMG Eq. 7-9 and from Equation 14 in Section 7. For tangential or radial velocities with respect to a governing body that are small compared to the speed of light ($v/c \rightarrow 0$), the velocity-dependent wave propagation terms become negligible.

The spatial variation in the speed of light, ∇c , with respect to a single gravitational body, is derived in TMG section 4.1 as equation 4-8 and presented here in vector form as

$$\nabla c = -\frac{2GM}{c_s^2 r^2} c \hat{r}$$

Therefore, in the weak-field limit, the acceleration vector provided by Equation 14 (see section 7.6 on page 12) reduces to

$$\frac{dv}{dt} = -\frac{GM}{r^2} \hat{r}$$

No additional assumptions, force laws, or parameter adjustments are required. Classical Newtonian gravity therefore appears as the natural limiting case of the curvature of wave propagation framework. This shows that CWPM does not replace Newtonian mechanics in weak fields; rather, it nests and extends it in a continuous and physically transparent manner.

6.12 Emergent Equivalence

In CWPM, both motion and gravitation influence matter through changes in wave propagation and natural frequency (see Section 6.6 and TMG sections 5, 6, and 7). Because these effects arise from the same underlying mechanism, inertial and gravitational behavior are inherently linked. As a result, equivalence-principle behavior—stated as ‘the final state, rather than the duration or timing, of a local non-gravitational experiment is unaffected by velocity or position within a gravitational field’—emerges naturally and does not need to be introduced as a separate postulate. The observed agreement with the standard classical tests of general relativity follows directly from the propagation properties of space and the wave characteristics that constitute matter.

6.13 Summary of Theoretical Structure

Within CWPM, gravity is not treated as a force, and TIME is not a measured quantity. Observable gravitational phenomena arise from spatial variations in wave propagation that influence clock rate, measured length, and measured velocity in a self-consistent manner.

CWPM provides a precise definition of rest mass and a direct derivation of the wavelength of matter (de Broglie), giving the framework inherent compatibility between gravitation and wave mechanics.

This framework provides a clear and computationally efficient basis for modeling N-body dynamics across a wide range of physical conditions. By grounding motion and gravitation in wave propagation rather than forces or spacetime curvature, CWPM offers a unified approach that remains consistent from weak-field solar-system dynamics to strong-field environments near compact objects.

7. Numerical Approach and Supporting Equations

This section outlines the N-body numerical method used in the accompanying MATLAB scripts and presents the supporting equations required for implementation. It includes the derivation of the CWPM acceleration vector and the computational steps needed to apply it consistently across multi-body gravitational systems.

The CWPM acceleration vector is fully general; however, the specific gradient-of-space calculation used here reflects the simplifying assumptions introduced in Section 7.2. This approximation is sufficient for the applications considered in this paper, but future work—such as modeling gravitational waves or simulating black-hole mergers—will require a more complete evaluation of the gradient of space.

7.1 The Curvature of Wave Propagation

Figure 1 illustrates the curvature of wave propagation, $\vec{r}_c$, in **Euclidean space** for a body, λ , changing position in a gravitational field. $\vec{r}_c$ is calculated from $\vec{\beta}$ and the local properties of space in terms of the gradient of the speed of light, ∇c . This method, derived in [TMG](#), is valid for all known forms of matter in both weak and strong gravitational fields ([Anderson, M. G., 2004, 2021](#)). Though not yet proven mathematically, the results presented in this paper indicate CWPM is valid for both fixed and moving reference frames.

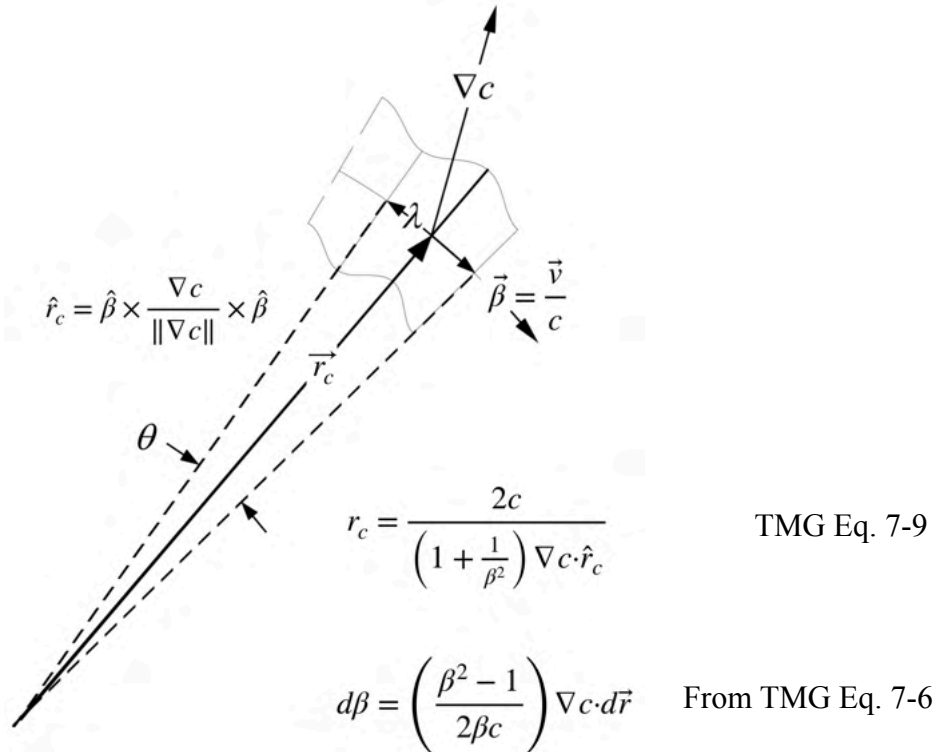


Figure 1 The curvature of wave propagation in a gravitational field.

The equations are evaluated in Euclidean space using meters and seconds defined at the standard location, conventionally realized on Earth for SI units. The Euclidean solution can also be evaluated for reference frames in which the standard location is placed elsewhere. For example, the standard location may be defined as if it were at rest in a frame co-moving with the solar-system barycenter but positioned sufficiently far from the Solar System that its gravitational influence is negligible. The speed of light at the standard location is represented by c_s , and this reference value remains fixed throughout any given simulation.

7.2 Simplifications Made in Creating the MATLAB Scripts

1. Calculate the local gradient of space (∇c) by assuming the rest mass of each body is constant (insignificant for the solar system - see independent verification on page 36).

Note: CWPM separates curvature of motion from gradient of space calculations, allowing users to add influences such as rest-mass variation, solar oblateness, and interstellar matter.

2. The time delay in changes to the properties of space emanating from the motion of a governing body is not accounted for. Hence, the MATLAB script files provided in this paper are most accurate for slow changes in the gravitational gradient, such as those caused by a passing body moving with a velocity much less than the speed of light. Future versions are expected to incorporate the influence of time delay effects.

7.3 Equations of Motion

Calculate the local properties of space for a single governing body with [TMG](#) equation 4-11.

$$c = c_{\infty} e^{\left(\frac{-2MG}{c_s^2 r} \right)} \quad (\text{TMG Eq. 4-11})$$

Note: The author derived this equation independently as documented in TMG. Similar exponential dependencies in the context of light propagation in gravitational fields have appeared in earlier work (e.g., [Krogh 1999, 2006](#); [Robertson 2018](#)), where exponential forms were introduced or utilized in alternative refractive-index or variable-speed-of-light models.

Assuming TMG Eq. 4-11 is true in general, calculate the N-body solution as:

$$c = c_{\infty} e^{\left(\frac{-2G}{c_s^2} \sum_{k=1}^n \left(\frac{M_k}{r_k} \right) \right)} \quad (1)$$

The radial distances r_k are defined relative to the point at which the gradient ∇c is evaluated, such that $r_k = |x_k - x|$, where x is the evaluation point and x_k is the location of mass M_k .

$$\nabla c = \left(\frac{2G}{c_s^2} \sum_{k=1}^n \left(\frac{M_k}{r_k^2} \hat{r}_k \right) \right) c_{\infty} e^{\left(\frac{-2G}{c_s^2} \sum_{k=1}^n \left(\frac{M_k}{r_k} \right) \right)} \quad (2)$$

Substitute Eq. 1 into 2 to obtain the local gradient of c .

$$\nabla c = c \left(\frac{2G}{c_s^2} \sum_{k=1}^n \left(\frac{M_k}{r_k^2} \hat{r}_k \right) \right) \quad (3)$$

From TMG
$$r_c = \frac{2c}{\left(1 + \frac{1}{\beta^2} \right) \nabla c \cdot \hat{r}_c} \quad (\text{TMG Eq. 7-9})$$

Apply TMG Eq. 7-9 to rotate the position and motion from point 1 to 2 (see Figure 2 on page 13).

Calculate the change in β from point 1 to 2 along the path of motion with TMG Equation 7-6 as follows.

From TMG
$$\frac{d\beta}{\beta} = \frac{(\beta^2 - 1)dc}{2\beta^2 c} \quad \text{TMG Eq. 7-6}$$

$$dc = \nabla c \cdot d\vec{r} \quad (4)$$

Substitute Eq. 4 into (TMG Eq. 7-6)

$$d\beta = \left(\frac{\beta^2 - 1}{2\beta c} \right) \nabla c \cdot d\vec{r} \quad (5)$$

The examples presented in this paper integrate the path by rotating the position and direction of motion from point 1 to point 2 with TMG Equation 7-9 (as depicted in Figure 2 on page 13) and calculating the change in β with equation 5.

The traditional method of integrating the path of motion can also be achieved by calculating the acceleration vector from the given equations. Subsequently, the path of motion can be integrated using an ordinary differential equation solver, such as a 4th-order Runge-Kutta method. The acceleration vector is the vector sum of the tangential and radial accelerations resulting from the curvature of motion.

By definition:

$$\vec{v} = \frac{d\vec{r}}{dt} = \frac{\vec{v}}{c}c = \frac{v\hat{v}}{c}c = \beta\hat{\beta}c = \vec{\beta}c \quad (6)$$

Therefore:

$$\frac{d\vec{v}}{dt} = \frac{d\vec{\beta}}{dt}c + \frac{\vec{\beta}dc}{dt} \quad (7)$$

7.4 CWPM Radial Acceleration

The radial acceleration for circular motion is given by:

$$\left. \frac{d\vec{v}}{dt} \right|_{\text{radial}} = -\frac{v^2}{r_c}\hat{r}_c$$

Substitute in [TMG](#) Eq. 7-9

$$\begin{aligned} \left. \frac{d\vec{v}}{dt} \right|_{\text{radial}} &= -\frac{v^2 \left(1 + \frac{1}{\beta^2}\right) \nabla c \cdot \hat{r}_c}{2c} \hat{r}_c \\ \left. \frac{d\vec{v}}{dt} \right|_{\text{radial}} &= -\left(\frac{v^2 c}{2c^2} \left(1 + \frac{1}{\beta^2}\right) \nabla c \cdot \hat{r}_c \right) \hat{r}_c \end{aligned} \quad (8)$$

Substitute Eq. 6 into 8

$$\begin{aligned} \left. \frac{d\vec{v}}{dt} \right|_{\text{radial}} &= -\left(\frac{\beta^2}{2} \left(1 + \frac{1}{\beta^2}\right) c \nabla c \cdot \hat{r}_c \right) \hat{r}_c \\ \left. \frac{d\vec{v}}{dt} \right|_{\text{radial}} &= -\left(\frac{\beta^2}{2} \left(\frac{\beta^2 + 1}{\beta^2} \right) c \nabla c \cdot \hat{r}_c \right) \hat{r}_c \\ \left. \frac{d\vec{v}}{dt} \right|_{\text{radial}} &= -\left(\left(\frac{\beta^2 + 1}{2} \right) c \nabla c \cdot \hat{r}_c \right) \hat{r}_c \end{aligned} \quad (9)$$

7.5 CWPM Tangential Acceleration

Substitute Eq. 6 into Eq. 5 and multiply by $\hat{\beta}$ to give the direction.

$$\begin{aligned}\frac{d\vec{\beta}}{dt} &= \hat{\beta} \left(\frac{\beta^2 - 1}{2\beta c} \right) \nabla c \cdot \vec{\beta} c \\ \frac{d\vec{\beta}}{dt} &= \hat{\beta} \left(\frac{\beta^2 - 1}{2\beta} \right) \nabla c \cdot \vec{\beta}\end{aligned}\tag{10}$$

Substitute Eq. 10 into Eq. 7.

$$\left. \frac{d\vec{v}}{dt} \right|_{\text{tangential}} = \hat{\beta} \left(\frac{\beta^2 - 1}{2\beta} \right) \nabla c \cdot \vec{\beta} c + \frac{\vec{\beta} dc}{dt}\tag{11}$$

By definition

$$\frac{dc}{dt} = \nabla c \cdot \vec{\beta} c\tag{12}$$

Substitute Eq. 12 into Eq. 11.

$$\begin{aligned}\left. \frac{d\vec{v}}{dt} \right|_{\text{tangential}} &= \hat{\beta} \left(\frac{\beta^2 - 1}{2\beta} \right) \nabla c \cdot \vec{\beta} c + \vec{\beta} \nabla c \cdot \vec{\beta} c \\ \left. \frac{d\vec{v}}{dt} \right|_{\text{tangential}} &= \hat{\beta} \left(\beta + \frac{\beta^2 - 1}{2\beta} \right) \nabla c \cdot \vec{\beta} c \\ \left. \frac{d\vec{v}}{dt} \right|_{\text{tangential}} &= \left(\left(\frac{3\beta^2 - 1}{2} \right) c \nabla c \cdot \hat{\beta} \right) \hat{\beta}\end{aligned}\tag{13}$$

7.6 CWPM Acceleration Vector

The acceleration vector associated with the curvature of wave propagation is the sum of the tangential acceleration (as given in equation 13) and the radial acceleration (as given in equation 9).

$$\frac{d\vec{v}}{dt} = \left(\left(\frac{3\beta^2 - 1}{2} \right) c \nabla c \cdot \hat{\beta} \right) \hat{\beta} - \left(\left(\frac{\beta^2 + 1}{2} \right) c \nabla c \cdot \hat{r}_c \right) \hat{r}_c\tag{14}$$

$$\bullet \hat{r}_c = \hat{\beta} \times \frac{\nabla c}{\|\nabla c\|} \times \hat{\beta}$$

- Calculate c with equation 1
- Calculate ∇c with equation 3

Figure 2 presents a particle moving through space along a curved path. Starting with the direction and curvature of motion at point 1, calculate the direction of motion at point 2 by rotating the average curvature vector, r_{ca_n} , through the angle α . Estimate r_{ca_n} by calculating the local curvature with [TMG](#) Eq. 7-9 and applying results from the previous time or rotation step with equation 15.

$$r_{ca_n} = r_{c_{2n-1}} \frac{r_{c_{2n-1}} + r_{c_{1n-1}}}{2r_{c_{1n-1}}} \quad (15)$$

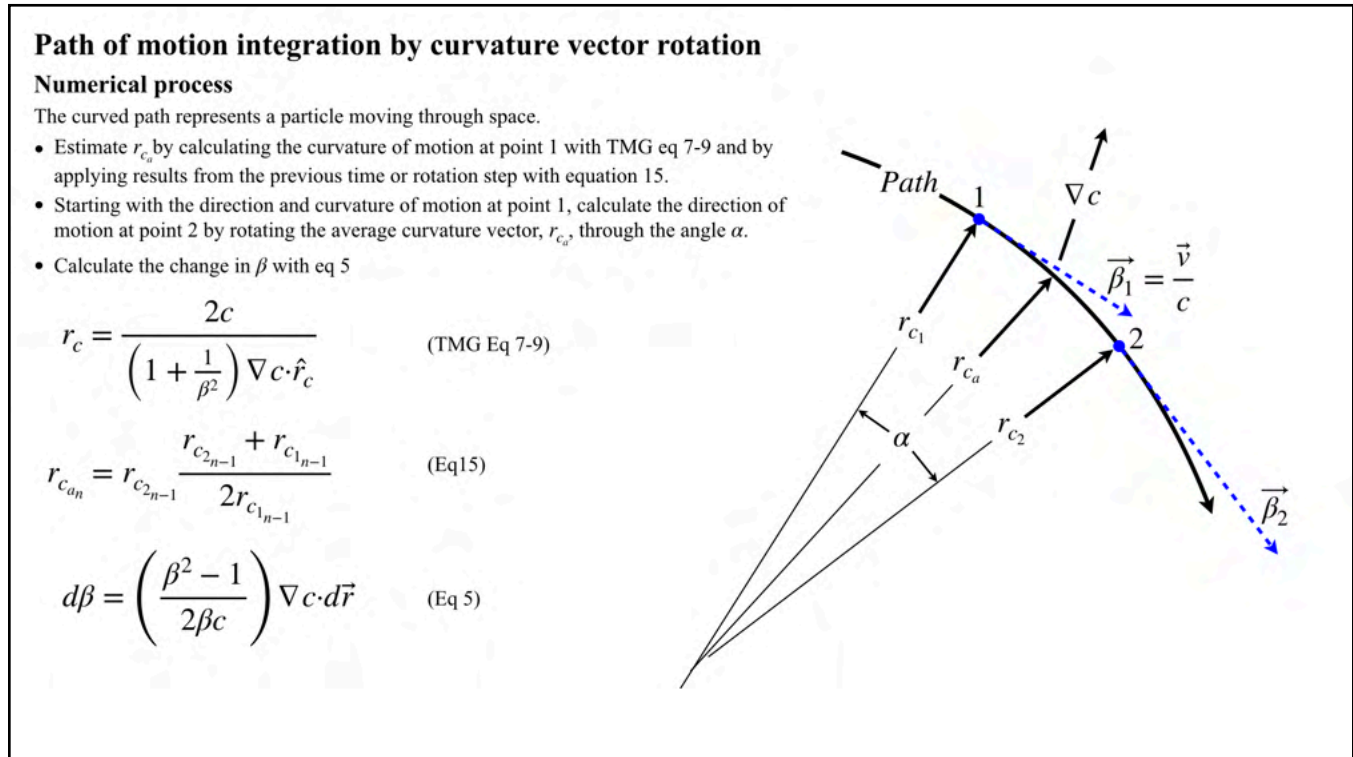


Figure 2 The average radius of curvature, r_{ca_n} , calculation.

After calculating β_2 with equation 5, proceed to calculate r_{c_2} and so forth. The TMG_nb_path_dt_integration.m script file, included in Appendix B on page 49, provides the code.

The rotation angle or corresponding time step selected for integrating the solution can have a significant impact on the results. Running multiple N-body solutions with different time steps enables identifying the maximum value for the desired accuracy.

A skilled numerical methods programmer should be able to significantly improve the solution speed and accuracy by incorporating advanced ordinary differential equation (ODE) solver methods.

One of the advantages of this numerical method is that it is simple enough for the N-body example files (provided with this paper) to run on a tablet with an app that does not have an advanced ODE solver, such as the [SIMO MATLAB programming app \(Lau, M.\)](#).

By applying this simple approach, we can teach science students (gifted high school and undergraduate college) to understand and solve these N-body gravity simulation examples.

Figure 3 illustrates the simplicity of this approach. Advantages associated with CWPM include:

- Validity for both rest mass and electromagnetic radiation.
- Eliminates the need to calculate point-to-point mass accelerations associated with the PPN method.
- Separates the curvature of motion calculation from the gradient of space computation, providing more flexibility in selecting the numerical approach for an N-body simulation.
- Enables integrating the trajectory with an angular rotation step, which equates to a variable time step for improved numerical efficiency.
- Accurately simulates the influence of gravity and motion on the natural frequency (clock rate) of matter, simplifying the task of calculating differences between Earth-based, rocket-based, and satellite-based clocks.
- Valid under extreme gravitational conditions including within the Schwarzschild radius.

Curvature of Wave Propagation Equations of Motion

Curvature of Wave Propagation

Curvature of Wave Propagation

$\hat{r}_c = \hat{\beta} \times \frac{\nabla c}{\|\nabla c\|} \times \hat{\beta}$

$\vec{r}_c$

$\vec{v} = \frac{\vec{v}}{c}$

∇c

θ

Method 1

$r_c = \frac{2c}{\left(1 + \frac{1}{\beta^2}\right) \nabla c \cdot \hat{r}_c}$

$d\beta = \left(\frac{\beta^2 - 1}{2\beta c}\right) \nabla c \cdot d\vec{r}$

$$\frac{d\vec{v}}{dt} = \left(\left(\frac{3\beta^2 - 1}{2} \right) c \nabla c \cdot \hat{\beta} \right) \hat{\beta} - \left(\left(\frac{\beta^2 + 1}{2} \right) c \nabla c \cdot \hat{r}_c \right) \hat{r}_c$$

For a simple n-body simulation

$$c = c_\infty e^{\left(\frac{-2G}{c_s^2} \sum_{k=1}^n \left(\frac{M_k}{r_k} \right) \right)}$$

$$\nabla c = c \left(\frac{2G}{c_s^2} \sum_{k=1}^n \left(\frac{M_k}{r_k^2} \hat{r}_k \right) \right)$$

Path is integrated by rotating the curvature vector $\vec{r}_c$
or by integrating $d\vec{v}/dt$ with an ODE solver.

for comparison

NASA's JPL Orbital Integration Approach

applies the parametrized post-Newtonian (PPN) method.

The equation, in the extract below, is copied from page 5 of:

The JPL Planetary and Lunar Ephemerides DE440 and DE441

Ryan S. Park , William M. Folkner , James G. Williams , and Dale H. Boggs

(Will & Nordtvedt 1972; Moyer 2003)

$$\begin{aligned} \mathbf{a}_{i,\text{pm-pm}} = & \sum_{j \neq i} \frac{GM_j(\mathbf{r}_j - \mathbf{r}_i)}{r_{ij}^3} \left\{ 1 - \frac{2(\beta + \gamma)}{c^2} \right. \\ & \times \sum_{k \neq i} \frac{GM_k}{r_{ik}} - \frac{2\beta - 1}{c^2} \sum_{k \neq j} \frac{GM_k}{r_{jk}} \\ & + \gamma \left(\frac{v_i}{c} \right)^2 + (1 + \gamma) \left(\frac{v_j}{c} \right)^2 - \frac{2(1 + \gamma)}{c^2} \mathbf{v}_i \cdot \mathbf{v}_j \\ & \left. - \frac{3}{2c^2} \left[\frac{(\mathbf{r}_i - \mathbf{r}_j) \cdot \mathbf{v}_j}{r_{ij}} \right]^2 + \frac{1}{2c^2} (\mathbf{r}_j - \mathbf{r}_i) \cdot \mathbf{a}_j \right\} \\ & + \frac{1}{c^2} \sum_{j \neq i} \frac{GM_j}{r_{ij}^3} \{ [\mathbf{r}_i - \mathbf{r}_j] \cdot [(2 + 2\gamma)\mathbf{v}_i \\ & - (1 + 2\gamma)\mathbf{v}_j] \} (\mathbf{v}_i - \mathbf{v}_j) \\ & + \frac{(3 + 4\gamma)}{2c^2} \sum_{j \neq i} \frac{GM_j \mathbf{a}_j}{r_{ij}}, \end{aligned} \quad (27)$$

where the summations are over all bodies, and β and γ are the Eddington–Robertson–Schiff parameters representing the measure of nonlinearity in the superposition law for gravity and the amount of space curvature produced by a unit rest mass, respectively, and are constrained to unity as predicted by the general theory of relativity (GTR).

Path is calculated by integrating a_i with an ODE solver

Figure 3 CWPM (above left) provides two simple ways to integrate an N-body simulation. The approach applied by NASA is also presented (above right) for comparison. See [DE430 \(Folkner, W. M. et al., 2014\)](#); and [DE440 \(Park, R. S. et al., 2021\)](#) for more information regarding the NASA JPL PPN numerical method.

7.7 N-Body Kinetic and Gravitational Potential Energy

Based on the derivation provided in [TMG](#), the rest mass of an object is a function of position in a gravitational field and is calculated as:

$$m_o = m_{o\infty} \sqrt{\frac{c}{c_\infty}} \quad \text{From TMG Eq. 5-10}$$

The relationship between mass and energy is also derived to be:

$$E = mc_s^2 \quad \text{TMG Eq. 2-5}$$

$$m = \frac{m_o}{\sqrt{1 - \beta^2}} \quad \text{TMG Eq. 2-12}$$

$$ke = c_s^2(m - m_o) \quad \text{TMG Eq. 2-14}$$

Hence, for an N-body system, the kinetic energy is calculated as:

$$ke = c_s^2 \sum_{k=1}^n \left(\frac{m_o}{\sqrt{1 - \beta^2}} - m_o \right) \quad (16)$$

We can also define the gravitational potential energy as:

$$pe = 0.5c_s^2 \left(\left. \sum_{k=1}^n m_o \right|_{t=t} - \left. \sum_{k=1}^n m_{o\infty} \right|_{t=0} \right) \quad (17)$$

If energy is conserved during the calculation, then, based on the above definitions:

$$(ke_t - ke_{t=0}) - (pe_t - pe_{t=0}) = 0 \quad (18)$$

as demonstrated by the star cluster (Example 5) script files.

8. Example MATLAB Script Files for Comparison With Data

Appendix B on page 49 provides MATLAB script files for the examples listed below. The script files are also available for download at: <https://www.mathworks.com/matlabcentral/fileexchange/178949-n-body-gravity-simulation-by-curvature-of-wave-propagation>. These examples illustrate that the method presented in this paper closely simulates the influence of gravity on the path of motion.

Example 1 The Solar System and the precession of Mercury's orbital perihelion

Example 2 Shapiro time delay and the bending of light

Example 3 Apparent black hole shadow diameter

Example 4 Simple trajectory motion

Example 5 Star cluster, symmetric and random

Example 6 Pioneer 10, 11, Voyager 1, and 2 trajectories

In the preprint of this work, published on February 21, 2025, the results for Examples 1, 2, and 6 were calculated under the erroneous assumption that the NASA JPL Horizons position and velocity vector output is defined at the standard location, conventionally realized on Earth for SI units. However, the Horizons system is actually based on **Barycentric Dynamical Time (TDB)**. This crucial information is clearly explained in the following comment that is included with vectors downloaded from the Horizons system.

```
Barycentric Dynamical Time ("TDB" or T_eph) output was requested. This
continuous coordinate time is equivalent to the relativistic proper time of a
clock at rest in a reference frame co-moving with the solar system barycenter
but outside the system's gravity well. It is the independent variable in the
solar system relativistic equations of motion.
```

```
TDB runs at a uniform rate of one SI second per second and is independent of
irregularities in Earth's rotation.
```

According to this information, the NASA Solar System ephemerides and the Horizons system output for time, length, and the gravitational parameter (GM) are based on a convention where the standard location for time is set at infinity outside the Solar System's gravitational well. Therefore, to ensure consistency, when initializing and modeling a CWPM simulation with vectors from the Horizons system, both the standard speed of light (c_s) and the speed of light away from the Solar System's influence (c_∞) should be set equal to 299 792 458 m/s.

The example script files have been updated to align with the NASA TDB convention.

This correction doesn't significantly impact the results or conclusions of Examples 1, 2, and 6. However, it becomes crucial when comparing predicted positions with ephemerides for calculations that extend several years into the past or future.

The initial conditions for the examples presented in this paper are sourced from the JPL Horizons website, relative to the Solar System barycenter. However, it's important to note that the simulations presented here do not include all of the Solar System bodies. Consequently, the simulations begin with a non-zero group velocity that causes the barycenter to drift. While this does not significantly affect the orbital period or precession calculations, it introduces a discrepancy when comparing the results with the ephemeris tables. Therefore, it's crucial to consider this factor when making such comparisons.

Example 1: Solar System N-body Simulation and the Precession of Mercury's Orbital Perihelion

This example demonstrates CWPM provides accurate orbital precession results for calculations conducted in the International Celestial Reference Frame (ICRF) with initial conditions from the JPL Horizons website. The N-body mass values for this simulation are from [DE440 Table 2 \(Park, R. S. et al., 2021\)](#) and [DE430 Table 12 \(Folkner, W. M. et al., 2014\)](#).

Example 1 is a 15-body simulation of the Solar System starting January 1, 2000. Included bodies are the Sun, Mercury, Venus, the Earth and Moon, Mars system, Vesta, Ceres, Pallas, Juno, Jupiter system, Saturn system, Uranus system, Neptune system, and the Pluto system. However, it may be run with fewer bodies by changing the number of included bodies. It can also be used with a custom initial conditions file to evaluate an N-body simulation of choice. With this example, we can investigate the individual motion, orbital period, and precession of bodies in the Solar System.

Figure 4 illustrates two solutions obtained with Example 1. The first is with initial conditions as obtained from JPL. The second is with a group velocity of 230 km/s added to the JPL initial conditions to simulate Solar System motion relative to the galactic center. Both solutions are nearly identical when evaluated relative to the position of the Earth. However, upon careful examination of the MATLAB solution results, slight discrepancies are evident, which result in inaccurate orbital precession results for the simulation with a group velocity of 230 km/s. These differences may emanate from the simplifying assumptions associated with the gradient of space calculation.

Though not proven by derivation yet, it appears equation [TMG Eq. 7-9](#) is valid for both fixed and moving reference frames. As stated in Section 7.2, the examples presented in this paper do not account for the time delay in changes to the properties of space emanating from the motion of a governing body. Therefore, these calculation results are most accurate for slow changes in the gravitational gradient, such as caused by a passing body moving with a velocity much less than the speed of light. We should be able to achieve greater accuracy in the future by taking these influences into account when calculating the gradient of the local properties of space. Hopefully, this will also enable accurately calculating orbital precession in a moving reference frame.

Example 1 identifies the orbital perihelion as the point of the minimum gravitational potential and calculates the precession by the dot product of the initial and final perihelion velocity vectors. This approach is consistent with the traditional definition of a closed elliptical orbit around a fixed body. Implementation is simple and works well for N-body systems, which usually do not have closed elliptical orbits. However, it does not always identify the orbital perihelion. When applied to the Moon, it captures the period of the lunar phase cycle instead of the perihelion relative to Earth, as illustrated in Figure 4.

Note: The paper *Comparative Study of Mercury's Perihelion Advance* by [Pogossian S. P. \(2022\)](#) provides a thorough and in-depth analysis of various alternative methods for determining the orbital perihelion advance.

The left side of Figure 4 illustrates results calculated in the ICRF. The solution on the right includes a group velocity of 230 km/s added to the ICRF initial conditions to simulate motion relative to the center of the Milky Way. The resulting motion relative to the Earth is nearly identical, with both solutions providing the same period for the phases of the Moon.

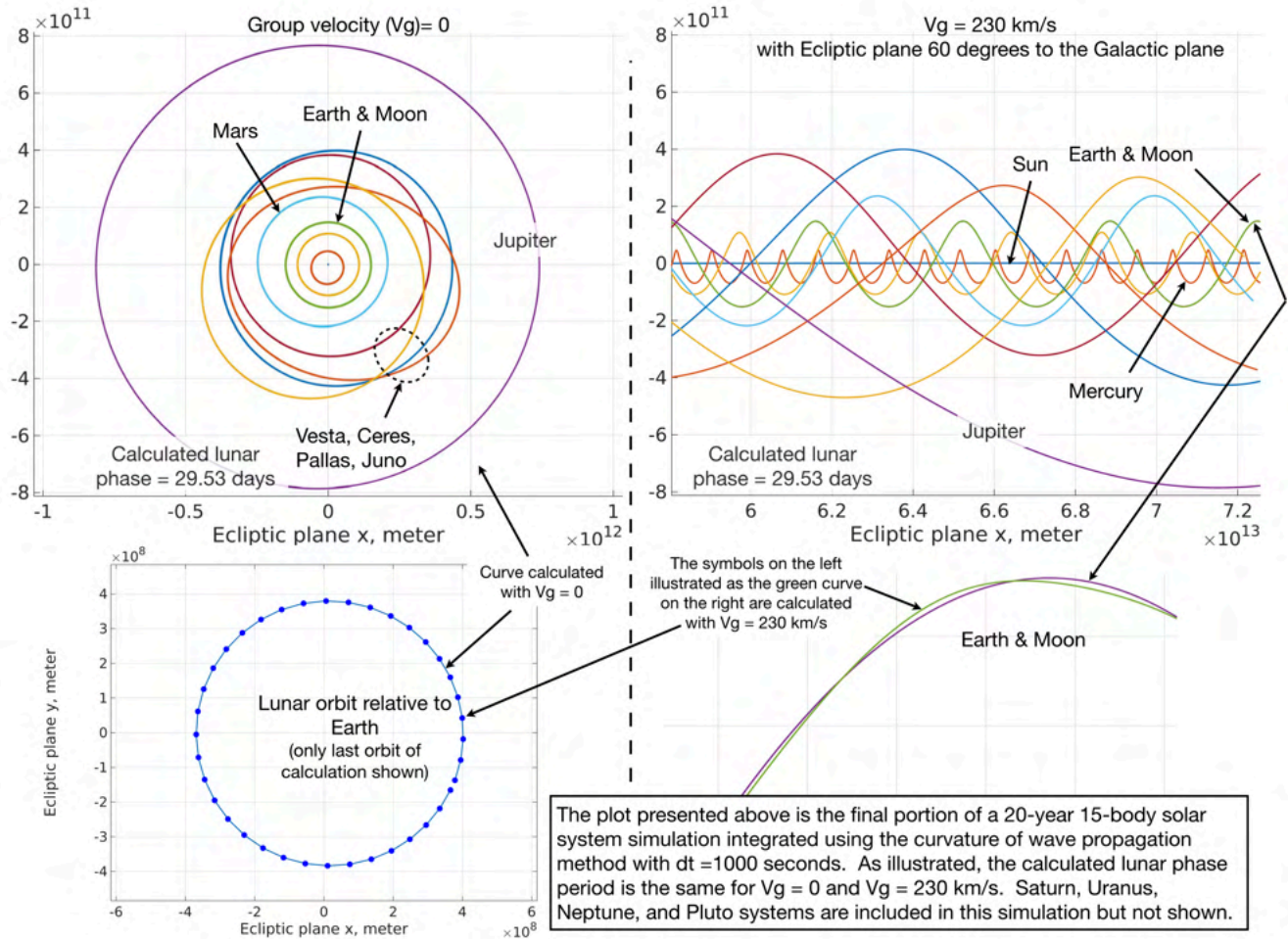


Figure 4 15-body Solar System simulation results, calculated with group velocities of 0 and 230 km/s, respectively, indicate CWPM is valid for both fixed and moving reference frames. Initial conditions (for group velocity = 0) are from the JPL Horizons website.

Hypothetical 2-Body Simulation of Mercury's Orbital Precession

Figures 5 and 6 illustrate results for a hypothetical 2-body simulation of Mercury and the Sun obtained with the Example 1 script file by setting the number of the included bodies equal to 2. Each result in Figure 5 represents a 20-year average, starting at A.D. 2000-Jan-01, illustrating that the calculated precession changes insignificantly for time steps less than 20 seconds. Figure 6 presents results integrated forward and backward in time by 2000 years. The initial conditions and gravitational parameters are from the JPL Horizons website for A.D. 2000-Jan-01, for which the velocity of the Sun is about 16 m/s. Therefore, the group velocity of these 2-body simulations is also approximately 16 m/s.

The calculated precession rate of Mercury converges to about 42.94"/century (see Figure 5) with an orbital period of approximately 7600531.113 seconds (87.96911 days). For comparison, the widely published value associated with general relativity is 42.98"/century.

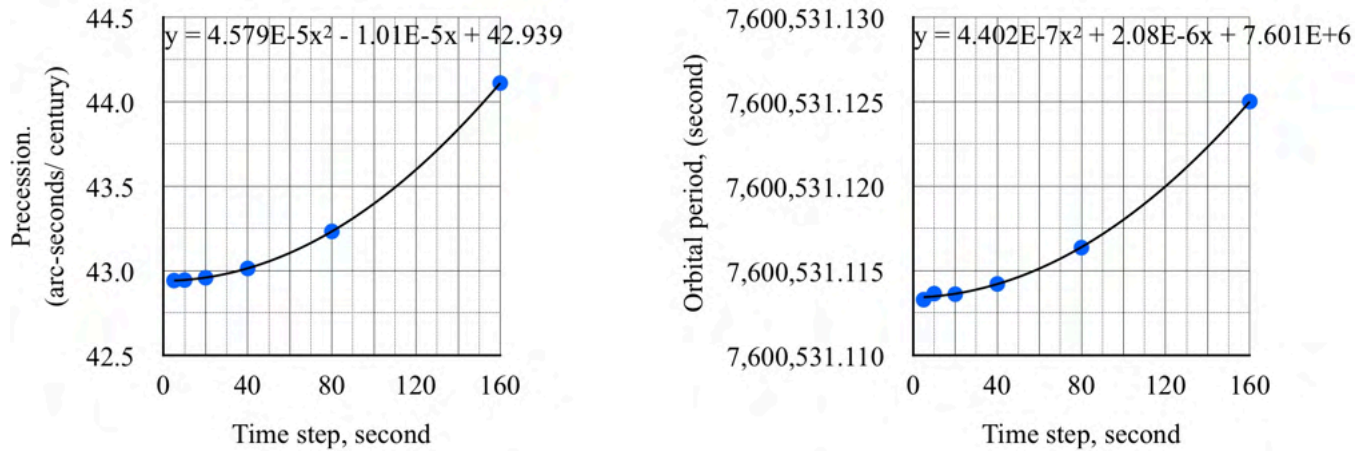


Figure 5 Sample 2-body Sun-Mercury simulation results - averaged over 20 years.

As illustrated in Figure 6 (right side), the calculated orbital period (perihelion to perihelion) decreases over time. A likely cause is numerical truncation errors associated with the increasing distance from the reference frame origin related to the simulation group velocity. However, the average precession value (Figure 6 left side) remains unchanged. The drift in the orbital period may also be associated with the group velocity and the simplification of not accounting for the time delay in changes to the properties of space emanating from the motion of the Sun.

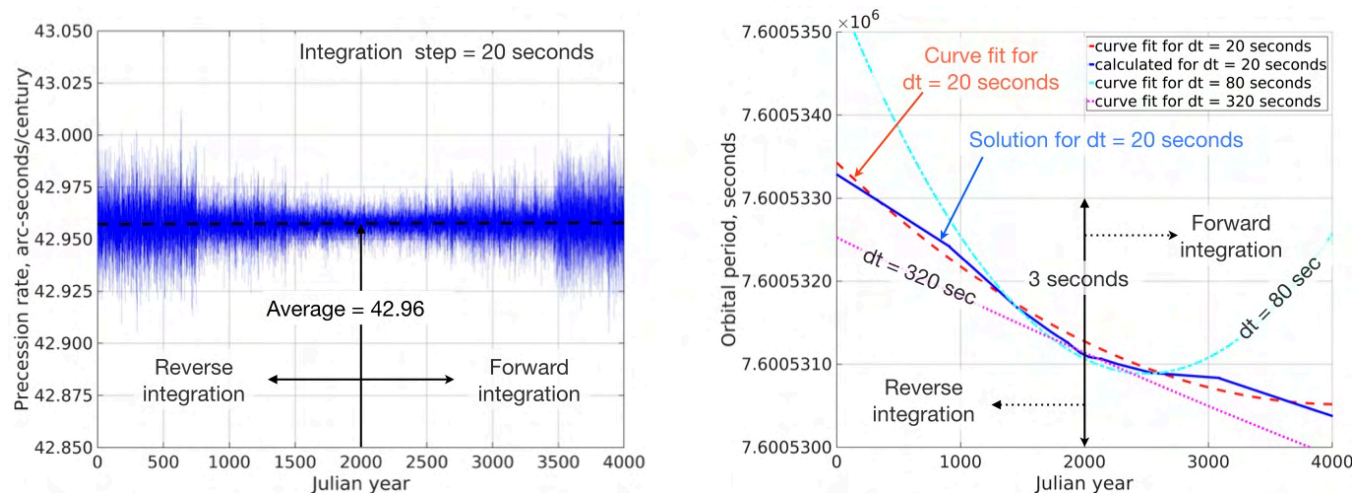


Figure 6 Sample 2-body Sun-Mercury simulation results integrated forward and backward in time.

The results presented in Figures 5 and 6 confirm the following:

- The numerical approach converges asymptotically as the time step approaches zero.
- The calculated precession agrees closely with the generally accepted value.
- Time steps on the order of 20 seconds provide good results for this simulation.
- The numerical integration is stable over the 2000-year simulation time.
- Numerical truncation errors, evident by the increased scatter in the calculated orbital precession with time, don't significantly alter the solution.

The results presented in Figures 5 and 6 are calculated for an earth based standard location. However, the NASA data match is based on the TDB convention (explained on page 16), which it located outside the Solar System's influence. While placing the standard location to align with NASA TDB doesn't significantly affect the calculated precession, it does result in a slight increase in the calculated orbital period of Mercury by approximately 0.5 seconds. The attached script files have been updated to reflect this change. This adjustment also applies to the following 15-Body simulation.

15-Body Simulation for Calculating Mercury's Orbital Precession

The initial conditions for this simulation are the same as in the previous 2-body example. They include the barycenter position and velocity values from the JPL Horizons website (planetary system mass not available) combined with the planetary system mass documented in [DE440 and DE441 \(Park, R. S. et al., 2021\)](#).

Figure 7 illustrates the precession of Mercury's perihelion integrated forward and backward 2000 years by CWPM. Venus has the most influence, followed by Jupiter and Earth. The numerical values provided in Table 1, for a time-step of 20 seconds, agree closely with the [Park, R. S. et al. \(2017\)](#) PPN estimate matched to radiometric range measurements of the MESSENGER spacecraft.

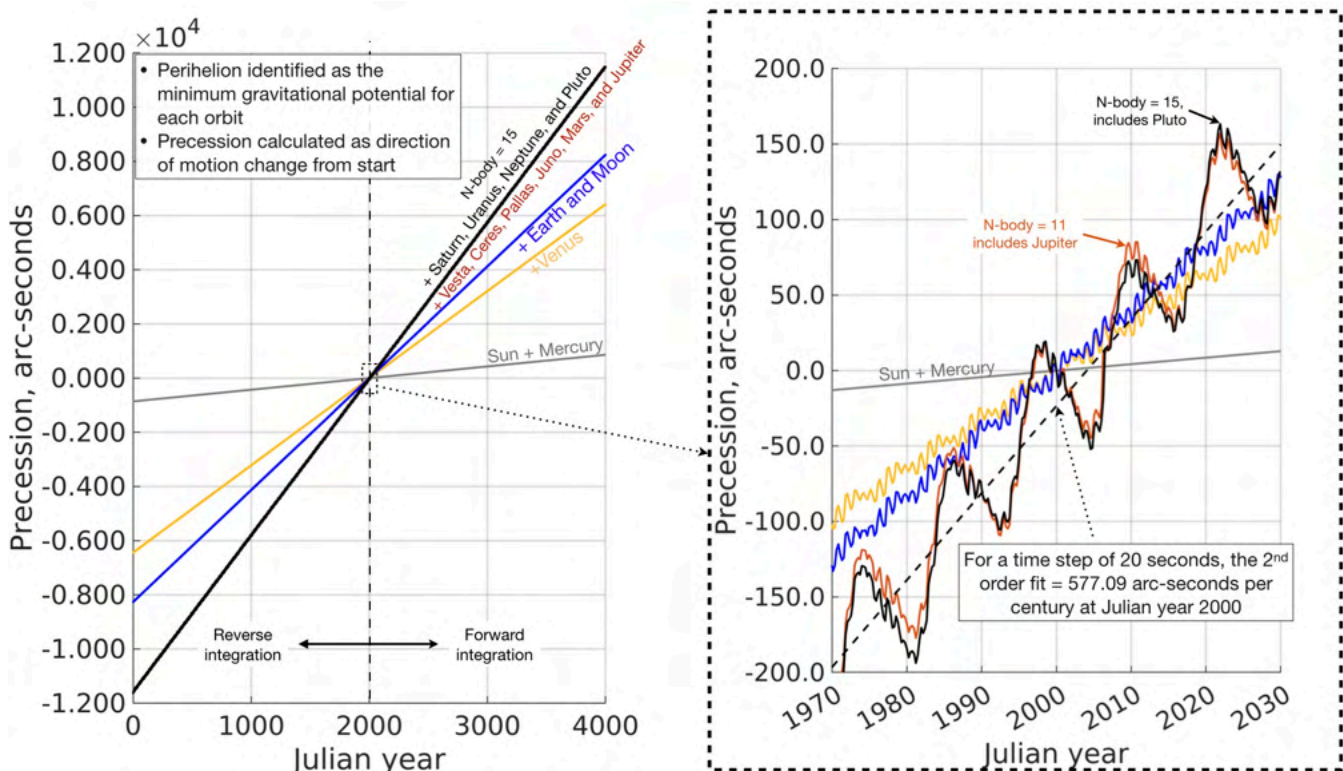


Figure 7 The additive influence of Solar System bodies on the precession of Mercury's orbit calculated with a time step of 20 seconds.

Mercury orbital precession for Julian Year 2000 arc seconds / Julian century Calculated with a 2 nd order polynomial fit of N-body simulation results, forward and aft, 2000 years with a time step of 20 seconds		Precession Rate	Delta Precession Rate	Perihelion to Perihelion
Gravitational influence	Number of simulated bodies	"/century	"/century	Average Period Days
Sun + Mercury	2	42.9575	42.9575	87.9691120
+ Venus	3	321.4255	278.4681	87.9693306
+ Earth & Moon	5	412.5603	91.1348	87.9693317
+ Mars System	6	415.0424	2.4821	87.9693340
+ Vesta, Ceres, Pallas, and Juno	10	415.0435	0.0011	87.9693339
+ Jupiter System	11	569.3688	154.3253	87.9693446
+ Saturn System	12	576.9065	7.5377	87.9693444
+ Uranus System	13	577.0483	0.1418	87.9693445
+ Neptune System	14	577.0930	0.0447	87.9693445
+ Pluto System	15	577.0930	0.0000	87.9693445

Table 1 Individual contributions to Mercury's in-plane orbital perihelion shift for $dt = 20$ seconds.

The right side of Figure 8 confirms that a 20-second time step is sufficient to integrate the results presented in Table 1. The calculated precession of Mercury, within its orbital plane, is $577.08''/\text{century}$, which is equivalent to $\dot{\omega} = (\dot{\omega} + \dot{\Omega} \cos i) = 575.93''/\text{century}$ for Julian year 2000 (see Appendix A on page 42).

Figure 8 illustrates these results are in close agreement with values obtained by conducting a best fit of Keplerian elements to observational data ([Standish and Williams, 2006](#)) and the PPN estimates calculated by [Park, R. S. et al. \(2017\)](#) and [Kuijpers, N., \(2022\)](#).

As reported by Park, R. S. et al., (2017), the orbital precession rate of Mercury decreases with time. Figure 8 illustrates that the curvature of wave propagation approach predicts a similar rate.

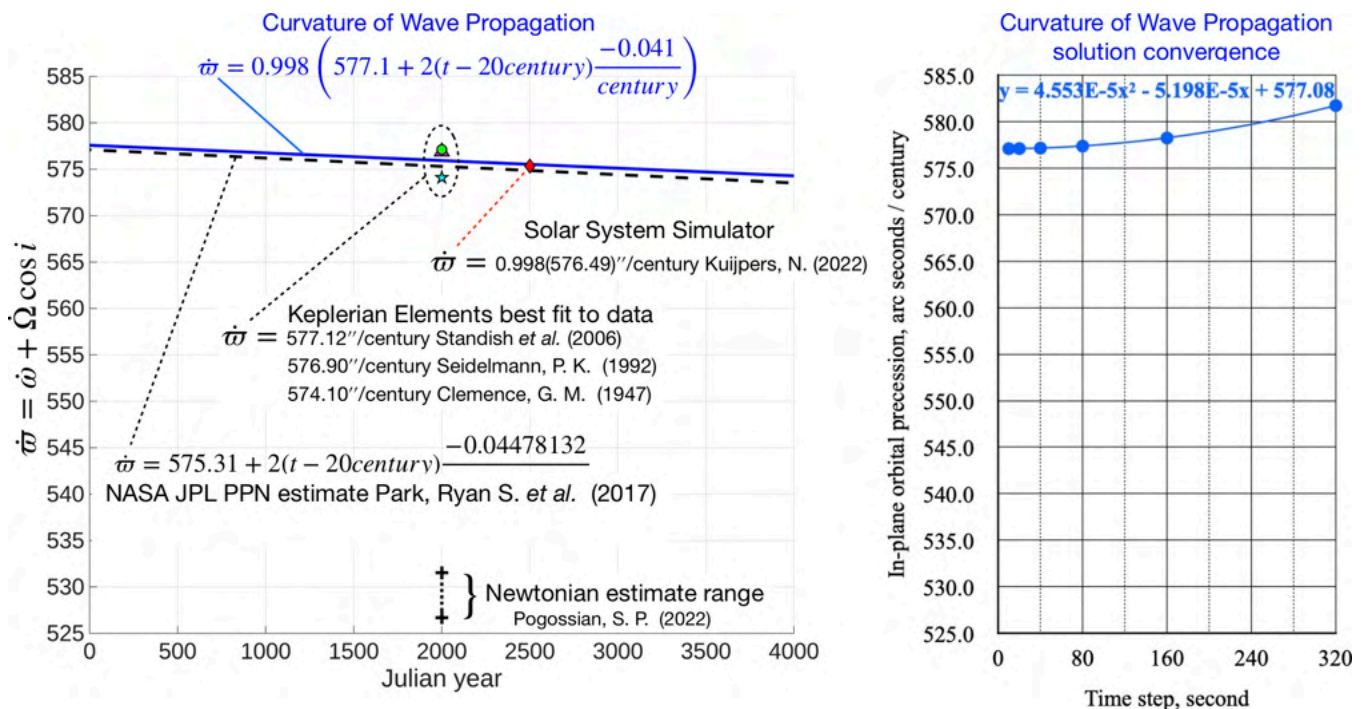


Figure 8 N-Body simulation of Mercury's orbital precession and solution convergence

Observations

- The forward and reversed integrated orbital period drift is similar indicating it is caused by numerical errors.
- The slight difference between forward and reverse integration indicates Mercury's actual orbital period shift (for the simulated conditions) is less than 1 second in 10,000 years

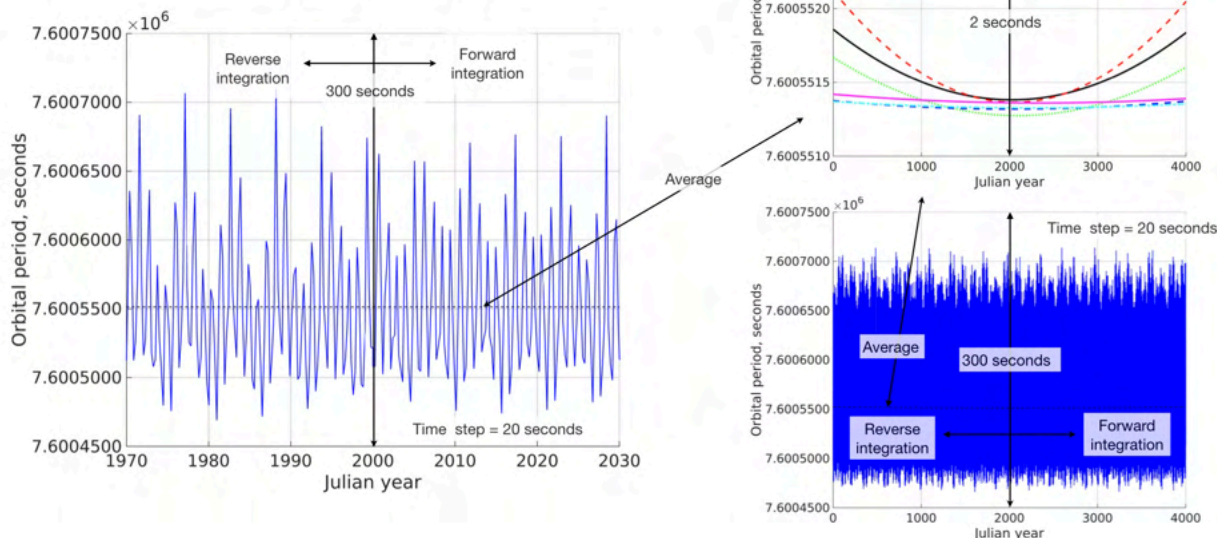


Figure 9 Mercury orbital period oscillations over time, calculated by CWPM, confirm changes in the average orbital period are likely less than 1 second / 10000 years.

The results presented in Figure 9 illustrate that changes in the orbital period are similar when integrating forward or backward in time. The influence of time step on numerical accuracy is also apparent. Judging from these results, the orbital period of Mercury changes by less than one second in ten thousand years due to gravitational influences. This study does not consider other influences, such as the drag from the solar wind, the accretion of cosmic dust, magnetic fields, etc.

The results presented in Figures 7, 8, 9, and Table 1 confirm the following:

- The numerical approach converges asymptotically as the time step approaches zero for the orbital period and precession (see Figure 8 on the right side).
- The calculated precession results are in close agreement with observational data.
- Time steps on the order of 20 seconds provide good results for this simulation.
- The numerical integration is stable over the 2000-year forward and backward simulation time.
- Numerical truncation errors (evident in the orbital period calculation - see upper right of Figure 9) don't significantly alter the solution convergence (see Figure 8).
- Identifying the orbital perihelion as the point of the minimum gravitational potential and calculating the precession by the dot product of the initial and final perihelion velocity vectors works well.

Remarks

The results (presented in this example) are calculated using the standard gravitational constant, mass properties, and initial conditions from the [JPL Solar System Dynamics](#) website (without modification). Fitting calculated results by adjusting the mass and initial conditions of the simulated bodies to observational data (similar to what is done for [DE430 and DE431](#) and modifying the gradient of space calculation to simulate the solar oblateness, etc.) should improve accuracy.

The orbital perihelion determination (minimum gravitational potential) and precession calculation (dot product between initial and final velocity vectors) methods presented here are much different than those used by [Park, R. S. et al. \(2017\)](#), which may cause some of the differences illustrated in Figure 8.

Example 2: Shapiro Time Delay and the Bending of Light

This example (see Figure 10) calculates the Shapiro time delay and deflection of light passing between multiple bodies in our Solar System. It begins with initial conditions from the JPL Horizons website for January 25, 1970, and rotates the planets into position for the calculation. A photon is then added to the simulation, starting from the surface of one body and directed to arrive at another as it passes near the Sun. The numerically integrated results provide the Shapiro time delay and light deflection.

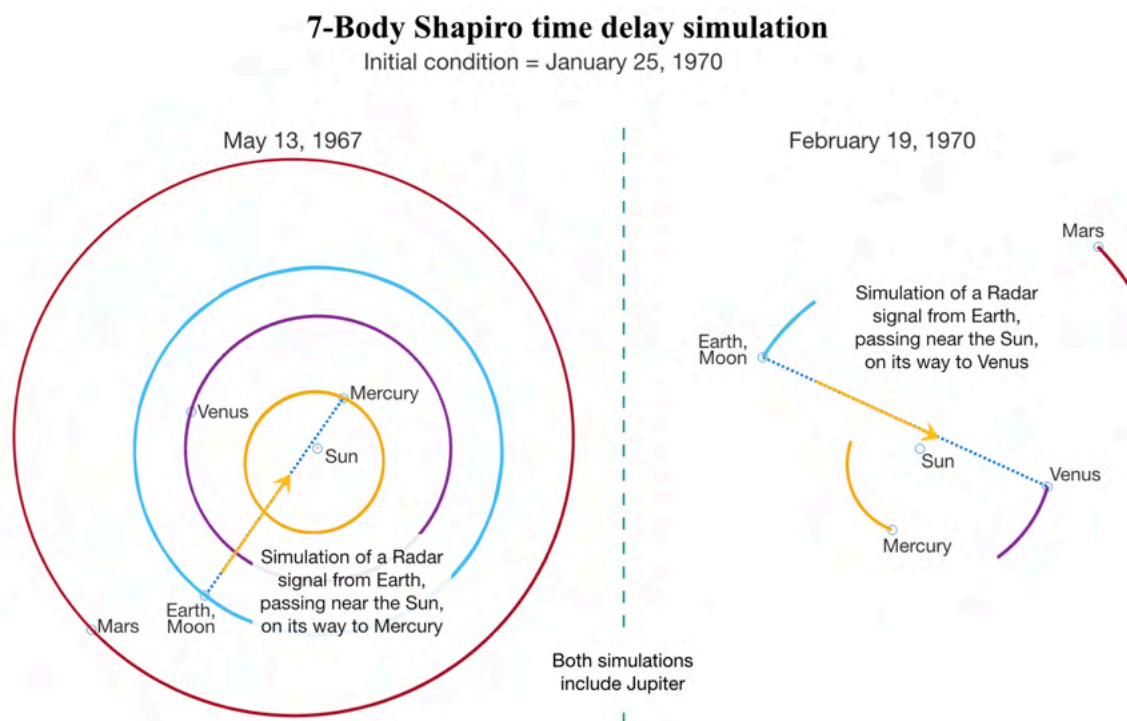


Figure 10 N-body simulation for comparison with the [1967](#) and [1970](#) Shapiro time delay data.

The Curvature of Motion in a Gravitational Field ([Anderson, M. G., 2021](#)) explains the physics and calculation. It also illustrates the influence of propagation speed and curvature on the signal delay. Some of the following information is adapted from that document.

Light follows a curved path near a gravitational body, which causes a direction change of about 1.75 arcseconds for light that grazes the solar limb on its way to Earth ([Blanco, V. M. and McCuskey, S. W., 1961](#)). Additionally, light takes longer to pass through space near a governing body (Shapiro, I. I., [1964](#), [1968](#), [1971](#)), as confirmed by test results documented in the PHYSICAL REVIEW LETTERS ([Shapiro, I. I., 1968](#)).

This example enables a direct comparison of CWPM with the Shapiro time delay data (see Figures 11 and 12) and the light deflection measurements recorded by [Bruns D. G. \(2018\)](#) (see Figure 11).

The time delay is calculated as:

$$\text{Time delay} = \text{path time} - \frac{\text{Euclidean distance}}{c_{\text{reference}}}$$

$$c_{\text{reference}} = \text{Light propagation speed away from the Solar System's influence}$$

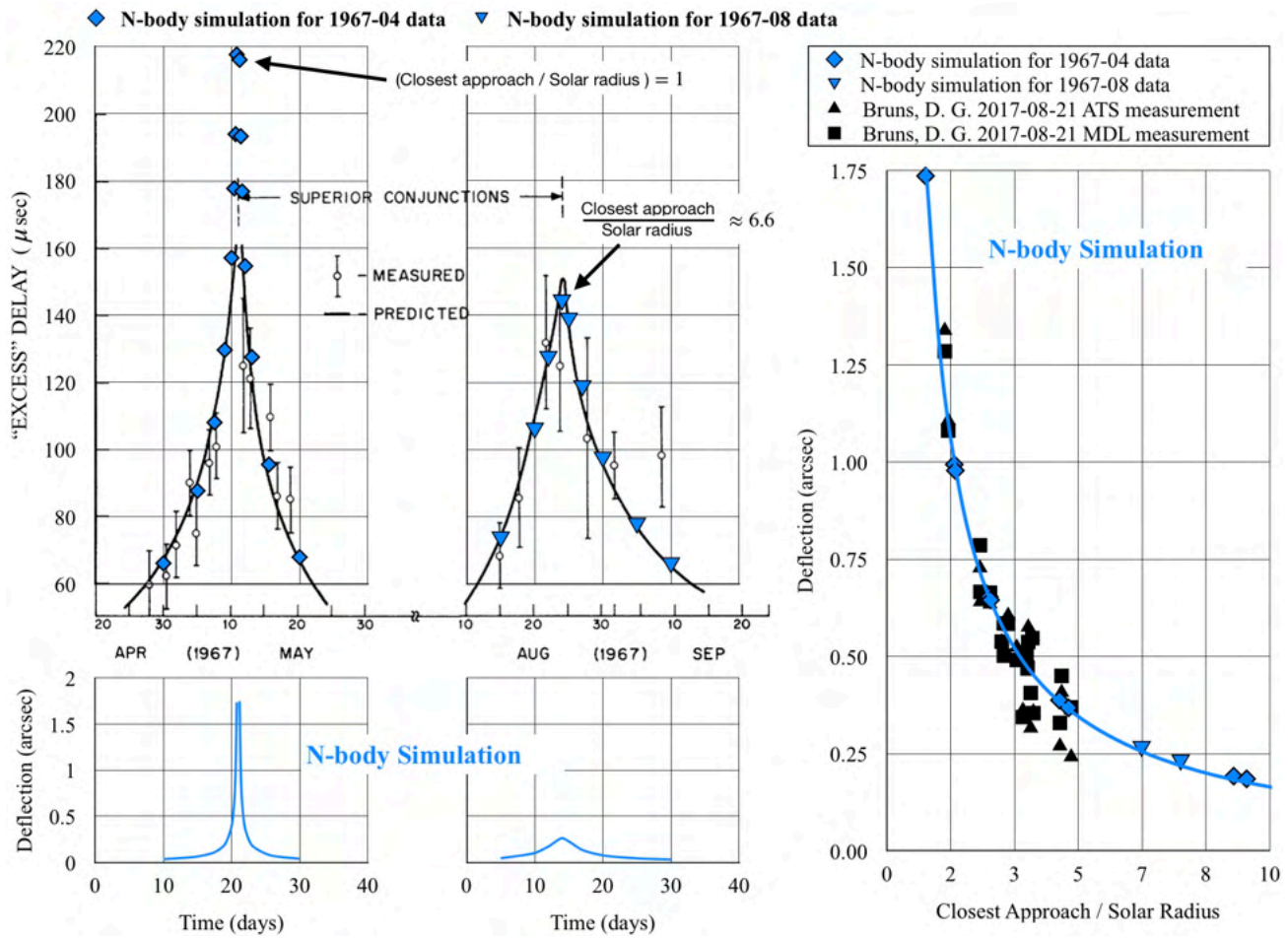


Figure 11 8-body (Sun, Mercury, Venus, Earth, Moon, Mars, Jupiter, and one photon) calculation of the Shapiro time delay between Earth and Mercury. The upper left corner of this Figure illustrates the curvature of wave propagation results (blue diamonds and triangles) plotted on top of Figure 3 from [Shapiro, I. I. et al. \(1968\)](#). The right side illustrates that the predicted light deflection agrees with the 2017 light deflection data obtained by [Bruns, D. G. \(2018\)](#).

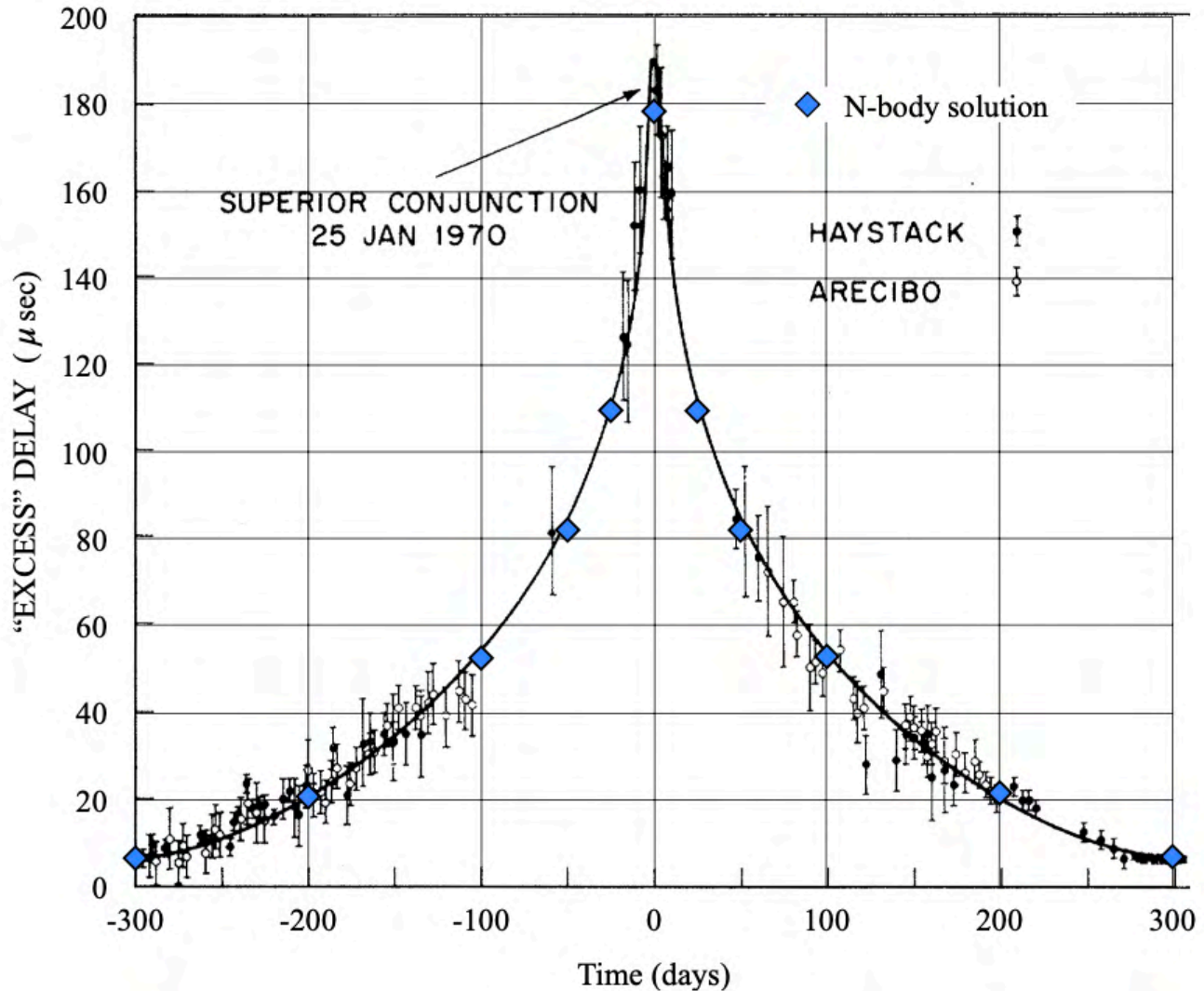


Figure 12 N-body calculation of the Shapiro time delay between Earth and Venus (blue diamond symbols). These results are plotted on top of Figure 1 from [Shapiro, I. I. et al. \(1971\)](#).

The results presented in Figures 11 and 12 confirm that CWPM closely simulates the measured Shapiro time delay of radar signals and deflection of light passing near the Sun. Therefore, we can conclude that CWPM also accurately simulates the motion of photons passing through a gravitational field.

Example 3: Apparent Black Hole Shadow Diameter

This example demonstrates the ability to simulate motion in a strong gravitational field. The term Apparent Black Hole refers to a body that is classified as a black hole under general relativity. Within the [TMG](#) framework, however, it only appears as such due to its very low inherent emissivity, which causes the central body to be outshone by surrounding material (gas, dust, stars, etc.). This concept is explained in detail in *The Curvature of Motion in a Gravitational Field* ([Anderson, M. G., 2021](#)).

Figure 13 illustrates a wavefront of photons (modeled as particles with zero rest mass) approaching an apparent black hole. A portion of the wavefront is deflected toward the central mass, with the photons inside the dashed lines contributing to the shadow region. The shadow diameter is defined as the span of the captured portion of the wavefront.

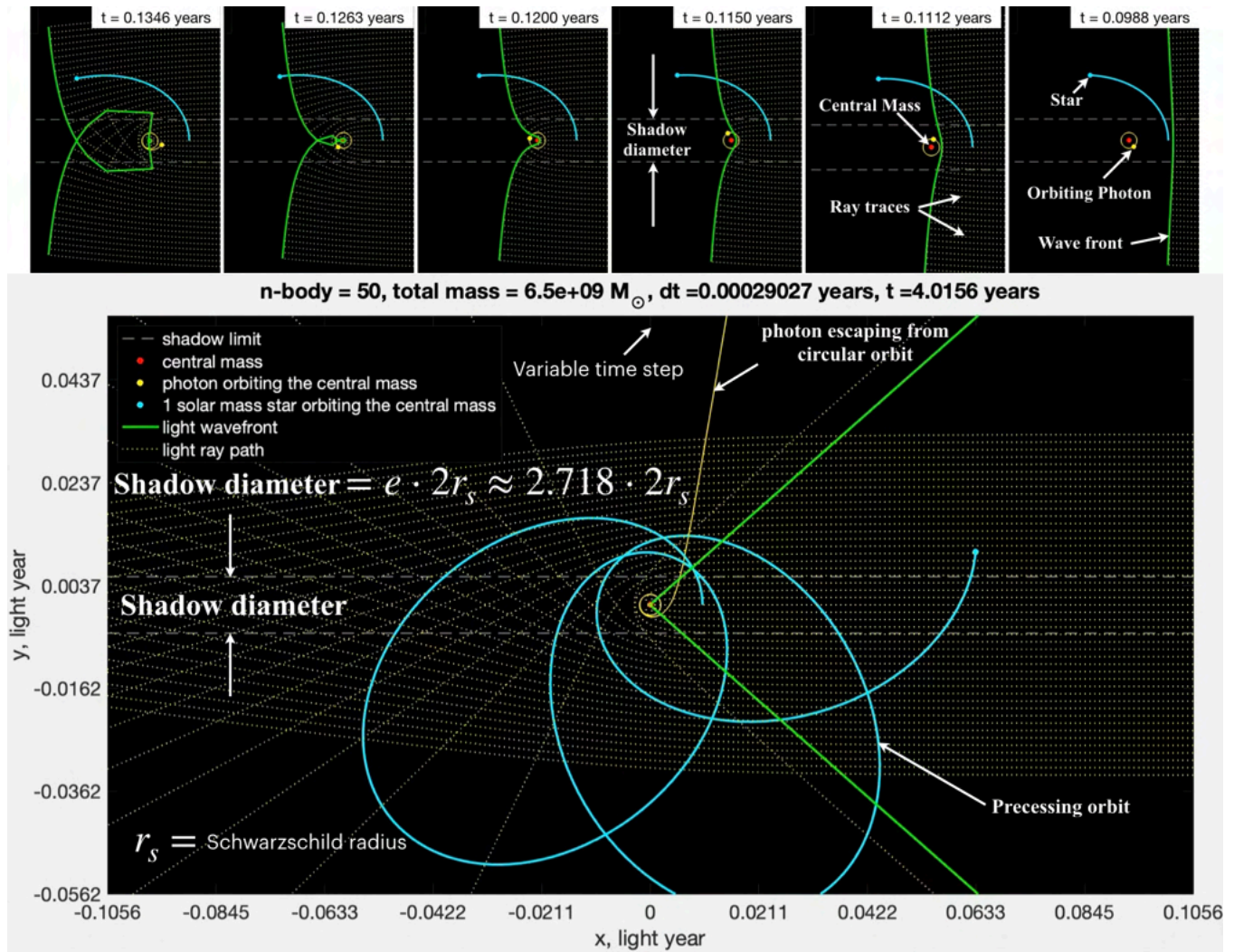


Figure 13 Apparent black hole shadow and orbital precession n-body simulation.

This simulation also includes a photon initially placed in a circular orbit around the central mass, together with a one-solar-mass star undergoing significant orbital precession. After several revolutions, the photon's trajectory becomes unstable, allowing it to escape the gravitational influence of the central mass. Numerical truncation errors, combined with perturbations from the orbiting star, may have contributed to this outcome.

The time delay associated with changes in the gravitational field does not significantly affect the solution, since the initial velocity of the central mass—dominating the spatial gradient of the field—is only about 0.3 m/s. The motion of the photons does not alter this gradient, and the star's mass is negligible compared to the central body. Consequently, the paths of the orbiting star and photon, together with the photon wavefront, are consistent with the physical conditions represented in the simulation.

The Curvature of Motion in a Gravitational Field ([Anderson, M. G., 2021](#)) provides additional background information and integration results, demonstrating close agreement between numerical simulations and observational data.

Strong gravitational conditions thus serve as a key testing ground for evaluating differences between CWPM and general relativity. A summary of these differences is presented in Table 2.

	^{1, 2} CWPM	^{3, 4} General Relativity	Significance
Gravitational Redshift (z)	$z = \frac{1}{\sqrt{e^\phi}} - 1$ $\phi = \frac{-2GM}{c_s^2 r}$	$z = \frac{1}{\sqrt{1 + \phi}} - 1$ $\phi = \frac{-2GM}{c_s^2 r}$	CWPM provides a continuous natural logarithmic solution. The general relativity redshift is singular if $r = r_s \text{ where } r_s = \frac{2GM}{c_s^2}$ r_s = Schwarzschild radius
Photon Sphere	$Radius = r_s$	$Radius = 1.5 \cdot r_s$	CWPM < GR
Black hole shadow diameter (D_{sh})	$D_{sh} = 2er_s$ $D_{sh} \approx 2.718 \cdot 2r_s$	$D_{sh} = \frac{3\sqrt{3}}{2} \cdot 2r_s$ $D_{sh} \approx 2.598 \cdot 2r_s$	CWPM $\approx$ 4.6 % greater than GR (likely within measurement uncertainty)
Event horizon diameter	None	$Diameter = 2r_s$	CWPM predicts no event horizon, but the central mass's very low emissivity can make it appear black-hole-like.
Frame dragging	<i>Under theoretical development</i>	Yes	<i>Frame dragging in the CWPM method is still in its theoretical development phase.</i>
Polar jets	From central body and accretion disk	From accretion disk	CWPM predicts that apparent black holes without an accretion disk can still emit particles and high-redshifted radiation.
Reference	1. <i>A Method for Predicting Quasar Luminosity Consistent With the NASA/IPAC Extragalactic Database</i> (Anderson, M.G., 2007) 2. <i>The Curvature of Motion in a Gravitational Field</i> (Anderson, M. G., 2021) 3. <i>Seeing Black Holes: from the Computer to the Telescope</i> (Luminet, Jean-Pierre, 2018) 4. <i>LECTURE NOTES ON GENERAL RELATIVITY</i> (Carroll, S. M., 1997)		

Table 2 Strong field differences between CWPM and general relativity.

The photon orbital radius depicted in Figure 13 is equal to the Schwarzschild radius (derived in [TMG](#) as Eq. 7-12). Moreover, Figure 13 illustrates that the wave-propagation method yields a continuous natural-logarithmic solution without an event horizon, extending smoothly inside the Schwarzschild radius to the central body. This allows the simulation of both particles and light up to and within r_s .

In contrast, according to general relativity, the photon sphere is located at $1.5 r_s$, with the event horizon positioned at r_s itself ([Carroll, S. M., 1997](#)). The static-observer gravitational redshift factor diverges at the event horizon, so that particular expression cannot be extended inside the horizon.

Example 4: Simple Trajectory Motion

This example demonstrates the ease of modeling simple trajectory motion with CWPM. The symbols presented in Figure 14 represent an N-body solution for several 0.25 kg objects launched with a velocity of 40 m/s at different angles relative to horizontal on the Earth's surface. For comparison, the line curves provide Newtonian results calculated for a constant acceleration of gravity. Students can try different time steps and initial conditions to see how variations impact the results.

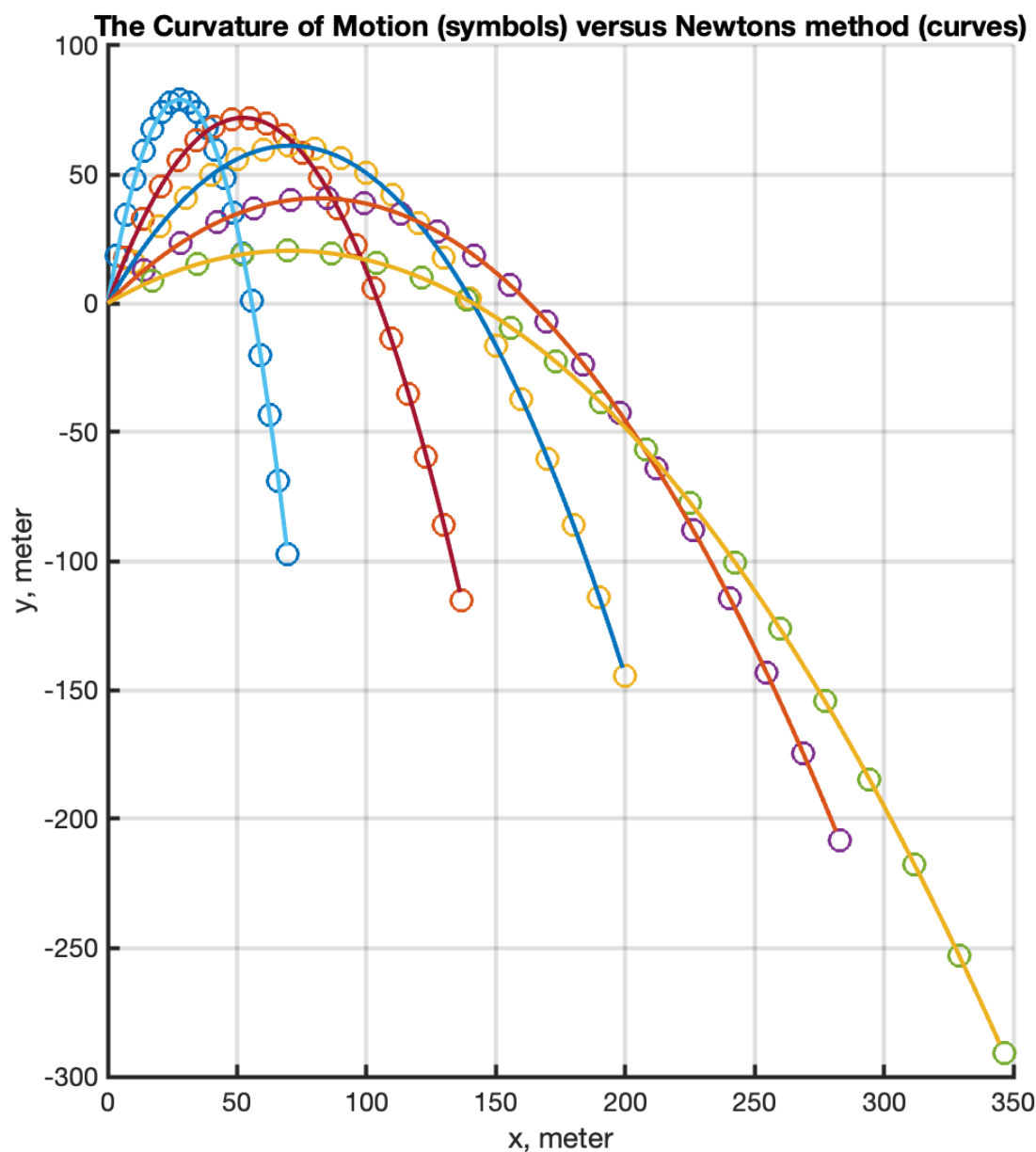


Figure 14 CWPM compared with Newtonian gravity for simple trajectory motion.

Initial conditions = 40 m/s at inclined angles of 30, 45, 60, 70, and 80 degrees relative to horizontal. Total simulated motion = 10 seconds. The time between consecutive symbols is 0.5 seconds. Time step = 0.05 seconds.

Example 5: Star Cluster, Symmetric and Random

Figures 15 and 16 illustrate symmetric simulations for 5 and 8 body systems. The maximum rotation per iteration is limited to 0.5 degrees, resulting in a variable time step based on the minimum curvature of motion within the simulation. The lower part of each plot illustrates the exchange between potential and kinetic energy and the conservation of total energy.

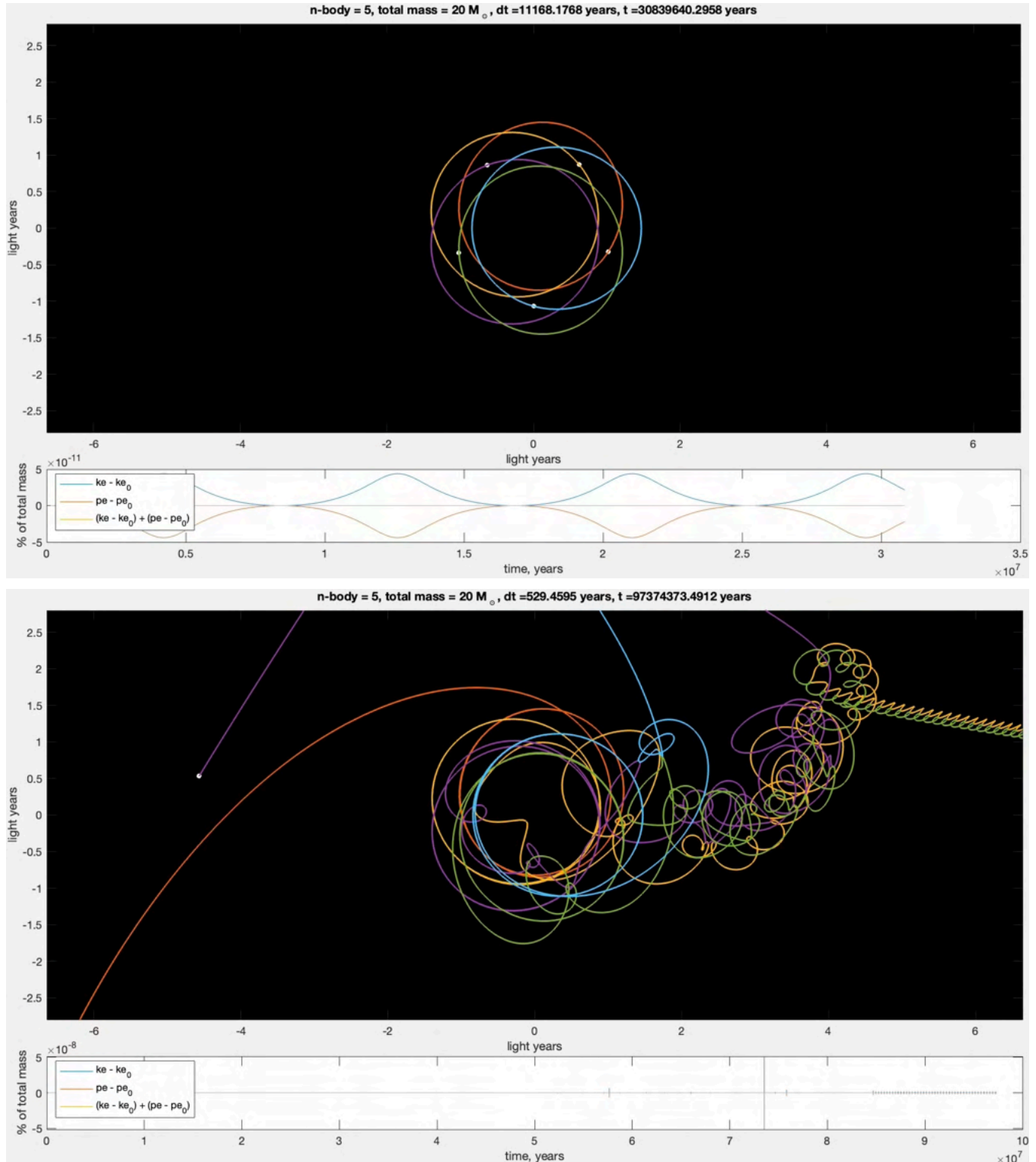


Figure 15 5-body simulation symmetry holds for about 31 million years (top view), it then becomes random because of numerical truncation and integration errors (bottom view).

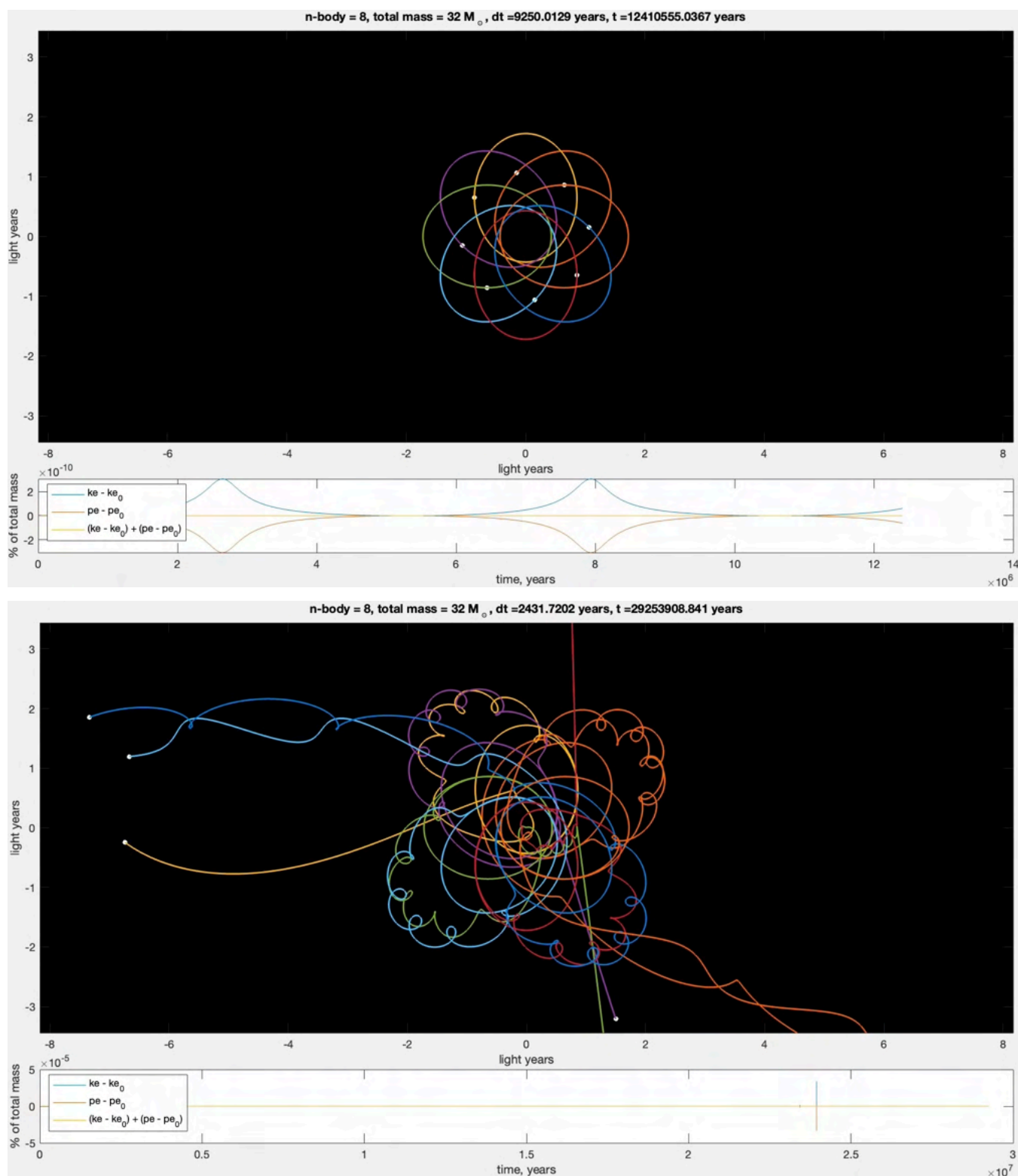


Figure 16 8-body simulation symmetry holds for about 12 million years (top view), it then becomes random because of numerical truncation and integration errors (bottom view).

Calculating a random (mass, velocity, and position) star cluster simulation is similar, as illustrated in Figure 17. Students can experiment with the attached script files to see how changing the number of bodies and the maximum rotation per time step influences the results. The bottom part of these plots also illustrates a conservation of energy.

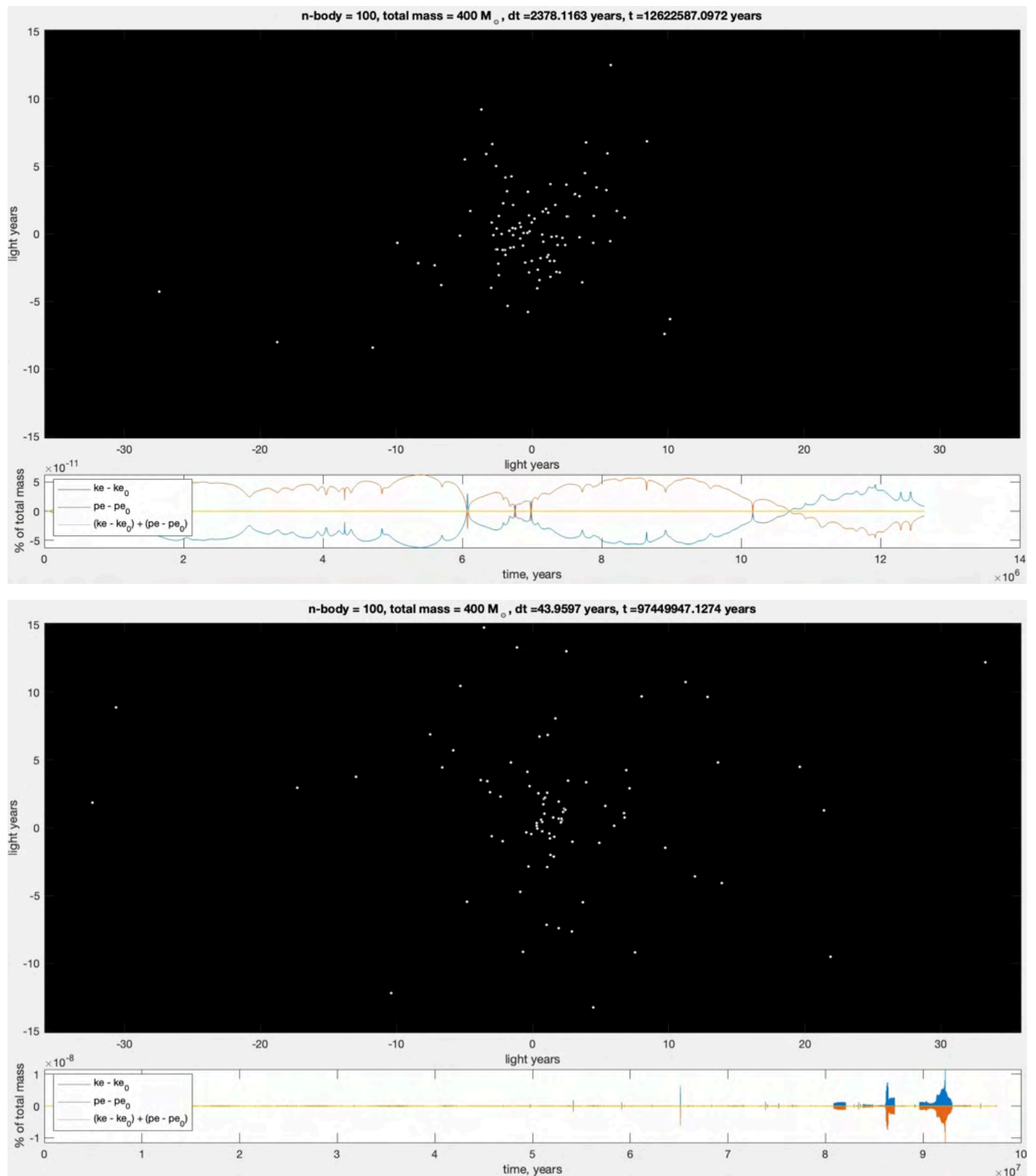


Figure 17 100-body simulation after about 13 million years (top view) and 100 million years (bottom view). Average star = 4 solar mass'. Initial density = 1.5 stars per cubic light year.

Example 6: Pioneer 10, 11, Voyager 1, and 2 Trajectories

This example illustrates gravity assist interplanetary spaceflight for the Pioneer and Voyager spacecraft, and how small course corrections are leveraged to complete these types of missions. The N-body simulation runs in about 15 minutes on a MacBook Pro M1 making it a practical learning exercise for students. The run time may be reduced to less than 6 minutes by saving and plotting fewer points during the simulation (set variable `dgps = 120` instead of `dgps = 5` on line 75 of the example MATLAB script file `Example_6_Pioneer_10_11_Voyager_1_2.m`).

After making a sincere effort (searching the internet over several months), sufficient data to understand and simulate all of the mission course corrections for the Pioneer and Voyager spacecraft could not be found. Therefore, this simulation is not a highly accurate model of each probe's orbital path. Instead, it is a rough N-body simulation of the mission trajectories as illustrated in the bottom half of Figure 18.

Note: The influence of drag, radiation, and magnetic forces, etc. on the spacecraft is not included.

Students can work on improving this model to enhance their understanding of how to plan and conduct interplanetary space missions.

This example starts by initializing the Solar System with Pioneer 11 as it departs near Earth orbit which begins the simulation of Pioneer 10 close to its flyby of Jupiter. Voyager 1 and 2 spacecraft are added just before their flyby of Jupiter to illustrate partial studies of their trajectories. The simulation is integrated until 1995 for comparison with NASA JPL horizons trajectory data for each spacecraft as illustrated in Figure 18.

This simulation requires course corrections of about 96 m/s for Pioneer 11 and 93 m/s for Voyager 2. These corrections are within the design specifications of 200 m/s for Pioneer 11 ([Hall, C. F., and Smith, M. A. \(1971\) see Table 3.4-1](#)) and 190 m/s for Voyager 2 ([NASA \(1980\) Voyager Backgrounder - see page 19](#)). It is challenging to define course corrections necessary to match the trajectories. For comparison, [Santos, I. \(2019\)](#), illustrates this type of analysis with NASA's GMAT code.

The simulation steps are:

1973-04-12 Start + 0 days

Initialize the solution with data from the JPL Horizons website.

Integrate simulation to 1974-04-19

1974-04-19 Start + 372.0 days

Add in a simulated Pioneer 11 course correction - change velocity by 63.7 m/s to sharpen its trajectory for the Jupiter Flyby ([Fimmel, R. O. et al. \(1980\) NASA SP-446 page 77](#)).

Integrate simulation to 1978-07-13

1978-07-13 Start + 1918.0 days

Add in a simulated Pioneer 11 course correction - change velocity by 32.0 m/s to sharpen its trajectory for the Saturn Flyby. Documentation for this correction is not found, instead, it is estimated by adjusting the value for a reasonable trajectory.

Add in Voyager 1 and 2 spacecraft with conditions from the JPL Horizons website.

Integrate simulation to 1979-12-01

1979-12-01 Start + 2424.0 days

Add in simulated course corrections by changing velocities by 41.5 m/s for Voyager 1 and 92.0 m/s for Voyager 2 after their flyby of Jupiter to sharpen their trajectories for the Saturn Flyby. Documentation for this correction is not found, instead, it is estimated by adjusting the value for a reasonable trajectory.

Integrate simulation to 1982-01-01

1982-01-01 Start + 3186.0 days

Add in a simulated course correction by changing the velocity of Voyager 2 by 0.96 m/s after passing Saturn to sharpen its trajectory for the Uranus Flyby. Documentation for this correction is not found, instead, it is estimated by adjusting the value for a reasonable trajectory.

Integrate simulation to 1995-01-01

1995-01-01 Start + 7934.0 days - Stop

The top half of Figure 18 illustrates the path each spacecraft would take without course corrections. The bottom half of Figure 18 demonstrates that only minor course corrections (within the spacecraft capabilities - see simulation steps above) enable completing the missions successfully.

Simulate this example with a variable time step by limiting the maximum allowed rotation per integration step to 0.0025 degrees. Select this value based on the results presented in Figure 8 of Example 1, for which an angular rotation step of 0.0025 degrees equates to a time step of about 53 seconds for Mercury. Figures 18 and 19 indicate this is a reasonable choice.

The solution presented in Figure 18 closely simulates the trajectories extracted from the JPL Horizons website for the Pioneer and Voyager spacecraft (see Figure 18, light white lines), indicating that the rotation step and simulated course corrections are realistic for these missions.

Figure 18 presents results with and without velocity course corrections, demonstrating that small changes can significantly alter the solution.

The time step, or, in this case, the maximum allowed rotation angle per step, can also significantly impact the solution, as illustrated in Figure 19. In this example, a total velocity correction difference of only 1 m/s (after the flyby of Jupiter and Saturn) can cause the spacecraft to pass by Neptune at a distance of more than the diameter of Earth's solar orbit.

These results are calculated for an Earth-based standard location. Placing it outside the Solar System to properly match the NASA TDB convention (see page 16) slightly alters the course corrections needed for the probe trajectories. The attached script files have been updated to reflect this change.

These simulations show that small velocity changes upstream of a planetary flyby can result in significant trajectory differences afterward. Uncertainties associated with knowing the position and velocity, along with simulation errors of the trajectory, throughout the mission are inevitable. Therefore, course corrections during an interplanetary mission are required to reach the desired goal.

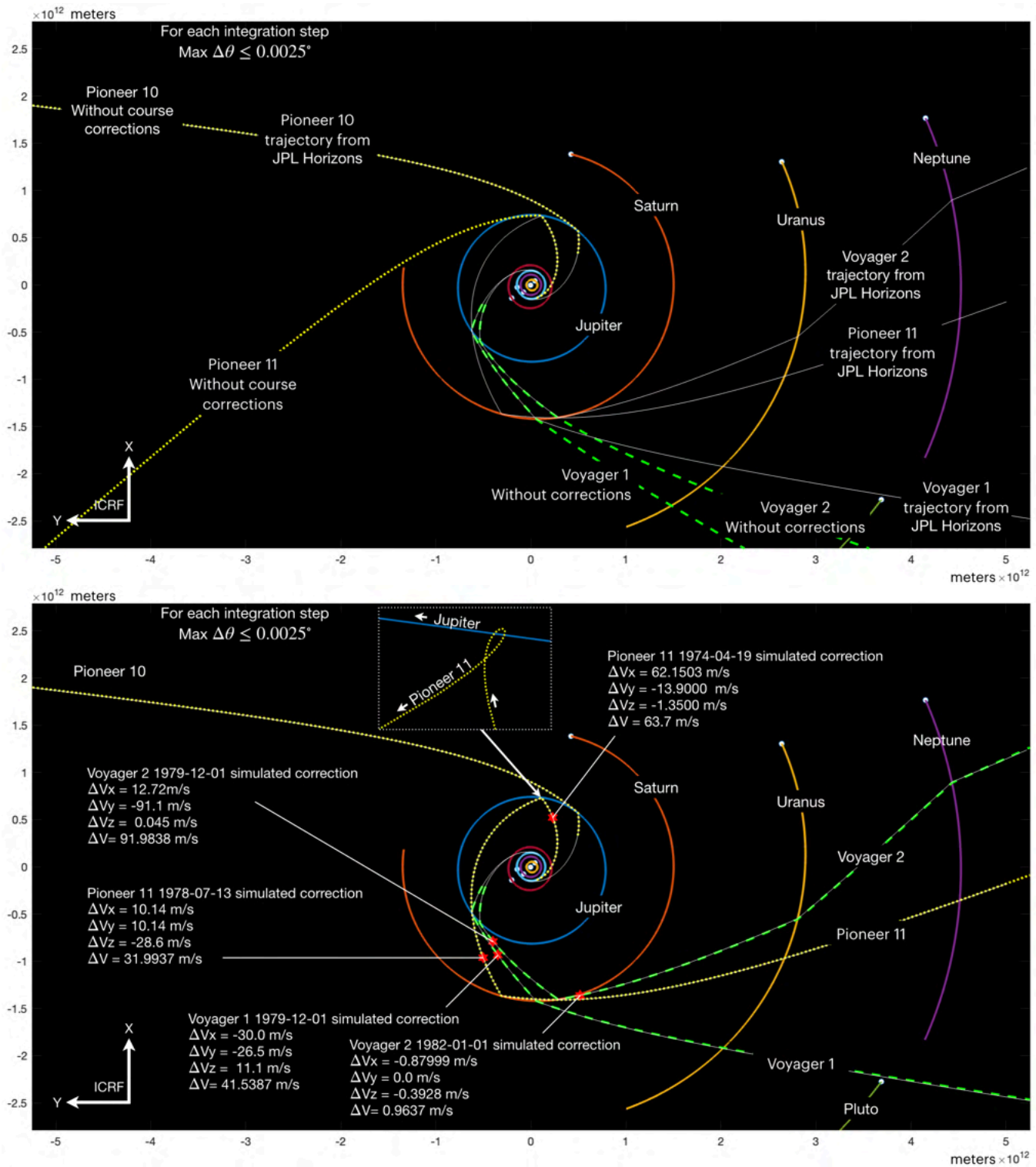


Figure 18 N-body simulation results calculated by CWPM.

Top view: Calculated without course corrections

Bottom view: Calculated with sample course corrections (obtained by trial and error) to approximately simulate the spacecraft trajectories extracted from the JPL Horizons website.

The magnified insert illustrates the complexity of Pioneer 11's flyby of Jupiter.

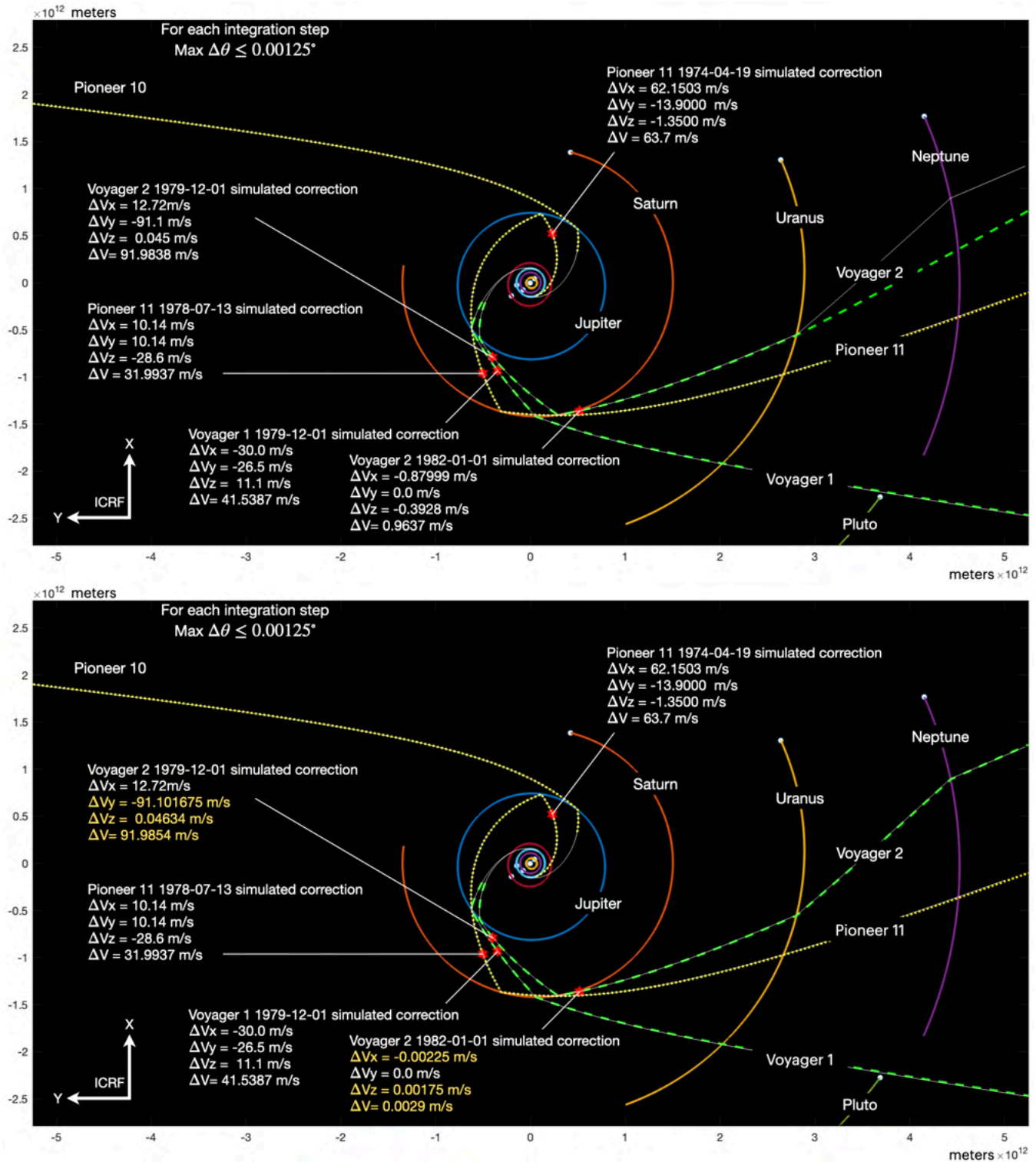


Figure 19 The top half presents a solution with the maximum integration step reduced to 0.00125 degrees using the same course corrections as the bottom of Figure 18. Notice how the path of Voyager 2 deviates from the JPL Horizons data after its flyby of Uranus (see top half of Figure 19). The bottom half illustrates how minor course correction changes (yellow text) bring it back to a more accurate solution.

9. Comparisons With General Relativity for the Examples Presented

In weak gravitational fields, such as those in the Solar System:

- Differences are indistinguishable for light deflection and time delay (see Example 2).
- Differences are less than $[((42.9799 - 0.0020) / 42.9575) - 1 = 5e-4]$ for the hypothetical 2-body Sun-Mercury system's precession (compare Table 1 with [Park, R. S. et al., 2017 Table 3](#)).
- Differences fall within observational error for Mercury's orbital precession (see Figure 8). Some discrepancies may stem from simplifications in the integration process.

In strong gravitational fields:

- The apparent black hole shadow diameter differs by about 4.6%, which remains within experimental error (see Example 3 and [Anderson, M. G., 2021 Example 8](#)).
- Modeling frame dragging within CWPM simulations remains under theoretical development.

10. Independent Verification and Performance Benchmark

Nico Kuijpers independently implemented the exact CWPM acceleration vector (Equation 14) in his pre-existing open-source [SolarSystemSimulator](#) ([Kuijpers, N., 2025](#)). Using identical JPL-derived initial conditions, no adjustable parameters, and constant rest mass for the ∇c calculation, he carried out 600-year, 10-body integrations with three dynamical models: classical Newtonian gravity, the post-Newtonian (PPN) approximation of general relativity, and CWPM. The simulations employed either a 4th-order Runge–Kutta scheme (for PPN and CWPM, 1-hour time step) or an Adams–Bashforth–Moulton scheme (for Newtonian mechanics, 0.5-hour time step). The resulting trajectories were then compared against [DE405](#) ephemeris data. (See Appendix C, p. 102, for Kuijpers' full report.)

Table 4 summarizes the results. Newtonian orbits diverge by several hundred to one hundred thousand kilometers after 600 years. Both PPN and CWPM remain stable and accurate to within several hundred km after 600 years. These differences are small and likely within the observational and modeling uncertainty over that time scale.

Method	Runtime (600 yr)	Speed-up vs PPN	Mercury deviation at 600 yr	Earth deviation at 600 yr	Jupiter deviation at 600 yr	Worst-case body at 600 yr
Newton	36 s	—	96 487 km	35 650 km	2 034 km	Mercury 96 487 km
GR / PPN	461 s	1.00×	173 km	273 km	636 km	Moon 2 159 km
CWPM	94 s	4.9×	144 km	300 km	630 km	Moon 2 114 km
Deviation calculated relative to NASA JPL Ephemeris DE405						

Table 4 [SolarSystemSimulator](#) results - [Accuracy experiment 600 years for NM, GR, CWPM](#)

Across all major bodies, the DE405-referenced deviations produced by CWPM and GR-based PPN differ by <30 km RMS over 600 years. Such close correspondence indicates that the analytically derived CWPM acceleration vector reproduces the leading-order relativistic effects encoded in the PPN expansion within a full N-body Solar System simulation.

In the independent benchmarks of [Kuijpers, N. \(2025\)](#), CWPM completes the 600-year Solar System integration in 94 seconds, compared to 461 seconds for the PPN implementation — a 4.9× speed-up. These computations were performed on a MacBook Pro with a 2.6 GHz 6-Core Intel Core i7 processor.

This improvement reflects the compact analytic form of the CWPM acceleration vector, which avoids the numerous velocity-dependent terms required by the PPN method.

Kuijpers also verified that including the influence of rest mass variation on ∇c increases CWPM deviations by only about 4km/600 year for Mercury while increasing solution time by ~6%. Therefore, it is not included in the [SolarSystemSimulator](#).

The CWPM implementation is now publicly selectable in the SolarSystemSimulator. Users can select between Newtonian gravity, general relativity PPN, and CWPM to independently verify these results.

11. Conclusions

The N-body gravity simulation by CWPM demonstrates several benefits:

- Provides results that closely match observational data for orbital motion, orbital precession, Shapiro time delay, light deflection, and apparent black hole shadow diameter.
- Offers accessibility for undergraduate students to understand and implement.
- Includes MATLAB scripts for calculating N-body gravity simulations of the Solar System, Shapiro time delay, light deflection, apparent black hole shadow diameter, star clusters, and spacecraft trajectories (Pioneer 10, 11, Voyager 1, and 2).
- Demonstrates support for the equivalence principle through the [TMG](#) derivation, showing that local non-gravitational experiments are independent of velocity or position in a gravitational field.
- Shows that local properties of space govern gravitational motion.
- Applies equally to matter with or without rest mass.
- Accurately simulates motion and gravitational effects on the natural frequency of Earth-based, rocket-based, and satellite-based clocks.
- Provides a smooth, continuous logarithmic solution even under extreme gravitational conditions. In general relativity, the Schwarzschild coordinate representation becomes singular at the event horizon, and those coordinates no longer retain their usual interpretation inside it.
- Avoids point-to-point mass acceleration calculations, unlike the PPN method.
- Reduces solution time by approximately 80% (~5x faster) relative to PPN with similar accuracy for solar system-scale simulations, and it is only three times slower than Newtonian Mechanics.
- Separates the curvature of motion or acceleration vector and the gradient of space calculations, offering flexibility in numerical approaches for N-body simulations.
- Enables trajectory integration with angular rotation steps, yielding variable time steps that improve numerical efficiency.
- The example results indicate that CWPM is valid for both fixed and moving reference frames.

12. Recommendations

- Extend derivations to incorporate gravitational time-delay and frame dragging effects.
- Quantify the combined influence of rest mass variation and gravitational time delay on the Solar System solution.
- Demonstrate by derivation that CWPM is valid for both fixed and moving reference frames.

13. Acknowledgments

This work was made possible through access to online scientific educational materials, reports, and data.

I am grateful to NASA for providing exceptional access to data via the JPL Horizons system and the technical reports necessary to create these simulations and benchmark results against observational data.

I extend my appreciation to Dr. S. P. Pogossian for guidance in representing Mercury's precession.

I thank Dr. Nico Kuijpers and Marco Brassé for encouraging the derivation of the CWPM acceleration vector (Equation 14) and evaluating it using the SolarSystemSimulator. Their work confirmed that, for consistency, a CWPM simulation initialized with vectors from the Horizons system (based on Barycentric Dynamical Time) should be modeled with both the standard speed of light and the speed of light away from the Solar System's influence set equal to 299,792,458 m/s. I also thank Dr. Nico Kuijpers for independently implementing and benchmarking the CWPM method and verifying that the influence of variable rest mass on the Solar System simulation is minimal.

14. Author's Reflections on Science and Nature

This effort has given me great cause to reflect and marvel at the beauty of nature.

I cannot find words to express my gratitude for the equivalence principle as it emerges in CWPM (Section 6.12). This characteristic of natural law enables us to experience daily life in an orderly fashion while flying through space at an incomprehensible velocity alongside everything around us.

Putting forth the effort to create the Example 6 simulation has given me great admiration for those who planned and conducted the Pioneer and Voyager programs. It's incredible that the scientists and engineers accurately predicted the necessary course corrections, enabling them to design spacecraft with sufficient fuel to ensure the successful completion of each mission! I believe it is miraculous that we have gained enough knowledge by [scientific and religious methods \(Anderson, M. G., 2021\)](#) to conduct these missions.

I am amazed at how much time and effort it takes to calculate an N-body solution. The Solar System does not need to do it. It follows its course by obeying natural law and thereby enjoys a beautiful existence, enabling us to come along for the ride!

15. Author Declarations

Authorship: This work was solely authored by Morris G. Anderson (Independent Researcher).

Editorial Assistance Statement: Digital copy-editing and proofreading software were utilized strictly for syntax, grammar, and linguistic formatting refinement. All scientific concepts, physical frameworks, mathematical derivations, numerical simulations, and conclusions are entirely the original intellectual work of the author, who assumes full responsibility for the content.

Verification Statement: To ensure research integrity, all literature references, planetary data sets, and historical citations were manually cross-referenced and verified against primary published documents. The author has reviewed the final text and stands by its empirical and analytical accuracy.

Conflict of Interest: The author declares no competing financial interests.

Clarification of status (self-published): This manuscript is an independent work, benefiting only from informal colleague feedback, and has not undergone formal peer review or academic institutional evaluation.

References

- Anderson, M. G., 2004
[Time, Matter, and Gravity](#)
<https://vixra.org/abs/2102.0123> <https://doi.org/10.13140/RG.2.2.34401.33125>
- Anderson, M. G., 2007
[A Method for Predicting Quasar Luminosity Consistent With the NASA/IPAC Extragalactic Database](#) <https://vixra.org/abs/2102.0165> <https://vixra.org/pdf/2102.0165v1.pdf>
- Anderson, M. G., 2021
[How To Gain Knowledge By Scientific and Religious Methods](#)
<https://vixra.org/abs/2105.0160> <https://doi.org/10.13140/RG.2.2.25913.49760>
- Anderson, M. G., 2021
[The Curvature of Motion in a Gravitational Field](#)
<https://doi.org/10.13140/RG.2.2.32663.43685> <https://vixra.org/pdf/2106.0147v1.pdf>
- Blanco, V. M. and McCuskey, S. W. (1961) Basic Physics Of The Solar System page 217
<https://archive.org/details/basicphysicsofso0000blan/page/216/mode/2up>
- Bruns, D. G., 2018
 Gravitational Starlight Deflection Measurements during the 21 August 2017 Total Solar Eclipse
 Submitted on 31 Jan 2018 <https://doi.org/10.48550/arXiv.1802.00343>
- Carroll, S. M., 1997
[LECTURE NOTES ON GENERAL RELATIVITY](#)
<https://doi.org/10.48550/arXiv.gr-qc/9712019>
- Clemence, G. M., 1947
[The Relativity Effect in Planetary Motions](#)
 Rev. Mod. Phys. 19, 361 – Published 1 October 1947
<https://doi.org/10.1103/RevModPhys.19.361>
- Edvardsson, S., 2023
[Relativistic gravitational force](#)
 Celest Mech Dyn Astron 135, 25 (2023).
<https://doi.org/10.1007/s10569-023-10138-3>
- Einstein, A., et al., with notes by Sommerfeld, A. (1952)
 The principle of relativity : a collection of original memoirs on the special and general theory of relativity page 40 and 117. Dover Publications ISBN: 9780486600819
https://archive.org/details/principleofrelat00lore_0/page/116/mode/2up
 see also https://myweb.rz.uni-augsburg.de/~eckern/adp/history/einstein-papers/1916_49_769-822.pdf
 1916 No. 7. ANNALEN DER PHYSIK. VIERTE FOLGE. BAND 49.
 1. Die Grundlage der allgemeinen Relativitätstheorie; von A. Einstein. Page 775
 Wir gelangen also zu dem Ergebnis: In der allgemeinen Relativitätstheorie können Raum- und Zeitgrößen nicht so definiert werden, daß räumliche Koordinatendifferenzen unmittelbar mit dem Einheitsmaßstab, zeitliche mit einer Normal-uhr gemessen werden könnten.
 Translated by google on 2025-08-11 as:
 1916 No. 7. ANNALS OF PHYSICS. FOURTH SERIES. VOLUME 49.
 1. The Foundations of the General Theory of Relativity; by A. Einstein. Page 775
 We therefore arrive at the following conclusion: In general relativity, spatial and temporal quantities cannot be defined in such a way that spatial coordinate differences can be measured directly with a standard scale, and temporal ones with a standard clock.

Fimmel, Richard O. *et al.*, 1980

NASA SP-446 PIONEER FIRST TO JUPITER, SATURN, AND BEYOND

https://atmos.nmsu.edu/data_and_services/atmospheres_data/SATURN/logs/nasa-sp-446-Pioneer-First-to-Jupiter-Saturn-and-Beyond.pdf

Folkner, William M. *et al.*, 2014

[The Planetary and Lunar Ephemerides DE430 and DE431](#)

IPN Progress Report 42-196 (2014)

JPL CL#13-547

https://ssd.jpl.nasa.gov/doc/de430_de431.html

https://ipnpr.jpl.nasa.gov/progress_report/42-196/196C.pdf

Ficarra, G. and Lousto, C. O., 2024

[Close Encounter of Three Black Holes III](#)

<https://doi.org/10.48550/arXiv.2406.11985>

Hall, Charlie F., and Smith, Martha A., 1971

Pioneer F/G: Spacecraft Operational Characteristics

[NASA/ NASA/TP-2019-220324](#)

<https://ntrs.nasa.gov/citations/20190030279>

Krogh, K., 1999, 2006 Gravitation Without Curved Space-time

<https://arxiv.org/abs/astro-ph/9910325v23>

<https://arxiv.org/pdf/astro-ph/9910325v23.pdf>

Kuijpers, Nico, 2022, 2025

[Solar System Simulator](#)

<https://github.com/nicokuijpers/SolarSystemSimulator>

Luminet, Jean-Pierre

[Seeing Black Holes: from the Computer to the Telescope](#)

<https://doi.org/10.48550/arXiv.1804.03909>

Lau, Mark

SIMO (<https://www.simo.ac>), an app for scientific computation on iOS

Running MATLAB on iOS

Hong Kong Baptist University

NASA 1980

[Voyager Backgrounder](#)

<https://ntrs.nasa.gov/citations/19810001583>

Park, Ryan S. *et al.*, 2017

[Precession of Mercury's Perihelion from Ranging to the MESSENGER Spacecraft](#)

2017 *AJ* **153** 121

<https://doi.org/10.3847/1538-3881/aa5be2>

Park, Ryan S. *et al.*, 2021

[The JPL Planetary and Lunar Ephemerides DE440 and DE441](#)

2021 *AJ* **161** 105

<https://doi.org/10.3847/1538-3881/abd414>

Pogossian, S. P., 2022

[Comparative study of Mercury's perihelion advance](#)

Celestial Mechanics and Dynamical Astronomy (2022) 134:33

<https://doi.org/10.1007/s10569-022-10085-5>

Pogossian, S. P., 2025

[ICARUS'S ORBIT AS THE ULTIMATE TEST FOR THE VULCAN HYPOTHESIS AND GENERAL RELATIVITY](#)

<http://dx.doi.org/10.48550/arXiv.2507.01569>

Robertson S. L., 2018 Where Einstein Got It Wrong

<https://vixra.org/abs/1808.0642>

<https://vixra.org/pdf/1808.0642v1.pdf>

Santos, Ignacio, 2019

[Simulation and Study of Gravity Assist Maneuvers](#)

<https://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-276462>

Seidelmann, P. K., 1992

[Explanatory Supplement to the Astronomical almanac 1992 page 316 Table 5.8.1](#)

Seidelmann, P. K., 1992 & 2006

[Explanatory Supplement to the Astronomical almanac : a revision to the Explanatory supplement to the astronomical ephemeris and the American ephemeris and nautical almanac 1992 & 2006 page 316 Table 5.8.1](#)

Shapiro, Irwin I., 1964

[Fourth Test of General Relativity](#)

Phys. Rev. Lett. 13, 789 – Published 28 December 1964

<https://doi.org/10.1103/PhysRevLett.13.789>

Shapiro, I. I., *et al.*, 1968

[Fourth Test of General Relativity: Preliminary Results](#)

Phys. Rev. Lett. 20, 1265 – Published 27 May 1968; Erratum [Phys. Rev. Lett. 21, 266 \(1968\)](#)

<https://doi.org/10.1103/PhysRevLett.20.1265>

Shapiro, I. I., *et al.*, 1971

[Fourth Test of General Relativity: New Radar Result](#)

Phys. Rev. Lett. 26, 1132 – Published 3 May 1971

<https://doi.org/10.1103/PhysRevLett.26.1132>

Standish, E. M. and Williams, J. G., 2006

[Orbital Ephemerides of the Sun, Moon, and Planets - see Tables 8.10.2 and 8.10.3](#)

https://www.researchgate.net/publication/232203657_Orbital_Ephemerides_of_the_Sun_Moon_and_Planets

https://ssd.jpl.nasa.gov/planets/approx_pos.html

Urban, S. E. and Seidelmann P. K., 2012

[The Explanatory Supplement to the Astronomical Almanac 3rd ed 2012 page 340](#)

Will, C. M., 1993

[Theory and Experiment in Gravitational Physics Revised Edition page 179 equation 7.51](#)

Appendix A

Observational Data

Keplerian Elements to Orbital Precession Calculation Method

Background

- Figure A1 defines the orbital elements.
- "...because observations of the planets are made with reference to geocentric coordinates, the perihelion measured is the perihelion relative to the equinox, ϖ , given by $\varpi = \omega + \Omega \cos i$ ([Will, C. M., 1993](#))."
- "The rate of precession of Mercury's perihelion along its orbit plane is typically represented as $\dot{\varpi} = \dot{\omega} + \dot{\Omega} \cos i$ ([Park, Ryan S. et al., 2017](#))."

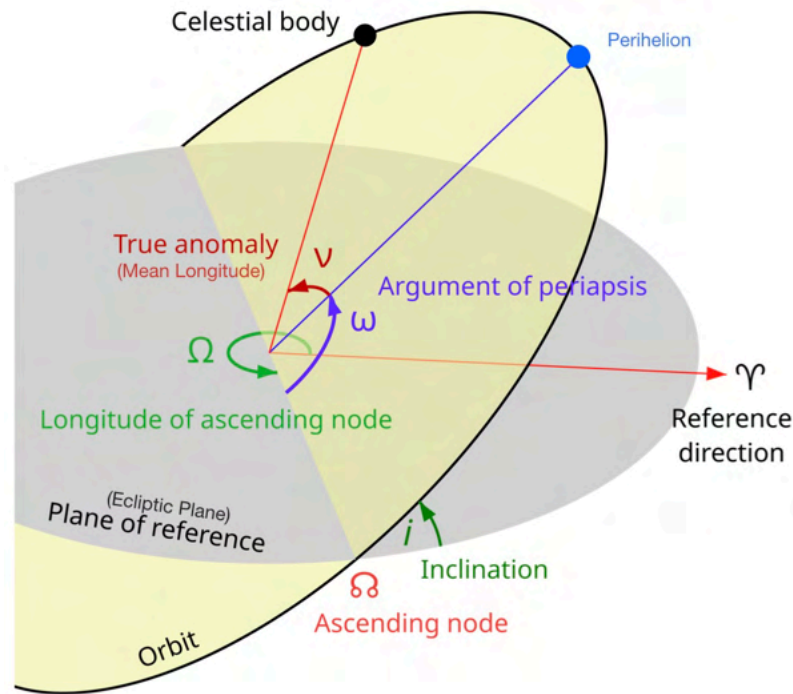


Figure A1 Orbital elements ([from wikipedia - adapted by author](#))

Definitions

With respect to Figure A1 for Keplerian elements in the supplement to the Astronomical Almanac ([Standish and Williams \(2006\) section 8.10.1](#)).

- ϖ_{KE} = the longitude of perihelion. Therefore, ϖ_{KE} is a compound angle with the first part in the reference plane and the second part in the orbital plane.
- $\varpi_{KE} = \Omega + \omega = \text{Longitude of ascending node} + \text{Argument of periapsis}$ (A1)
- $\dot{\varpi}_{KE}$ = the rate of change in the longitude of perihelion
- $\varpi = \omega + \Omega \cos i$ = the perihelion relative to the equinox (A2)
- $\dot{\varpi} = (\dot{\omega} + \dot{\Omega} \cos i)$ = the typical representation for the precession of Mercury (A3)
- $\dot{\varpi}_{in-plane}$ = in-plane orbital precession (A4)

Figure A2 illustrates the orbits of the inner planets and the location of the orbital perihelion.

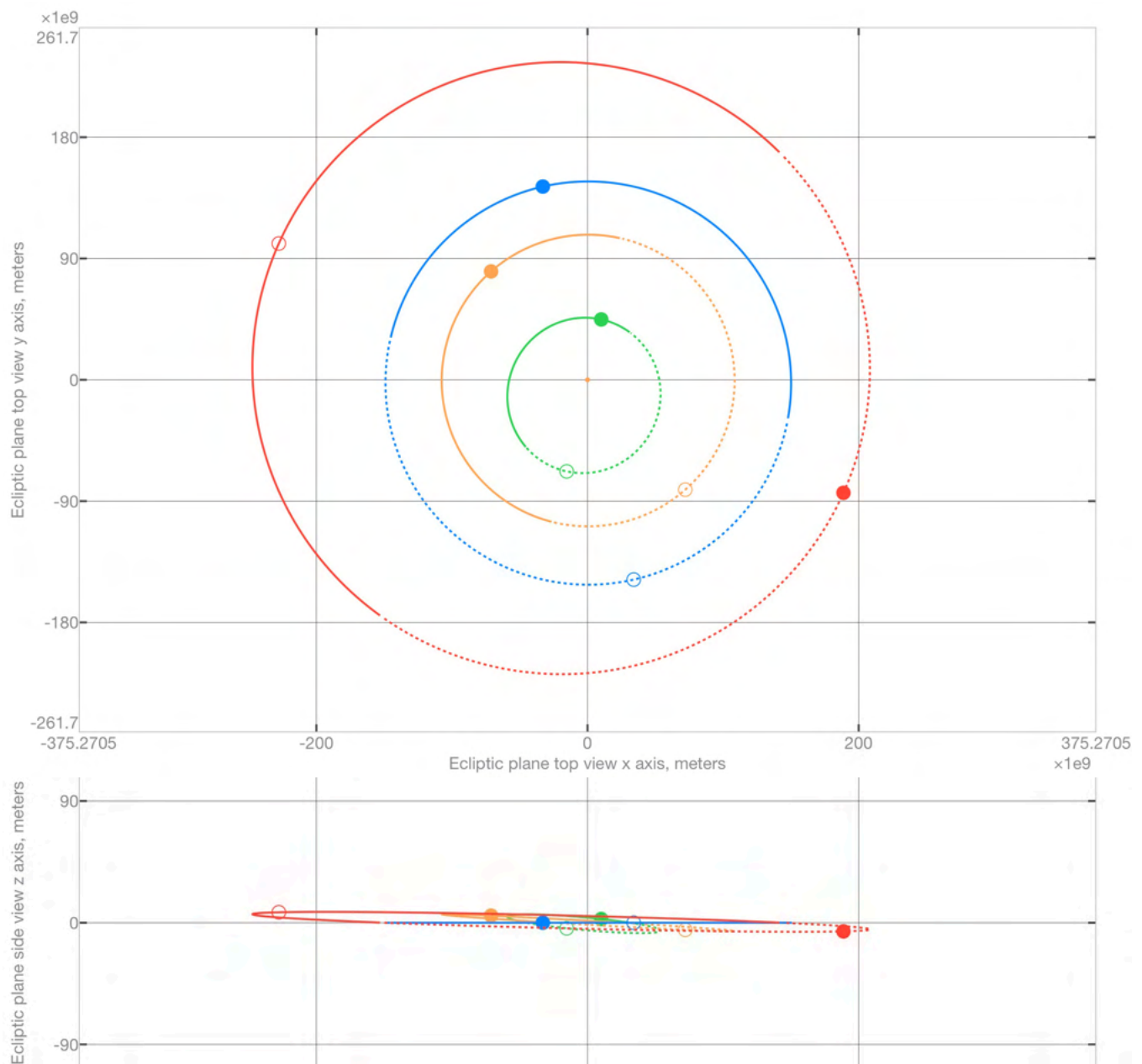


Figure A2 Orbits of Mercury, Venus, Earth, and Mars
The Curvature of Motion in a Gravitational Field ([Anderson, M. G., 2021](#)).

Upper plot illustrates orbits as viewed from above the ecliptic plane
 Lower plot illustrates orbits as viewed parallel to the ecliptic plane
 The solid symbol identifies the orbit perihelion, the open symbol indicates the aphelion.
 The dotted portion of the orbit is below the ecliptic plane, the solid portion is above.

The orbital precession calculated by the Example 1 MATLAB script file is $\dot{\vec{w}}_{in-plane}$. However, Mercury's precession is typically represented by $\dot{\vec{w}} = \dot{\omega} + \dot{\Omega} \cos i$. The following calculation demonstrates how to scale $\dot{\vec{w}}_{in-plane}$ to the typical expression of $\dot{\vec{w}} = \dot{\omega} + \dot{\Omega} \cos i$.

Calculate the In-Plane Precession From Keplerian Elements

Assumption

- The ratio of equation A3 to A4 is the same for the N-body integrated results as for the Keplerian elements that have been matched to observational data.

Process

From the Explanatory Supplement to the Astronomical almanac ([Standish and Williams \(2006\) Table 8.10.3](#)), obtain the longitude of the ascending node, the longitude of periapsis (or perihelion), and the orbital inclination, along with the corresponding rates of change. Calculate the precession in the orbital plane by numerical differentiation as follows:

1. Create two separate vectors by rotating the reference direction unit vector (the x-axis in Figure A2) to the direction defined in the Explanatory Supplement to the Astronomical Almanac to align with the perihelion direction depicted in Figure A1 for two different time intervals: one for $t = (\text{reference} - \Delta t)$ and another for $t = (\text{reference} + \Delta t)$.

Calculate the dot product of these two vectors and divide by $2\Delta t$ to obtain the precession of the perihelion within the orbital plane ($\dot{\varpi}_{in-plane_{KE}}$).

2. Calculate the typical expression for orbital precession from Keplerian elements as follows:

Substitute equation A1 into A2

$$\varpi = (\varpi_{KE} - \Omega) + \Omega \cos i$$

$$\varpi = \varpi_{KE} + \Omega(\cos i - 1)$$

Assuming the inclination, i , is constant

$$\dot{\varpi} = \dot{\varpi}_{KE} + \dot{\Omega}(\cos i - 1) \quad (\text{A5})$$

3. Scale $\dot{\varpi}_{in-plane}$ to the typical expression of $\dot{\varpi} = \dot{\omega} + \dot{\Omega} \cos i$ as:

$$\dot{\varpi} = \dot{\varpi}_{in-plane} \left(\frac{\dot{\varpi}_{KE} + \dot{\Omega}(\cos i - 1)}{\dot{\varpi}_{in-plane_{KE}}} \right) \quad (\text{A6})$$

The following results obtained by this method with the associated MATLAB files provide the orbital precession information for Mercury at Julian year 2000.

Observational Data

Keplerian Elements to Orbital Precession Calculation Results

Standish, E. M. and Williams, J. G. 2006

Orbital Ephemerides of the Sun, Moon, and Planets January 2006 Table 8.10.2

valid for the time interval 1800 AD - 2050 AD

For the following

Inclination	= 7.005 degrees
Inclination change	= -21.411 arcseconds/century
Longitude of Perihelion	= 77.458 degrees
Longitude of Perihelion change	= 577.72 arcseconds/century
Longitude of Ascending Node	= 48.331 degrees
Longitude of Ascending Node change	= -451.23 arcseconds/century

In-plane KE Perihelion precession = 582.303 arcseconds/century from Keplerian elements

Observed precession = 581.085 arcseconds/century calculated by Eq A5

Observed precession / In-plane KE Perihelion precession = 0.997908 see Eq A6

In-plane precession from 15-body simulation = 577.08 arcseconds/century - see Example 1

15-body simulation scaled to typical expression = 575.87 see Eq A6

Standish, E. M. and Williams, J. G. 2006

Orbital Ephemerides of the Sun, Moon, and Planets January 2006 Table 8.10.3

valid for the time interval 3000 BC - 3000 AD

For the following

Inclination	= 7.0056 degrees
Inclination change	= -21.246 arcseconds/century
Longitude of Perihelion	= 77.458 degrees
Longitude of Perihelion change	= 573.84 arcseconds/century
Longitude of Ascending Node	= 48.34 degrees
Longitude of Ascending Node change	= -439.71 arcseconds/century

In-plane KE Perihelion precession = 578.277 arcseconds/century from Keplerian elements

Observed precession = 577.123 arcseconds/century calculated by Eq A5

Observed precession / In-plane KE Perihelion precession = 0.998004 see Eq A6

In-plane precession from 15-body simulation = 577.08 arcseconds/century - see Example 1

15-body simulation scaled to typical expression = 575.93 see Eq A6

Seidelmann, P. K. 1992

Orbital Ephemerides of the Sun, Moon, and Planets 1992 page 316 Table 5.8.1

For the following

Inclination	= 7.0049 degrees
Inclination change	= -23.51 arcseconds/century
Longitude of Perihelion	= 77.456 degrees
Longitude of Perihelion change	= 573.57 arcseconds/century
Longitude of Ascending Node	= 48.332 degrees
Longitude of Ascending Node change	= -446.3 arcseconds/century

In-plane KE Perihelion precession = 578.03 arcseconds/century from Keplerian elements

Observed precession = 576.901 arcseconds/century calculated by Eq A5

Observed precession / In-plane KE Perihelion precession = 0.998047 see Eq A6

In-plane precession from 15-body simulation = 577.08 arcseconds/century - see Example 1

15-body simulation scaled to typical expression = 575.95 see Eq A6

>>

The results in yellow above are included in the text of this paper.

Information from the time period between 1800 AD and 2050 AD is not included because it deviates substantially from the long-term trend.

Observational Data: Almanac To Precession Calculation Script Files

Main File

```
% Keplerian_elements_best_fit.m
% Extract the orbital precession from Keplerian elements
clear all
clc

rdir = 1; % 1 = counter-clockwise rotation, -1 = clockwise rotation

t_delta = 0.01; % centuries. Used to calculate the precession rate

i_scalar = 1; % for scaling the Inclination
i_dot_scalar = 1; % for scaling the Inclination rate change

LP_scalar = 1.0; % for scaling the Longitude of Perihelion
LP_dot_scalar = 1.0; % for scaling the Longitude of Perihelion rate change

LA_scalar = 1.0; % for scaling the Longitude of ascending node
LA_dot_scalar = 1.0; % for scaling the Longitude of ascending node rate change

% disp(' ')
% disp(['t_delta = ' num2str(t_delta,5) ' century'])
% disp(['i_scalar = ' num2str(i_scalar,5) ' multiplier on orbital inclination'])
% disp(['i_dot_scalar = ' num2str(i_dot_scalar,5) ' multiplier on orbital inclination change'])

% From https://www.researchgate.net/publication/232203657_Orbital_Ephemerides_of_the_Sun_Moon_and_Planets
% Orbital Ephemerides of the Sun, Moon, and Planets
% January 2006
% In book: Explanatory Supplement to the Astronomical Almanac Chapter: 8 Publisher: University Science Books Editors: Seidelmann, P K
% Table 8.10.2
% Keplerian elements and their rates, with respect to the mean ecliptic and equinox of J2000, valid for the time-interval 1800 AD - 2050
AD.

% For Julian year 2000
Orb_Inc_Kepler_2000 = 7.00497902*i_scalar; % degrees, Orbital Inclination
Orb_Inc_Kepler_dot = -0.00594749*i_dot_scalar; % degrees/century, Orbital Inclination
Orb_LP_Kepler_2000 = 77.45779628*LP_scalar; % degrees, Orbital Longitude of Perihelion
Orb_LP_Kepler_dot = 0.16047689*LP_dot_scalar; % degrees, Orbital Longitude of Perihelion
Orb_LA_Kepler_2000 = 48.33076593*LA_scalar; % degrees, Orbital Longitude of Ascending Node
Orb_LA_Kepler_dot = -0.12534081*LA_dot_scalar; % degrees/century, Orbital Longitude of Ascending Node

t_cent = -t_delta; % For Julian year 2000 - t_delta
Orb_Inc_Kepler = rdir * (Orb_Inc_Kepler_2000 + t_cent * Orb_Inc_Kepler_dot);
Orb_LA_Kepler = rdir * (Orb_LA_Kepler_2000 + t_cent * Orb_LA_Kepler_dot);
Orb_LP_Kepler = rdir * (Orb_LP_Kepler_2000 + t_cent * Orb_LP_Kepler_dot);
Rotate_to_Perihelion % call script to rotate vectors
rhat1 = rhat;

t_cent = t_delta; % For Julian year 2000 + t_delta
Orb_Inc_Kepler = rdir * (Orb_Inc_Kepler_2000 + t_cent * Orb_Inc_Kepler_dot);
Orb_LA_Kepler = rdir * (Orb_LA_Kepler_2000 + t_cent * Orb_LA_Kepler_dot);
Orb_LP_Kepler = rdir * (Orb_LP_Kepler_2000 + t_cent * Orb_LP_Kepler_dot);
Rotate_to_Perihelion % call script to rotate vectors
rhat2 = rhat;

zdot_r = dot(rhat1(4,:),rhat2(4,:));
Precession_r = (acos(zdot_r)*180/pi)*3600;

disp(' ')
disp('Standish, E. M. and Williams, J. G. 2006')
disp('Orbital Ephemerides of the Sun, Moon, and Planets January 2006 Table 8.10.2')
disp('valid for the time interval 1800 AD - 2050 AD')
disp('For the following')
disp(['Inclination = ' num2str(Orb_Inc_Kepler_2000,5) ' degrees' ])
disp(['Inclination change = ' num2str(Orb_Inc_Kepler_dot*3600,5) ' arcseconds/century' ])
disp(['Longitude of Perihelion = ' num2str(Orb_LP_Kepler_2000,5) ' degrees' ])
disp(['Longitude of Perihelion change = ' num2str(Orb_LP_Kepler_dot*3600,5) ' arcseconds/century'])
disp(['Longitude of Ascending Node = ' num2str(Orb_LA_Kepler_2000,5) ' degrees' ])
disp(['Longitude of Ascending Node change = ' num2str(Orb_LA_Kepler_dot*3600,5) ' arcseconds/century' ])
disp(' ')
disp(['In-plane KE Perihelion precession = ' num2str(Precession_r/(2*t_cent),6) ' arcseconds/century from Keplerian elements'])
Observed_precession = (Orb_LP_Kepler_dot + Orb_LA_Kepler_dot * (cos(Orb_Inc_Kepler_2000*pi/180)-1))*3600;
disp(['Observed precession = ' num2str(Observed_precession,6) ' arcseconds/century calculated by Eq A5'])
disp(['Observed precession / In-plane KE Perihelion precession = ' num2str(((Observed_precession)/(Precession_r/(2*t_cent))))),6) ' see Eq A6'])
disp('In-plane precession from 15-body simulation = 577.08 arcseconds/century - see Example 1')
disp(['15-body simulation scaled to typical expression = ' num2str(577.08*((Observed_precession)/(Precession_r/(2*t_cent))))),5) ' see Eq A6'])
disp('-----')

% From https://www.researchgate.net/publication/232203657_Orbital_Ephemerides_of_the_Sun_Moon_and_Planets
% Orbital Ephemerides of the Sun, Moon, and Planets
% January 2006
% In book: Explanatory Supplement to the Astronomical Almanac Chapter: 8 Publisher: University Science Books Editors: Seidelmann, P K
% Table 8.10.3
% Keplerian elements and their rates, with respect to the mean ecliptic and equinox of J2000, valid for the time-interval 1800 AD - 2050
AD.

% For Julian year 2000
Orb_Inc_Kepler_2000 = 7.00559432*i_scalar; % degrees, Orbital Inclination
Orb_Inc_Kepler_dot = -0.00590158*i_dot_scalar; % degrees/century, Orbital Inclination
Orb_LP_Kepler_2000 = 77.45771895*LP_scalar; % degrees, Orbital Longitude of Perihelion
Orb_LP_Kepler_dot = 0.15940013*LP_dot_scalar; % degrees, Orbital Longitude of Perihelion
Orb_LA_Kepler_2000 = 48.33961819*LA_scalar; % degrees, Orbital Longitude of Ascending Node
Orb_LA_Kepler_dot = -0.12214182*LA_dot_scalar; % degrees/century, Orbital Longitude of Ascending Node
```

```

t_cent = -t_delta; % For Julian year 2000 - t_delta
Orb_Inc_Kepler = rdir * (Orb_Inc_Kepler_2000 + t_cent * Orb_Inc_Kepler_dot );
Orb_LA_Kepler = rdir * (Orb_LA_Kepler_2000 + t_cent * Orb_LA_Kepler_dot );
Orb_LP_Kepler = rdir * (Orb_LP_Kepler_2000 + t_cent * Orb_LP_Kepler_dot );
Rotate_to_Perihelion % call script to rotate vectors
rhat1 = rhat;

t_cent = t_delta; % For Julian year 2000 + t_delta
Orb_Inc_Kepler = rdir * (Orb_Inc_Kepler_2000 + t_cent * Orb_Inc_Kepler_dot );
Orb_LA_Kepler = rdir * (Orb_LA_Kepler_2000 + t_cent * Orb_LA_Kepler_dot );
Orb_LP_Kepler = rdir * (Orb_LP_Kepler_2000 + t_cent * Orb_LP_Kepler_dot );
Rotate_to_Perihelion % call script to rotate vectors
rhat2 = rhat;

zdot_r = dot(rhat1(4,:),rhat2(4,:));
Precession_r = (acos(zdot_r)*180/pi)*3600;

disp(' ')
disp('Standish, E. M. and Williams, J. G. 2006')
disp('Orbital Ephemerides of the Sun, Moon, and Planets January 2006 Table 8.10.3')
disp('valid for the time interval 3000 BC - 3000 AD')
disp('For the following')
disp(['Inclination = ' num2str(Orb_Inc_Kepler_2000,5) ' degrees' ])
disp(['Inclination change = ' num2str(Orb_Inc_Kepler_dot*3600,5) ' arcseconds/century' ])
disp(['Longitude of Perihelion = ' num2str(Orb_LP_Kepler_2000,5) ' degrees' ])
disp(['Longitude of Perihelion change = ' num2str(Orb_LP_Kepler_dot*3600,5) ' arcseconds/century'])
disp(['Longitude of Ascending Node = ' num2str(Orb_LA_Kepler_2000,5) ' degrees' ])
disp(['Longitude of Ascending Node change = ' num2str(Orb_LA_Kepler_dot*3600,5) ' arcseconds/century' ])
disp(' ')
disp(['In-plane KE Perihelion precession = ' num2str(Precession_r/(2*t_cent),6) ' arcseconds/century from Keplerian elements'])
Observed_precession = (Orb_LP_Kepler_dot + Orb_LA_Kepler_dot * (cos(Orb_Inc_Kepler_2000*pi/180)-1))*3600;
disp(['Observed precession = ' num2str(Observed_precession,6) ' arcseconds/century calculated by Eq A5'])
disp(['Observed precession / In-plane KE Perihelion precession = ' num2str(((Observed_precession/(Precession_r/(2*t_cent))))),6) ' see Eq A6'])
disp(['In-plane precession from 15-body simulation = 577.08 arcseconds/century - see Example 1'])
disp(['15-body simulation scaled to typical expression = ' num2str(577.08*((Observed_precession/(Precession_r/(2*t_cent))))),5) ' see Eq A6'])
disp('-----')

% From https://archive.org/search.php?query=external-identifier%3A%22urn%3Aoclc%3Arecord%3A1034676245%22
% Explanatory supplement to the Astronomical almanac : a revision to the Explanatory supplement to the
% astronomical ephemeris and the American ephemeris and nautical almanac
% by Seidelmann, P. Kenneth; United States Naval Observatory. Nautical Almanac Office; Great Britain. Nautical Almanac Office
% 2006 page 316 Table 5.8.1 Classical Keplerian elements at the epoch J2000
% Originally published: 1992

% For Julian year 2000
Orb_Inc_Kepler_2000 = 7.00487*i_scalar; % degrees, Orbital Inclination
Orb_Inc_Kepler_dot = (-23.51/3600)*i_dot_scalar; % degrees/century, Orbital Inclination
Orb_LP_Kepler_2000 = 77.45645*LP_scalar; % degrees, Orbital Longitude of Perihelion
Orb_LP_Kepler_dot = 573.57/3600*LP_dot_scalar; % degrees, Orbital Longitude of Perihelion
Orb_LA_Kepler_2000 = 48.33167*LA_scalar; % degrees, Orbital Longitude of Ascending Node
Orb_LA_Kepler_dot = -446.30/3600*LA_dot_scalar; % degrees/century, Orbital Longitude of Ascending Node

t_cent = -t_delta; % For Julian year 2000 - t_delta
Orb_Inc_Kepler = rdir * (Orb_Inc_Kepler_2000 + t_cent * Orb_Inc_Kepler_dot );
Orb_LA_Kepler = rdir * (Orb_LA_Kepler_2000 + t_cent * Orb_LA_Kepler_dot );
Orb_LP_Kepler = rdir * (Orb_LP_Kepler_2000 + t_cent * Orb_LP_Kepler_dot );
Rotate_to_Perihelion % call script to rotate vectors
rhat1 = rhat;

t_cent = t_delta; % For Julian year 2000 + t_delta
Orb_Inc_Kepler = rdir * (Orb_Inc_Kepler_2000 + t_cent * Orb_Inc_Kepler_dot );
Orb_LA_Kepler = rdir * (Orb_LA_Kepler_2000 + t_cent * Orb_LA_Kepler_dot );
Orb_LP_Kepler = rdir * (Orb_LP_Kepler_2000 + t_cent * Orb_LP_Kepler_dot );
Rotate_to_Perihelion % call script to rotate vectors
rhat2 = rhat;

zdot_r = dot(rhat1(4,:),rhat2(4,:));
Precession_r = (acos(zdot_r)*180/pi)*3600;

disp('For the following')
disp(['Inclination = ' num2str(Orb_Inc_Kepler_2000,5) ' degrees' ])
disp(['Inclination change = ' num2str(Orb_Inc_Kepler_dot*3600,5) ' arcseconds/century' ])
disp(['Longitude of Perihelion = ' num2str(Orb_LP_Kepler_2000,5) ' degrees' ])
disp(['Longitude of Perihelion change = ' num2str(Orb_LP_Kepler_dot*3600,5) ' arcseconds/century'])
disp(['Longitude of Ascending Node = ' num2str(Orb_LA_Kepler_2000,5) ' degrees' ])
disp(['Longitude of Ascending Node change = ' num2str(Orb_LA_Kepler_dot*3600,5) ' arcseconds/century' ])
disp(' ')
disp(['In-plane KE Perihelion precession = ' num2str(Precession_r/(2*t_cent),6) ' arcseconds/century from Keplerian elements'])
Observed_precession = (Orb_LP_Kepler_dot + Orb_LA_Kepler_dot * (cos(Orb_Inc_Kepler_2000*pi/180)-1))*3600;
disp(['Observed precession = ' num2str(Observed_precession,6) ' arcseconds/century calculated by Eq A5'])
disp(['Observed precession / In-plane KE Perihelion precession = ' num2str(((Observed_precession/(Precession_r/(2*t_cent))))),6) ' see Eq A6'])
disp(['In-plane precession from 15-body simulation = 577.08 arcseconds/century - see Example 1'])
disp(['15-body simulation scaled to typical expression = ' num2str(577.08*((Observed_precession/(Precession_r/(2*t_cent))))),5) ' see Eq A6'])
disp('-----')

```

Script For Rotating the Vectors

```
% Rotate_to_Perihelion.m

% Convert Kepler angles to radians
Orb_Inc_Kepler = Orb_Inc_Kepler * pi/180.0;
Orb_LA_Kepler  = Orb_LA_Kepler  * pi/180.0;
Orb_LP_Kepler  = Orb_LP_Kepler  * pi/180.0;

%   Define initial unrotated position and motion unit vectors for the orbital Perihelion
%   Counter-clockwise rotation with the Vernal Equinox in the +x direction or to the right.

%   Define the initial position unit vector
rhat(1,1) = 1.0;
rhat(1,2) = 0.0;
rhat(1,3) = 0.0;
rhat = rhat / sqrt( rhat(1,1)*rhat(1,1) + rhat(1,2)*rhat(1,2)+ rhat(1,3)*rhat(1,3) );

%   Define the initial motion unit vector
bhat(1,1) = 0.0;
bhat(1,2) = rdir * 1.0;
bhat(1,3) = 0.0;
bhat = bhat / sqrt( bhat(1,1)*bhat(1,1) + bhat(1,2)*bhat(1,2)+ bhat(1,3)*bhat(1,3) );

%   Rotate unit vectors to Perihelion starting point.   Use Rodrigues' rotation formula.

theta = Orb_LA_Kepler;
kr = cross(rhat(1,:),bhat(1,:));
vr1 = rhat(1,:);
vr2 = cos(theta)*vr1 + cross(kr,vr1)*sin(theta) + kr*(dot(kr,vr1))*(1 - cos(theta));
rhat(2,:) = vr2;
vr1 = bhat(1,:);
vr2 = cos(theta)*vr1 + cross(kr,vr1)*sin(theta) + kr*(dot(kr,vr1))*(1 - cos(theta));
bhat(2,:) = vr2;

theta = Orb_LP_Kepler - Orb_LA_Kepler;
kr = cross(rhat(1,:),bhat(1,:));
vr1 = rhat(2,:);
vr2 = cos(theta)*vr1 + cross(kr,vr1)*sin(theta) + kr*(dot(kr,vr1))*(1 - cos(theta));
rhat(3,:) = vr2;
vr1 = bhat(2,:);
vr2 = cos(theta)*vr1 + cross(kr,vr1)*sin(theta) + kr*(dot(kr,vr1))*(1 - cos(theta));
bhat(3,:) = vr2;

theta = Orb_Inc_Kepler;
kr = rhat(2,:);
vr1 = rhat(3,:);
vr2 = cos(theta)*vr1 + cross(kr,vr1)*sin(theta) + kr*(dot(kr,vr1))*(1 - cos(theta));
rhat(4,:) = vr2;
vr1 = bhat(3,:);
vr2 = cos(theta)*vr1 + cross(kr,vr1)*sin(theta) + kr*(dot(kr,vr1))*(1 - cos(theta));
bhat(4,:) = vr2;
```

Appendix B

MATLAB Script and Data Files

Example script file structure

1. Define input setup
2. Call data file if needed
3. Initialize the program by calling
TMG_nb_initialization.m
4. Calculate path by calling
TMG_nb_path_dt_integration.m
This integrator applies a constant time step to calculate the trajectory.
or
TMG_nb_path_dtheta_integration.m
This integrator calculates and applies a unique Δt for each time step, limiting the maximum rotation for each solution step to less than the user-defined value.
Note: For each solution step, each body can experience a different amount of path rotation.
5. Plot and or save output

The following files are also available for download at: <https://www.mathworks.com/matlabcentral/fileexchange/178949-n-body-gravity-simulation-by-curvature-of-wave-propagation>

Solver, Example, and Associated Data Script Files.	Page
Background	42
Definitions	42
Calculate the In-plane Precession From Keplerian Elements	44
Assumption	44
Process	44
Observational Data: Almanac to Precession Calculation Script Files	46
Background_and_Nomenclature.m	50
TMG_nb_initialization.m	51
TMG_nb_path_dt_integration.m	52
TMG_nb_path_dt_integration_dvdt.m	54
TMG_nb_path_dtheta_integration.m	57
Example_1_the_solar_system.m	59
Example_1_the_solar_system_dvdt.m	62
Example_2_Shapiro_Time_delay.m	65
Example_2_Shapiro_Time_delay_dvdt.m	69
Example_3_Apparent_black_hole_shadow.m	73
Example_4_Simple_trajectory.m	76
Example_5_Star_cluster_random.m	79
Example_5_Star_cluster_symmetric.m	82
Example_6_Pioneer_10_11_Voyager_1_2.m	85
Data file: zSS_1970_01_25_downloaded_on_2023_06_26.m	93
Data file: zSS_PS_1973_04_12.m	95
Data file: zSS_2000_01_01_PS_downloaded_on_2023_02_23.m	99

Background_and_Nomenclature.m

```
% Background_and_Nomenclature.m
%
% -----
% Created by Morris G. Anderson
% https://www.researchgate.net/profile/Morris-Anderson
%
% Documented in the Paper
% N-body Gravity Simulation by Curvature of Wave Propagation
%
% Theory published in the book:
%   Time Matter and Gravity
%   by
%   Morris G. Anderson
%   Copyright 2004
%
% This book can be downloaded from:
%   https://vixra.org/abs/2102.0123   https://vixra.org/pdf/2102.0123v1.pdf
%
% See also
%   https://vixra.org/author/morris_g_anderson
%   https://www.mathworks.com/matlabcentral/fileexchange/94835-the-curvature-of-motion-in-a-gravitational-field
%   Plan is to make this script file available at
%   https://www.mathworks.com/matlabcentral/fileexchange/178949-n-body-gravity-simulation-by-curvature-of-wave-propagation
% -----
%
% beta_option = 0 for initial motion input as v
%   vx = initial x velocity.  units = m/s
%   vy = initial y velocity.  units = m/s
%   vz = initial z velocity.  units = m/s
%
% beta_option = 1 for initial motion input as v/c
%   bx = (initial x velocity) / c
%   by = (initial y velocity) / c
%   bz = (initial z velocity) / c
%
% UNITS
% length = meters
% time = seconds
% mass = kg
%
% VARIABLES
% ai = The angle of rotation for each body per time step
% arc_count = The number of calculated arc segments
% b = v/c for each body
% b_ave = The average b for the arc segment rotation
% bl, b2 = v/c for each body at the beginning and end of each arc segment
% bu1, bu2 = v/c unit vectors for each body at the beginning and end of each arc segment
% c = The value of c at the location of each body. The influence of the individual body
%   is ignored when calculating the speed of light at its location.
% c1, c2 = c for each body at the beginning and end of each arc segment
% cs = The standard speed of light, see Time Matter and Gravity (TMG) section 2.4.1.
% c_inf = c away from the N-body simulation
% dbeta = The change in v/c between the beginning and end of each arc segment
%
% dotbugc1, dotbugc2 = dot product between the motion and the gradient of c vectors
%   at the beginning and end of each arc segment
% Drx, Dry, Drz = The distance between each body
% Drxu, Dryu, Drzu = The distance unit vector between each body
%
% Dru = The distance unit vector magnitude = 1 except when the separation = 0,
%   if so the magnitude = 0
% dt = The time step
% g = Newton's universal gravitational constant. Units = m^3/(kg-s)
% gc1, gc2 = The magnitude of the gradient of c for each body at the beginning and end of
%   each arc segment
% gcu1, gcu2 = The gradient of c unit vector for each body at the beginning and end of each
%   arc segment
% k1, k2 = The axis of rotation at the beginning and end of each arc segment
% mb = The rest mass for each body
% nac = The number of time steps or arc segments per point that is saved for plotting
% nb = The number of governing bodies
% n_orbit = The number of stored perihelion points for the orbital precession calculation
% np = The total number of points that can be plotted
% ni = The total number of time steps
% nps = The number of points saved for plotting
% b_pr = The body for calculating orbital precession
% rca = The average curvature of motion for each arc segment
% rc1, rc2 = The curvature of motion for each body at the beginning and end of each arc segment
% rcu1, rcu2 = The curvature of motion unit vector for each body at the beginning and end of
%   each arc segment
% rx, ry, rz = The x, y, z position of each body
% tau = 2*g/cs/cs
% tp = Time at perihelion for the precession calculation
% tt = The total path time
% t1, t2 = The time for each body at the beginning and end of each arc segment
% VX, VY, VZ = The initial VX, VY, VZ velocities of each body
% X, Y, Z = The initial X, Y, Z positions of each body
% zd = 1 or -1. If this value is negative the orbital rotation is reversed
% zlow = A limiting value to prevent dividing by zero
```

TMG_nb_initialization.m

```
% TMG_nb_initialization.m
%-----
% Created by Morris G. Anderson
%   https://www.researchgate.net/profile/Morris-Anderson
%
% Units, nomenclature, and documentation links provided in the accompanying script file:
%   Background_and_Nomenclature.m
%
% Theory provided in the paper: N-body Gravity Simulation by Curvature of Wave Propagation
%                               (Preprint published on ResearchGate)
%                               The radial acceleration vector is derived from Equation TMG Eq 7-9
%                               The tangential acceleration vector is derived from Equation 5
%                               This paper will be revised to include the derivation, with a tentative
%                               revision date of Fall 2025.
%
% The acceleration vector is:
% dvdt1 = ((3.*b1.^2 - 1)/2).*c1.*dot(gc1, bul).*bul - ((b1.^2 + 1)/2).*c1.*dot(gc1, rcul).*rcul;
%-----
% Initialize starting conditions
n_orbit = 0; % The number of times passing the perihelion
rx1 = rx(1,1:nb)';
ry1 = ry(1,1:nb)';
rz1 = rz(1,1:nb)';
% Calculate the distance between each body. zlow is the limiting value to prevent dividing by zero.
Drx = rx1' - rx1;
Dry = ry1' - ry1;
Drz = rz1' - rz1;
Dr = (Drx.^2 + Dry.^2 + Drz.^2 + zlow).^0.5;
% Calculate the unit vector defining the direction of separation. The unit vector magnitude is
% equal to 1 except when the separation = 0, if so the magnitude = 0. Note that Dru has a
% magnitude of 1 or zero, therefore, sqrt(1) = 1. This eliminates the need to take the sqrt.
Drxu = Drx ./Dr;
Dryu = Dry ./Dr;
Drzu = Drz ./Dr;
Dru = (Drxu.^2 + Dryu.^2 + Drzu.^2);
% Calculate the speed of light at the location of each body. The influence of the individual body
% is ignored when calculating the speed of light at its location
phil = (-tau) * sum(mb .* Dru ./ Dr);
dphil = 1; % initial value
dphi2 = 1; % initial value
c1 = c_inf * exp(phil);
c(1,:) = c1; % Store value just in case it is desired for post-processing
% Convert input velocities to beta
for zn = 1:nb
    if beta_option(zn) == 0
        bx(1,zn) = vx(1,zn)/c(1,zn);
        by(1,zn) = vy(1,zn)/c(1,zn);
        bz(1,zn) = vz(1,zn)/c(1,zn);
    end
end
beta_option(1:nb) = 1;
% calculate beta unit vector
bx1 = bx(1,1:nb)' + zlow;
by1 = by(1,1:nb)';
bz1 = bz(1,1:nb)';
b1 = ((bx1.^2 + by1.^2 + bz1.^2 + zlow).^0.5)';
% Calculate the direction of the motion unit vector. The unit vector is the same for b and v
bul = [bx1' ./b1
        by1' ./b1
        bz1' ./b1];
b(1,1:nb) = b1; % Store value just in case it is desired for post-processing
% Calculate the gradient of c
gc1 = [c1 .* (tau * sum(mb .* Dru .* Drxu .* Dr.^(-2)))
        c1 .* (tau * sum(mb .* Dru .* Dryu .* Dr.^(-2)))
        c1 .* (tau * sum(mb .* Dru .* Drzu .* Dr.^(-2)))];
gcul = gc1 ./ ((gc1(1,:).^2 + gc1(2,:).^2 + gc1(3,:).^2).^(1/2));
% Calculate the unit rotation vector which, by definition (see TMG Figure 13 and eq. 7-1),
% is equal to the cross-product of the gradient of c and the motion unit vector.
% zlow is the limiting value to prevent dividing by zero.
k1 = cross(gcul,bul);
k1 = k1 ./ ((k1(1,:).^2 + k1(2,:).^2 + k1(3,:).^2).^(1/2));
ka = k1;
% Calculate the direction and magnitude of the curvature vector (see TMG Figure 13 and eq. 7-1), at
% the starting point. This is done by the cross-product of motion and the rotation vector.
rcul = cross(bul,k1);
rc1 = (2*c1./(1+1./(b1.^2)))./(dot(gc1, rcul)+zlow);
rc2 = rc1;
rca = rc1;

rc(1,1:nb) = rca; % Store value just in case it is desired for post-processing
dotbugc1 = dot(bul, gc1); % Calculate the dot product between motion and the gradient of c vectors.

% The following code is included for the Runge-Kutta integrator,
% TMG_nb_path_dt_integration_dvdt.
% It does not interfere with the other solvers.

v1 = [bx1' .*c1
        by1' .*c1
        bz1' .*c1];
dvdt1 = ((3.*b1.^2 - 1)/2).*c1.*dot(gc1, bul).*bul - ((b1.^2 + 1)/2).*c1.*dot(gc1, rcul).*rcul;
```

TMG_nb_path_dt_integration.m

```
% TMG_nb_path_dt_integration.m
%
% Integrates the path with a constant time step
%-----
% Created by Morris G. Anderson
%   https://www.researchgate.net/profile/Morris-Anderson
%
% Units, nomenclature, and documentation links provided in the accompanying script file:
%   Background_and_Nomenclature.m
%
% Theory provided in the paper: N-body Gravity Simulation by Curvature of Wave Propagation
%-----
% tau = 2*g/cs/cs; % Defined in the input file

for arc_count = 1:nac % The following logic keeps all of the bodies in time with each other
    ai = (dt * ((n-1) * nac + arc_count) - (t1-tstart)) .* c1 ./ rca;
    % Rotate unit vectors with Rodrigues' formula to calculate position and motion at point 2
    bu2 = bu1 .* cos(ai) + cross(ka,bu1) .* sin(ai) + ka .* dot(ka, bu1) .* (1 - cos(ai));
    bu2 = bu2 ./ ((bu2(1,:).^2 + bu2(2,:).^2 + bu2(3,:).^2 ).^(1/2));
    % A portion of the Rodrigues formula is replaced with the following to reduce numerical error
    vtu = bu1 + bu2;
    vtu = vtu ./ ((vtu(1,:).^2 + vtu(2,:).^2 + vtu(3,:).^2 ).^(1/2));
    vt = 2.0 * rca .* sin(ai/2);
    rx2 = rx1 + vtu(1,:)' .* vt';
    ry2 = ry1 + vtu(2,:)' .* vt';
    rz2 = rz1 + vtu(3,:)' .* vt';
    % Calculate conditions for point 2. Start by calculating the distance (Dr) between each body
    Drx = rx2' - rx2;
    Dry = ry2' - ry2;
    Drz = rz2' - rz2;
    gc2 = (Drx.^2 + Dry.^2 + Drz.^2 + zlow); % Temporary storage for distance^2 between bodies
    Dr = gc2.^0.5;
    % Calculate the unit vector that defines the direction of separation. The unit vector magnitude is
    % equal to 1 except when the separation = 0, if so the magnitude = 0. Dru has a magnitude of 1 or zero,
    % therefore, sqrt(1) = 1. This eliminates the need to take the sqrt.
    Drxu = Drx ./ Dr;
    Dryu = Dry ./ Dr;
    Drzu = Drz ./ Dr;
    Dru = (Drxu.^2 + Dryu.^2 + Drzu.^2);
    % Calculate the speed of light at the location of each body (see TMG equations 4-10 through 4-14).
    % The influence of the individual body is ignored when calculating the speed of light at its location.
    phi2 = (-tau) * sum(mb .* Dru ./ Dr); % tau = 2*g/cs/cs as defined in the file that calls this script.
    c2 = c_inf .* exp(phi2);
    dphi2 = phi2(b_pr)-phi1(b_pr); % This value is used in finding the perihelion time for the body of interest
    % Calculate the gradient of c
    gc2 = c2 .* tau .* mb .* Dru ./ gc2; % Initial gc2 value equals distance^2 between bodies (see comments above)
    gc2 = [sum(gc2 .* Drxu)
           sum(gc2 .* Dryu)
           sum(gc2 .* Drzu)];
    % Calculate b2
    drby2c_ave = (rca .* ai) ./ (c1 + c2);
    dotbugc2 = dot( bu2, gc2);
    dotbugc_ave = (dotbugc1 + dotbugc2)/2;

    dbeta = ((b1.^2 - 1)./b1) .* (drby2c_ave .* dotbugc_ave);
    % Repeat a few times to get a solution with sufficient accuracy.
    b_ave = (2*b1 + dbeta)/2;
    dbeta = ((b_ave.^2 - 1)./b_ave) .* (drby2c_ave .* dotbugc_ave);
    b_ave = (2*b1 + dbeta)/2;
    dbeta = ((b_ave.^2 - 1)./b_ave) .* (drby2c_ave .* dotbugc_ave);
    b2 = b1 + dbeta;
    % Calculate the unit rotation vector (see TMG Figure 13 and eq. 7-1), which
    % is equal to the cross-product of the gradient of c and the motion unit vector.
    % Then calculate the rotation vector magnitude.
    k2 = cross(gc2,bu2);
    k2 = k2 ./ ((k2(1,:).^2 + k2(2,:).^2 + k2(3,:).^2 ).^(1/2));
    rcu2 = cross(bu2,k2);
    rcu2 = rcu2 ./ ((rcu2(1,:).^2 + rcu2(2,:).^2 + rcu2(3,:).^2 ).^(1/2));
    rc2 = (2*c2./(1+1./(b2.^2)))./(dot( gc2, rcu2)+zlow);
    % calculate path time
    t2 = t1 + ai .* rca ./ ((b1 .* c1 + b2 .* c2) / 2);
    % Find and save the perihelion time for the body of interest
    if dphi1 <= 0 && dphi2 >= 0 % The perihelion is between these values

        disp(['Orbit count = ' num2str(n_orbit) ' from 1st gravitational potential minimum for body ' num2str(b_pr,3)])
        n_orbit = n_orbit + 1;

        tp(n_orbit) = (t1(b_pr)+t2(b_pr))/2 - dphi2 * (t2(b_pr)-t1(b_pr)) / ((dphi2-dphi1)+zlow); % at the perihelion
        cp = (c1(b_pr)+c2(b_pr))/2 - dphi2 * (c2(b_pr)-c1(b_pr)) / ((dphi2-dphi1)+zlow); % at the perihelion
        bp = (b1(b_pr)+b2(b_pr))/2 - dphi2 * (b2(b_pr)-b1(b_pr)) / ((dphi2-dphi1)+zlow); % at the perihelion
        % Rotate unit vectors with Rodrigues' formula to the position and motion at the minimum gravitational potential
        zai = ai(b_pr) * (tp(n_orbit) - t1(b_pr)) / (t2(b_pr) - t1(b_pr));

        bpu_n = bu1(:,b_pr) .* cos(zai) + cross(ka(:,b_pr),bu1(:,b_pr)) .* sin(zai) + ka(:,b_pr) .* dot(ka(:,b_pr), bu1(:,b_pr)) .* (1 - cos(zai));

        bpu_n = bpu_n ./ ( (bpu_n(1).^2 + bpu_n(2).^2 + bpu_n(3).^2 ).^(1/2) ); % beta unit vector at the perihelion

        vp = bp .* cp .* bpu_n - vg'; % local velocity at the perihelion (absolute velocity minus group velocity)
        vpu_n = vp ./ ( (vp(1).^2 + vp(2).^2 + vp(3).^2 ).^(1/2) ); % velocity unit vector at the perihelion

        if n_orbit==1
            vpu_1 = vpu_n;
            zdot = 1;
        else
            zdot = dot(vpu_1',vpu_n');
        end
    end
end
```

```

        % Determine if the precession is positive or negative
        zcross = cross(vpu_1',vpu_n);
        zcross = zcross ./ sqrt((zcross(1).^2 + zcross(2).^2 + zcross(3).^2 ).^(1/2));
        prsine = zcross + ka(:,b_pr)';
        prsine = (prsine(1).^2 + prsine(2).^2 + prsine(3).^2 ).^(1/2);
        if prsine > 1.0
            zprecession(n_orbit) = acos(zdot);
        else
            zprecession(n_orbit) = -acos(zdot);
        end
    end
    if plot_option_1 == 1
    else
        plot(tp/(365.25*24*3600*100), zprecession * (3600*180/pi)); % precession per century
    end

end

% use linear extrapolation to estimate the next rca, for this step rca = (rc2 + rc1) ./ 2
% for the next step rc1 = rc2, therefore
rca = rc2 .* (rc2 + rc1) ./ (2 * rc1);
ka = k2 + (k2 - k1) / 2;
ka = ka ./ ( (ka(1,:).^2 + ka(2,:).^2 + ka(3,:).^2 ).^(1/2) );

b1 = b2;
bu1 = bu2;
c1 = c2;
gc1 = gc2;
k1 = k2;
phi1 = phi2;
dphi1 = dphi2;
rc1 = rc2;
rx1 = rx2;
ry1 = ry2;
rz1 = rz2;
t1 = t2;

dotbugc1 = dotbugc2;
end

```

TMG_nb_path_dt_integration_dvdt.m

```
% TMG_nb_path_dt_integration_dvdt.m
%
% The Runge-Kutta method is used to integrate the path with a constant time step.
%-----
% Created by Morris G. Anderson
%   https://www.researchgate.net/profile/Morris-Anderson
%
% Units, nomenclature, and documentation links provided in the accompanying script file:
%   Background_and_Nomenclature.m
%
% Theory provided in the paper:  N-body Gravity Simulation by Curvature of Wave Propagation
%                               (Preprint published on ResearchGate)
%                               The radial acceleration vector is derived from Equation TMG Eq 7-9
%                               The tangential acceleration vector is derived from Equation 5
%                               This paper will be revised to include the derivation, with a tentative
%                               revision date of Fall 2025.
%
% The acceleration vector is:
% dvdt1 = ((3.*b1.^2 - 1)/2). *c1.*dot(gc1, bu1).*bu1 - ((b1.^2 + 1)/2). *c1.*dot(gc1, rcu1).*rcu1;
%-----
% tau = 2*g/cs/cs;  % Defined in the input file

for arc_count = 1:nac

    % dvdt1 = k1 for Runge-Kutta starting point, point 1
    % Calculate velocity and position at Runge-Kutta point 2

    dtb2 = dt/2;
    v2 = v1 + dvdt1*dtb2;  % dvdt1 = k1 for Runge-Kutta
    rx2 = rx1+(dtb2*(v1(1,:)+v2(1,:))/2)'; ry2 = ry1+(dtb2*(v1(2,:)+v2(2,:))/2)'; rz2 = rz1+(dtb2*(v1(3,:)+v2(3,:))/2)';
    % Calculate conditions for point 2. Start by calculating the distance (Dr) between each body
    Drx = rx2' - rx2; Dry = ry2' - ry2; Drz = rz2' - rz2;
    Dr = (Drx.^2 + Dry.^2 + Drz.^2 + zlow).^0.5; % Temporary storage for distance^2 between bodies
    % Calculate the unit vector that defines the direction of separation. The unit vector magnitude is
    % equal to 1 except when the separation = 0, if so the magnitude = 0. Dru has a magnitude of 1 or zero,
    % therefore, sqrt(1) = 1. This eliminates the need to take the sqrt.
    Drxu = Drx ./ Dr; Dryu = Dry ./ Dr; Drzu = Drz ./ Dr;
    gc2 = (Drx.^2 + Dry.^2 + Drz.^2 + zlow); % Temporary storage for distance^2 between bodies
    Dr = gc2.^0.5;
    % Calculate the speed of light at the location of each body (see TMG equations 4-10 through 4-14).
    % The influence of the individual body is ignored when calculating the speed of light at its location.
    phi2 = (-tau) * sum(mb .* Dru ./ Dr);  % tau = 2*g/cs/cs as defined in the file that calls this script.
    c2 = c_inf * exp(phi2);
    % Calculate beta for each body
    b2 = ((v2(1,:).^2 + v2(2,:).^2 + v2(3,:).^2).^0.5)./c2;
    bu2 = (v2./c2)./b2;
    % Calculate the gradient of c
    gc2 = c2 .* tau .* mb .* Dru ./ gc2;  % Initial gc2 value equals distance^2 between bodies (see comments above)
    gc2 = [sum(gc2 .* Drxu)
           sum(gc2 .* Dryu)
           sum(gc2 .* Drzu)];
    % Calculate the unit rotation vector (see TMG Figure 13 and eq. 7-1)
    k2 = cross(gc2,bu2);
    k2 = k2 ./ ((k2(1,:).^2 + k2(2,:).^2 + k2(3,:).^2 ).^(1/2));
    % Calculate the radius of curvature direction vector
    rcu2 = cross(bu2,k2);

    dvdt2 = ((3.*b2.^2 - 1)/2). *c2.*dot(gc2, bu2).*bu2 - ((b2.^2 + 1)/2). *c2.*dot(gc2, rcu2).*rcu2;

    % dvdt2 = k2 for Runge-Kutta point 2
    % Calculate velocity and position at Runge-Kutta point 3

    v2 = v1 + dvdt2*dtb2;  % dvdt1 = k1 for Runge-Kutta
    rx2 = rx1+(dtb2*(v1(1,:)+v2(1,:))/2)'; ry2 = ry1+(dtb2*(v1(2,:)+v2(2,:))/2)'; rz2 = rz1+(dtb2*(v1(3,:)+v2(3,:))/2)';
    % Calculate conditions for point 2. Start by calculating the distance (Dr) between each body
    Drx = rx2' - rx2; Dry = ry2' - ry2; Drz = rz2' - rz2;
    Dr = (Drx.^2 + Dry.^2 + Drz.^2 + zlow).^0.5; % Temporary storage for distance^2 between bodies
    % Calculate the unit vector that defines the direction of separation. The unit vector magnitude is
    % equal to 1 except when the separation = 0, if so the magnitude = 0. Dru has a magnitude of 1 or zero,
    % therefore, sqrt(1) = 1. This eliminates the need to take the sqrt.
    Drxu = Drx ./ Dr; Dryu = Dry ./ Dr; Drzu = Drz ./ Dr;
    gc2 = (Drx.^2 + Dry.^2 + Drz.^2 + zlow); % Temporary storage for distance^2 between bodies
    Dr = gc2.^0.5;
    % Calculate the speed of light at the location of each body (see TMG equations 4-10 through 4-14).
    % The influence of the individual body is ignored when calculating the speed of light at its location.
    phi2 = (-tau) * sum(mb .* Dru ./ Dr);  % tau = 2*g/cs/cs as defined in the file that calls this script.
    c2 = c_inf * exp(phi2);
    % Calculate beta for each body
    b2 = ((v2(1,:).^2 + v2(2,:).^2 + v2(3,:).^2).^0.5)./c2;
    bu2 = (v2./c2)./b2;
    % Calculate the gradient of c
    gc2 = c2 .* tau .* mb .* Dru ./ gc2;  % Initial gc2 value equals distance^2 between bodies (see comments above)
    gc2 = [sum(gc2 .* Drxu)
           sum(gc2 .* Dryu)
           sum(gc2 .* Drzu)];
    % Calculate the unit rotation vector (see TMG Figure 13 and eq. 7-1)
    k2 = cross(gc2,bu2);
    k2 = k2 ./ ((k2(1,:).^2 + k2(2,:).^2 + k2(3,:).^2 ).^(1/2));
    % Calculate the radius of curvature direction vector
    rcu2 = cross(bu2,k2);

    dvdt3 = ((3.*b2.^2 - 1)/2). *c2.*dot(gc2, bu2).*bu2 - ((b2.^2 + 1)/2). *c2.*dot(gc2, rcu2).*rcu2;

    % dvdt3 = k3 for Runge-Kutta point, point 3
    % Calculate velocity and position at Runge-Kutta point 4

    v2 = v1 + dvdt3*dt;  % dvdt1 = k1 for Runge-Kutta
```

```

rx2 = rx1+(dt*(v1(1,:)+v2(1,:))/2)'; ry2 = ry1+(dt*(v1(2,:)+v2(2,:))/2)'; rz2 = rz1+(dt*(v1(3,:)+v2(3,:))/2)';
% Calculate conditions for point 2. Start by calculating the distance (Dr) between each body
Drx = rx2' - rx2; Dry = ry2' - ry2; Drz = rz2' - rz2;
Dr = (Drx.^2 + Dry.^2 + Drz.^2 + zlow).^0.5; % Temporary storage for distance^2 between bodies
% Calculate the unit vector that defines the direction of separation. The unit vector magnitude is
% equal to 1 except when the separation = 0, if so the magnitude = 0. Dru has a magnitude of 1 or zero,
% therefore, sqrt(1) = 1. This eliminates the need to take the sqrt.
Drxu = Drx ./ Dr; Dryu = Dry ./ Dr; Drzu = Drz ./ Dr;
gc2 = (Drx.^2 + Dry.^2 + Drz.^2 + zlow); % Temporary storage for distance^2 between bodies
Dr = gc2.^0.5;
% Calculate the speed of light at the location of each body (see TMG equations 4-10 through 4-14).
% The influence of the individual body is ignored when calculating the speed of light at its location.
phi2 = (-tau) * sum(mb .* Dru ./ Dr); % tau = 2*g/cs/cs as defined in the file that calls this script.
c2 = c_inf * exp(phi2);
% Calculate beta for each body
b2 = ((v2(1,:).^2 + v2(2,:).^2 + v2(3,:).^2).^0.5)./c2;
bu2 = (v2./c2)./b2;
% Calculate the gradient of c
gc2 = c2 .* tau .* mb .* Dru ./ gc2; % Initial gc2 value equals distance^2 between bodies (see comments above)
gc2 = [sum(gc2 .* Drxu)
       sum(gc2 .* Dryu)
       sum(gc2 .* Drzu)];
% Calculate the unit rotation vector (see TMG Figure 13 and eq. 7-1)
k2 = cross(gc2,bu2);
k2 = k2 ./ ((k2(1,:).^2 + k2(2,:).^2 + k2(3,:).^2 ).^(1/2));
% Calculate the radius of curvature direction vector
rcu2 = cross(bu2,k2);

dvdt4 = ((3.*b2.^2 - 1)/2).*c2.*dot(gc2, bu2).*bu2 - ((b2.^2 + 1)/2).*c2.*dot(gc2, rcu2).*rcu2;

% Calculate point 2

% dvdt4 = k4 for Runge-Kutta point 4
% Calculate velocity and position at point 2

v2 = v1 + dt * (dvdt1 + 2*dvdt2 + 2*dvdt3 + dvdt4) / 6;
rx2 = rx1+(dt*(v1(1,:)+v2(1,:))/2)'; ry2 = ry1+(dt*(v1(2,:)+v2(2,:))/2)'; rz2 = rz1+(dt*(v1(3,:)+v2(3,:))/2)';
% Calculate conditions for point 2. Start by calculating the distance (Dr) between each body
Drx = rx2' - rx2; Dry = ry2' - ry2; Drz = rz2' - rz2;
Dr = (Drx.^2 + Dry.^2 + Drz.^2 + zlow).^0.5; % Temporary storage for distance^2 between bodies
% Calculate the unit vector that defines the direction of separation. The unit vector magnitude is
% equal to 1 except when the separation = 0, if so the magnitude = 0. Dru has a magnitude of 1 or zero,
% therefore, sqrt(1) = 1. This eliminates the need to take the sqrt.
Drxu = Drx ./ Dr; Dryu = Dry ./ Dr; Drzu = Drz ./ Dr;
gc2 = (Drx.^2 + Dry.^2 + Drz.^2 + zlow); % Temporary storage for distance^2 between bodies
Dr = gc2.^0.5;
% Calculate the speed of light at the location of each body (see TMG equations 4-10 through 4-14).
% The influence of the individual body is ignored when calculating the speed of light at its location.
phi2 = (-tau) * sum(mb .* Dru ./ Dr); % tau = 2*g/cs/cs as defined in the file that calls this script.
c2 = c_inf * exp(phi2);
% Calculate beta for each body
b2 = ((v2(1,:).^2 + v2(2,:).^2 + v2(3,:).^2).^0.5)./c2;
bu2 = (v2./c2)./b2;
% Calculate the gradient of c
gc2 = c2 .* tau .* mb .* Dru ./ gc2; % Initial gc2 value equals distance^2 between bodies (see comments above)
gc2 = [sum(gc2 .* Drxu)
       sum(gc2 .* Dryu)
       sum(gc2 .* Drzu)];
% Calculate the unit rotation vector (see TMG Figure 13 and eq. 7-1)
k2 = cross(gc2,bu2);
k2 = k2 ./ ((k2(1,:).^2 + k2(2,:).^2 + k2(3,:).^2 ).^(1/2));
% Calculate the radius of curvature direction vector
rcu2 = cross(bu2,k2);

dvdt2 = ((3.*b2.^2 - 1)/2).*c2.*dot(gc2, bu2).*bu2 - ((b2.^2 + 1)/2).*c2.*dot(gc2, rcu2).*rcu2;

% calculate path time

t2 = t1 + dt;

% Find and save the perihelion time for the body of interest

dphi2 = phi2(b_pr)-phi1(b_pr); % This value is used in finding the perihelion time for the body of interest

if dphi1 <=0 && dphi2>=0 % The perihelion is between these values
    disp(['Orbit count = ' num2str(n_orbit) ' from 1st gravitational potential minimum for body ' num2str(b_pr,3)])
    n_orbit = n_orbit + 1;
    %tp(n_orbit) = (t1(b_pr)+t2(b_pr))/2 - dphi2 * (t2(b_pr)-t1(b_pr)) / ((dphi2-dphi1)+zlow)% at the perihelion

    zm = (dphi2-dphi1)/(t2(b_pr)-t1(b_pr));
    zb = dphi1 - zm*t1(b_pr);
    tp(n_orbit) = - zb / zm;

    zm = (bu2(:,b_pr)-bul(:,b_pr))/(t2(b_pr)-t1(b_pr));
    zb = bul(:,b_pr) - zm*t1(b_pr);

    vpu_n = zm*tp(n_orbit) + zb; % at the perihelion
    vpu_n = vpu_n ./ ((vpu_n(1,:).^2 + vpu_n(2,:).^2 + vpu_n(3,:).^2 ).^(1/2));

    if n_orbit==1
        vpu_1 = vpu_n;
        zdot = 1;
    else
        zdot = dot(vpu_1',vpu_n');
        % Determine if the precession is positive or negative
        zcross = cross(vpu_1',vpu_n);
        zcross = zcross ./ ((zcross(1).^2 + zcross(2).^2 + zcross(3).^2 ).^(1/2));
        prsine = zcross + k2(:,b_pr)';
        prsine = prsine(1).^2 + prsine(2).^2 + prsine(3).^2 ).^(1/2);
        if prsine >1.0

```

```

        zprecession(n_orbit) = acos(zdot);
    else
        zprecession(n_orbit) = -acos(zdot);
    end
end
if plot_option_1 == 1
else
    plot(tp/(365.25*24*3600*100), zprecession * (3600*180/pi)); % precession per century
end

end
% Set values for next time step
b1 = b2;
bu1 = bu2;
c1 = c2;
dvdt1 = dvdt2;
phi1 = phi2;
dphi1 = dphi2;
gc1 = gc2;
rx1 = rx2;
ry1 = ry2;
rz1 = rz2;
t1 = t2;
v1 = v2;
end

```

TMG_nb_path_dtheta_integration.m

```
% TMG_nb_path_dtheta_integration.m
%
% Integrates the path with a limited max rotation step
%-----
% Created by Morris G. Anderson
%   https://www.researchgate.net/profile/Morris-Anderson
%
% Units, nomenclature, and documentation links provided in the accompanying script file:
%   Background_and_Nomenclature.m
%
% Theory provided in the paper: N-body Gravity Simulation by Curvature of Wave Propagation
%-----
% tau = 2*g/cs/cs; % Defined in the input file

dt_check = 0;

for arc_count = 1:nac % The following logic keeps all bodies in time with each other
    [val,idx] = min(rca./b1./c1); % find the body with fastest rotation rate
    dt = rca(idx)*d_theta_max/(b1(idx)*c1(idx));
    if dt < dt_min
        dt = dt_min; % dt_min is set by the user
        % disp(['dt limited to dt_min, solution time =', num2str(t1(1)/365.25/24/3600), ' years'])
    end
    if (t1(idx)+dt)>tmax
        dt = tmax - t1(idx);
        dt_check = 1;
    end
    ai = ((dt +t1(idx)) - t1) .* c1 .* b1 ./ rca;
    % Rotate unit vectors with Rodrigues' formula to calculate position and motion at point 2
    bu2 = bu1 .* cos(ai) + cross(ka,bu1) .* sin(ai) + ka .* dot(ka, bu1) .* (1 - cos(ai));
    bu2 = bu2 ./ ((bu2(1,:).^2 + bu2(2,:).^2 + bu2(3,:).^2 ).^(1/2));
    % A portion of the Rodrigues formula is replaced with the following to reduce numerical error
    vtu = bu1 + bu2;
    vtu = vtu ./ ((vtu(1,:).^2 + vtu(2,:).^2 + vtu(3,:).^2 ).^(1/2));
    vt = 2.0 * rca .* sin(ai/2);
    rx2 = rx1 + vtu(1,:)' .* vt';
    ry2 = ry1 + vtu(2,:)' .* vt';
    rz2 = rz1 + vtu(3,:)' .* vt';
    % Calculate conditions for point 2. Start by calculating the distance between each body
    Drx = rx2' - rx2;
    Dry = ry2' - ry2;
    Drz = rz2' - rz2;
    gc2 = (Drx.^2 + Dry.^2 + Drz.^2 + zlow); % Temporary storage for distance^2 between bodies
    Dr = gc2.^0.5;
    % Calculate the unit vector that defines the direction of separation. The unit vector magnitude is
    % equal to 1 except when the separation = 0, if so the magnitude = 0. Dru has a magnitude of 1 or zero,
    % therefore, sqrt(1) = 1. This eliminates the need to take the sqrt.
    Drxu = Drx ./ Dr;
    Dryu = Dry ./ Dr;
    Drzu = Drz ./ Dr;
    Dru = (Drxu.^2 + Dryu.^2 + Drzu.^2);
    % Calculate the speed of light at the location of each body (see TMG equations 4-10 through 4-14).
    % The influence of the individual body is ignored when calculating the speed of light at its location.
    phi2 = (-tau) * sum(mb .* Dru ./ Dr);
    c2 = c_inf * exp(phi2);
    % Calculate the gradient of c
    gc2 = c2 .* tau .* mb .* Dru ./ gc2; % Initial gc2 value equals distance^2 between bodies (see comments above)
    gc2 = [sum(gc2 .* Drxu)
           sum(gc2 .* Dryu)
           sum(gc2 .* Drzu)];
    % Calculate b2
    drby2c_ave = (rca .* ai) ./ (c1 + c2);
    dotbugc2 = dot(bu2, gc2);
    dotbugc_ave = (dotbugc1 + dotbugc2)/2;

    dbeta = ((b1.^2 - 1)./b1) .* (drby2c_ave .* dotbugc_ave);
    % Repeat a few times to get a solution with sufficient accuracy.
    b_ave = (2*b1 + dbeta)/2;
    dbeta = ((b_ave.^2 - 1)./b_ave) .* (drby2c_ave .* dotbugc_ave);
    b_ave = (2*b1 + dbeta)/2;
    dbeta = ((b_ave.^2 - 1)./b_ave) .* (drby2c_ave .* dotbugc_ave);
    b2 = b1 + dbeta;
    % Calculate the unit rotation vector (see TMG Figure 13 and eq. 7-1), which
    % is equal to the cross-product of the gradient of c and the motion unit vector.
    % Then calculate the rotation vector magnitude.
    k2 = cross(gc2,bu2);
    k2 = k2 ./ ((k2(1,:).^2 + k2(2,:).^2 + k2(3,:).^2 ).^(1/2));
    rcu2 = cross(bu2,k2);
    rcu2 = rcu2 ./ ((rcu2(1,:).^2 + rcu2(2,:).^2 + rcu2(3,:).^2 ).^(1/2));
    rc2 = (2*c2./(1+1./(b2.^2)))./(dot(gc2, rcu2)+zlow);
    % calculate the path time
    t2 = t1 + ai .* rca ./ ((b1 .* c1 + b2 .* c2) / 2);
    % use linear extrapolation to estimate the next rca, for this step rca = (rc2 + rc1) ./ 2
    % for the next step rc1 = rc2, therefore
    rca = rc2 .* (rc2 + rc1) ./ (2 * rc1);
    ka = k2 + (k2 - k1) / 2;
    ka = ka ./ ( (ka(1,:).^2 + ka(2,:).^2 + ka(3,:).^2 ).^(1/2) );

    b1 = b2;
    bu1 = bu2;
    c1 = c2;
    gc1 = gc2;
    k1 = k2;
    phi1 = phi2;
    dphi1 = dphi2;
    rc1 = rc2;
    rx1 = rx2;
end
```

```
    ry1 = ry2;  
    rz1 = rz2;  
    t1 = t2;  
  
    dotbugc1 = dotbugc2;  
    if dt_check==1  
        disp('Solution runtime completed')  
        break  
    end  
end  
end
```

Example_1_The_Solar_System.M

```
% Example_1_the_solar_system.m
%
% This example simulates the solar system to calculate the total precession of individual planets.
%-----
% Created by Morris G. Anderson
%   https://www.researchgate.net/profile/Morris-Anderson
%
% Units, nomenclature, and documentation links provided in the accompanying script file:
%   Background_and_Nomenclature.m
%
% Theory provided in the paper: N-body Gravity Simulation by Curvature of Wave Propagation
%-----
clc      % Clear the command window.
clear

simo_option = 0; % Set=0 if using MatLab, Set simo_option=1 if using the SIMO MATLAB programming app
record=0;       % initialize value

if simo_option==0
    close ('all'); % Close all figures
    clearvars;
    simo_option = 0;
    record=0;    % set record=1 to record a video of the simulation
    plot_AR = 16/9; % plot window aspect ratio
    plot_pix = 1000;
    myplot=figure('Position',[100 100 plot_AR*plot_pix plot_pix]);hold on; % figure window
    grid on
    ax = gca;
    ax.LineWidth = 2;
    ax.GridLineWidth = 2;
    ax.Color = ('k');
    lw = 1.5; % plot line width
    ms = 5; % plot symbol size
    %plot_gAR = 55.3/29.4; % plot grid aspect ratio
    plot_gAR = 55.3/32.25; % plot grid aspect ratio
    if record==1
        vid = VideoWriter('Movie_Ex_1_our_solar_system.mp4','MPEG-4');
        vid.FrameRate = 30;
        open(vid);
    end
else
    lw = 1.0; % plot line width
    ms = 1; % plot symbol size
    plot_gAR = 49/29.1; % plot grid aspect ratio for ipad M2
end

g = 0.66743e-10; % Newton's universal gravitational constant. Units = m^3/(kg-s)
cs = 299792458.0; % The standard speed of light, see Time Matter and Gravity (TMG) section 2.4.1
tau = 2*g/cs/cs;
zlow = 1e-49; % zlow is a limiting value to prevent dividing by zero

disp(['Newton's G = ' num2str(g,6) ' m^3/(kg-s), Standard speed of light = ' num2str(cs,10) ' m/s'])
%-----
% n-body simulation inputs for paper Figure 4 left side -----
plot_option_1 = 1; % If = 1, plot orbits. If = 0, plot precession.
plot_option_2 = 1.0*1.1; % If = 0, no limit. If =>1, plot limit = plot_option_2 * max distance from body 1

nbc = 0; % The orbit is plotted relative to this body. If nbc =0, plot in the calculation reference frame.
zd = 1; % If = 1, the rotation is positive (right-hand rule). If = -1, the rotation is negative.

nb = 15 % The number of governing bodies. 2 = The Sun and Mercury. 3 = The Sun, Mercury, and Venus. etc.
b_pr = 5 % The body for calculating orbital precession and period
dt = 1000 % Time step, seconds. This is approximate, dt is recalculated below to match tt
days = 20.0*365.25 % Total simulation time.
tt = days * 24*60*60 % The total path time, seconds
tt_nps = tt*1.0; % The plotted solution time from current to the past.
nps = 300 % The number of points saved for plotting. Set = 10000 to duplicate Figure 4
zvg = 0.0 * 230000.0; % Group velocity of the solar system relative to the galactic center.
%-----
% n-body simulation inputs for paper Figures 7 & 8 -----
% plot_option_1 = 0; % If = 1, plot orbits. If = 0, plot precession.
% plot_option_2 = 1.1; % If = 0, no limit. If =>1, plot limit = plot_option_2 * max distance from body 1
%
% nbc = 0; % The orbit is plotted relative to this body. If nbc =0, plot in the calculation reference frame.
% zd = 1; % If = 1, the rotation is positive (right-hand rule). If = -1, the rotation is negative.
%
% nb = 15 % The number of governing bodies. 2 = The Sun and Mercury. 3 = The Sun, Mercury, and Venus. etc.
% b_pr = 2 % The body for calculating orbital precession and period
% dt = 320 % Time step, seconds. This is approximate, dt is recalculated below to match tt
% days = 200*365.25 % Total simulation time.
% tt = days * 24*60*60 % The total path time, seconds
% tt_nps = tt*1; % The plotted solution time from current to the past.
% nps = 30 % The number of points saved for plotting
% zvg = 0.0 * 230000.0; % Group velocity of the solar system relative to the galactic center.
%-----
%
% nbc = round(tt_nps/dt/nps) % The number of arc segments or time steps per point that are saved for plotting
% np = round((tt / dt) / nbc) % The total number of points that could be plotted
% ni = np * nbc % The total number of time steps
% dt = tt / ni % The time step
%-----
% For plotting orbits and other parameters
rx = zeros(nps+1,nb); % rx = x position
ry = zeros(nps+1,nb); % ry = y position
rz = zeros(nps+1,nb); % rz = z position
rc = zeros(nps+1,nb); % rc = radius of curvature
b = zeros(nps+1,nb); % b = v/c
```

```

c = zeros(nps+1,nb); % c = the speed of light at the location of each body. The influence of the
% individual body is ignored when calculating the speed of light at its location.
txyz = zeros(nps+1,nb); % time, seconds, corresponding to the saved positions.
%-----
% Read in initial starting conditions for each body
zSS_2000_01_01_PS_downloaded_on_2023_02_23 % This script initializes with 15 solar system bodies that include moon mass
% You may add a scalar group velocity. For example, You can add the group velocity of the solar system
% relative to the galactic center to the individual velocities. However, the precession calculation does is not
% as accurate - see paper for details.
vg(1) = zvg * cos(60*pi/180); % The simulation x group velocity.
vg(2) = 0.0; % The simulation y group velocity.
vg(3) = zvg * sin(60*pi/180); % The simulation z group velocity.

vx = vx + vg(1);
vy = vy + vg(2);
vz = vz + vg(3);

disp(['group velocity: vgx = ' num2str(vg(1),6) ' vgy = ' num2str(vg(2),6) ' vgz = ' num2str(vg(3),6) ' m/s' ])

% Calculate c away from the earth in the solar system. Also calculate c away from the solar system.

re = 6371008.7714; % Earth's IUGG mean radius (R1) see https://en.wikipedia.org/wiki/Earth_radius
reoa = 149.598e9; % Earth's Semi-major orbital axis, see https://nssdc.gsfc.nasa.gov/planetary/factsheet/earthfact.html

% c_inf = cs*exp((2*sum(mb(4))*g/cs/cs)/re); % Calculate c away from the earth with TMG eq 4-11.
%
% disp(['c = ' num2str(c_inf,15) ' m/s, away from Earth at ' num2str(re,8) ' meters from the Sun or the Earth's average orbital radius'])
%
% c_inf = c_inf*exp((2*(sum(mb)-mb(4))*g/cs/cs)/reoa); % Calculate c away from the solar system TMG eq 4-11.

% The initial conditions for this study are taken from the NASA JPL
% Horizons website. the Horizons system is based on Barycentric Dynamical
% Time (TDB). This crucial information is clearly explained in the
% following comment that is included with vectors downloaded from the
% Horizons system.
%
% Barycentric Dynamical Time ("TDB" or Teph) output was requested.
% This continuous coordinate time is equivalent to the relativistic
% proper time of a clock at rest in a reference frame co-moving with
% the solar system barycenter but outside the system's gravity well.
% It is the independent variable in the solar system relativistic
% equations of motion.
%
% TDB runs at a uniform rate of one SI second per second and is
% independent of irregularities in Earth's rotation.
%
% According to this information, the Horizons system output for time,
% length, velocity, and gravitational mass (GM) are based on a convention
% where the time standard location is set at infinity or outside the
% gravitational well of the solar system.

c_inf = 299792458.0

disp(['c = ' num2str(c_inf,15) ' m/s, away from the Solar Systems influence consistent to NASA JPL Horizons TDB' ])

% Reset scalars to the number of simulated bodies

mb = mb(1,1:nb);
rx = rx(:,1:nb);
ry = ry(:,1:nb);
rz = rz(:,1:nb);

for n = 1:(nps+1)
    rx(n,1:nb) = rx(1,1:nb);
    ry(n,1:nb) = ry(1,1:nb);
    rz(n,1:nb) = rz(1,1:nb);
end

vx = vx(1,1:nb);
vy = vy(1,1:nb);
vz = vz(1,1:nb);

vx = zd.*vx; vy = zd.*vy; vz = zd.*vz;

disp(['Number of bodies simulated = ' num2str(nb,5) ])

tic

TMG_nb_initialization % This script initializes the solution
tstart(1,1:nb) = 0; % Starting time, in seconds
t1(1,1:nb) = 0; % Starting time, in seconds

plot_limit = plot_option_2*max(Dr(:,1)); % Limits for the orbital plot
zprecession = 0; % Initialize the precession for the body of interest
tp = 0; % Initialize the perihelion time for the body of interest
tp_count = 1;

for n = 1:np % Integrate the orbits

    TMG_nb_path_dt_integration; % This script calculates the arc path

    if plot_option_1 == 1
        % Save values for plotting

        rx(1:nps,1:nb) = rx(2:nps+1,1:nb); rx(nps+1,1:nb) = rx1';
        ry(1:nps,1:nb) = ry(2:nps+1,1:nb); ry(nps+1,1:nb) = ry1';
        rz(1:nps,1:nb) = rz(2:nps+1,1:nb); rz(nps+1,1:nb) = rz1';
        txyz(1:nps,1:nb) = txyz(2:nps+1,1:nb); txyz(nps+1,1:nb) = t1';

        if nbc == 0

```

```

plot(rx(nps,:), ry(nps,:), 'o', 'MarkerSize', ms, 'MarkerFaceColor', 'w')
if plot_option_2 == 0
else
    xlim([-plot_gAR*plot_limit, plot_gAR*plot_limit])
    ylim([-plot_limit, plot_limit])
end
hold('on')
plot(rx(2:nps,:), ry(2:nps,:), 'linewidth', lw)
hold('off')
else
    plot((rx(nps,1:nb)- rx(nps,nbc)), (ry(nps,1:nb)- ry(nps,nbc)), 'o', 'MarkerSize', ms, 'MarkerFaceColor', 'w') % plot the saved
    hold('on')
    if plot_option_2 == 0
    else
        xlim([-plot_gAR*plot_limit, plot_gAR*plot_limit])
        ylim([-plot_limit, plot_limit])
    end
    plot((rx(:,1:nb)-rx(:,nbc)), (ry(:,1:nb)-ry(:,nbc)), 'linewidth', lw) % Plot relative with line connecting to the origin
    hold('off')
end

if simo_option ==0
    set(gca, 'Color', 'k')
    %axis equal
    drawnow
    if record==1
        A=getframe(myplot);
        writeVideo(vid,A);
    end
end
else
    plot(tp/(365.25*24*3600*100), zprecession * (3600*180/pi), 'color', 'b'); % precession per Julian century
    if simo_option ==0
        set(gca, 'Color', 'w')
        drawnow
    end
end
end

exportgraphics(ax, ['Plot_Ex_1_our_solar_system.png'])

if record==1
    close(vid); % close video file
end

Example_1_post_process_results; % Post-process data
% if simo_option ==0
%     save(['Out_Ex_1_SS_' int2str(nb) '_' _nb_ ' int2str(zd*round(t1(1)/365/24/3600)) '_years_dt_' int2str(round(dt)) '_seconds']);
% end
toc

```

Example_1_The_Solar_System_Dvdt.M

```
% Example_1_the_solar_system_dvdt.m
%
% This script file applies a 4th order Runge-Kutta solver method
%
% This example simulates the solar system to calculate the total precession of individual planets.
%-----
% Created by Morris G. Anderson
%   https://www.researchgate.net/profile/Morris-Anderson
%
% Units, nomenclature, and documentation links provided in the accompanying script file:
%   Background_and_Nomenclature.m
%
% Theory provided in the paper: N-body Gravity Simulation by Curvature of Wave Propagation
%-----
clc      % Clear the command window.
clear

simo_option = 0; % Set=0 if using MatLab, Set simo_option=1 if using the SIMO MATLAB programming app
record=0;       % initialize value

if simo_option==0
    close ('all'); % Close all figures
    clearvars;
    simo_option = 0;
    record=0;    % set record=1 to record a video of the simulation
    plot_AR = 16/9; % plot window aspect ratio
    plot_pix = 1000;
    myplot=figure('Position',[100 100 plot_AR*plot_pix plot_pix]);hold on; % figure window
    grid on
    ax = gca;
    ax.LineWidth = 2;
    ax.GridLineWidth = 2;
    ax.Color = ('k');
    lw = 1.5; % plot line width
    ms = 5; % plot symbol size
    %plot_gAR = 55.3/29.4; % plot grid aspect ratio
    plot_gAR = 55.3/32.25; % plot grid aspect ratio
    if record==1
        vid = VideoWriter('Movie_Ex_1_our_solar_system.mp4','MPEG-4');
        vid.FrameRate = 30;
        open(vid);
    end
else
    lw = 1.0; % plot line width
    ms = 1; % plot symbol size
    plot_gAR = 49/29.1; % plot grid aspect ratio for iPad M2
end

g = 0.66743e-10; % Newton's universal gravitational constant. Units = m^3/(kg-s)
cs = 299792458.0; % The standard speed of light, see Time Matter and Gravity (TMG) section 2.4.1
tau = 2*g/cs/cs;
zlow = 1e-49; % zlow is a limiting value to prevent dividing by zero

disp(['Newton's G = ' num2str(g,6) ' m^3/(kg-s), Standard speed of light = ' num2str(cs,10) ' m/s'])
%-----
% n-body simulation inputs for paper Figure 4 left side -----
plot_option_1 = 1; % If = 1, plot orbits. If = 0, plot precession.
plot_option_2 = 1.0*1.1; % If = 0, no limit. If =>1, plot limit = plot_option_2 * max distance from body 1

nbc = 0; % The orbit is plotted relative to this body. If nbc =0, plot in the calculation reference frame.
zd = 1; % If = 1, the rotation is positive (right-hand rule). If = -1, the rotation is negative.

nb = 15 % The number of governing bodies. 2 = The Sun and Mercury. 3 = The Sun, Mercury, and Venus. etc.
b_pr = 5 % The body for calculating orbital precession and period
dt = 1000 % Time step, seconds. This is approximate, dt is recalculated below to match tt
days = 20.0*365.25 % Total simulation time.
tt = days * 24*60*60 % The total path time, seconds
tt_nps = tt*1.0; % The plotted solution time from current to the past.
nps = 300 % The number of points saved for plotting. Set = 10000 to duplicate Figure 4
zvg = 0.0 * 230000.0; % Group velocity of the solar system relative to the galactic center.
%-----
% n-body simulation inputs for paper Figures 7 & 8 -----
% plot_option_1 = 0; % If = 1, plot orbits. If = 0, plot precession.
% plot_option_2 = 1.1; % If = 0, no limit. If =>1, plot limit = plot_option_2 * max distance from body 1
%
% nbc = 0; % The orbit is plotted relative to this body. If nbc =0, plot in the calculation reference frame.
% zd = 1; % If = 1, the rotation is positive (right-hand rule). If = -1, the rotation is negative.
%
% nb = 15 % The number of governing bodies. 2 = The Sun and Mercury. 3 = The Sun, Mercury, and Venus. etc.
% b_pr = 2 % The body for calculating orbital precession and period
% dt = 320 % Time step, seconds. This is approximate, dt is recalculated below to match tt
% days = 200*365.25 % Total simulation time.
% tt = days * 24*60*60 % The total path time, seconds
% tt_nps = tt*1; % The plotted solution time from current to the past.
% nps = 30 % The number of points saved for plotting
% zvg = 0.0 * 230000.0; % Group velocity of the solar system relative to the galactic center.
%-----
%
% nbc = round(tt_nps/dt/nps) % The number of arc segments or time steps per point that are saved for plotting
% np = round((tt / dt) / nbc) % The total number of points that could be plotted
% ni = np * nbc % The total number of time steps
% dt = tt / ni % The time step
%-----
% For plotting orbits and other parameters
rx = zeros(nps+1,nb); % rx = x position
ry = zeros(nps+1,nb); % ry = y position
rz = zeros(nps+1,nb); % rz = z position
```

```

rc = zeros(nps+1,nb); % rc = radius of curvature
b = zeros(nps+1,nb); % b = v/c
c = zeros(nps+1,nb); % c = the speed of light at the location of each body. The influence of the
% individual body is ignored when calculating the speed of light at its location.
txyz = zeros(nps+1,nb); % time, seconds, corresponding to the saved positions.
%-----
% Read in initial starting conditions for each body
zSS_2000_01_01_PS_downloaded_on_2023_02_23 % This script initializes with 15 solar system bodies that include moon mass
% You may add a scalar group velocity. For example, You can add the group velocity of the solar system
% relative to the galactic center to the individual velocities. However, the precession calculation does is not
% as accurate - see paper for details.
vg(1) = zvg * cos(60*pi/180); % The simulation x group velocity.
vg(2) = 0.0; % The simulation y group velocity.
vg(3) = zvg * sin(60*pi/180); % The simulation z group velocity.

vx = vx + vg(1);
vy = vy + vg(2);
vz = vz + vg(3);

disp(['group velocity: vgx = ' num2str(vg(1),6) ' vgy = ' num2str(vg(2),6) ' vgz = ' num2str(vg(3),6) ' m/s '])

% Calculate c away from the earth in the solar system. Also calculate c away from the solar system.

re = 6371008.7714; % Earth's IUGG mean radius (R1) see https://en.wikipedia.org/wiki/Earth_radius
reoa = 149.598e9; % Earth's Semi-major orbital axis, see https://nssdc.gsfc.nasa.gov/planetary/factsheet/earthfact.html

% c_inf = cs*exp((2*sum(mb(4))*g/cs/cs)/re); % Calculate c away from the earth with TMG eq 4-11.
%
% disp(['c = ' num2str(c_inf,15) ' m/s, away from Earth at ' num2str(re,8) ' meters from the Sun or the Earth's average orbital radius'])
%
% c_inf = c_inf*exp((2*(sum(mb)-mb(4))*g/cs/cs)/reoa); % Calculate c away from the solar system TMG eq 4-11.

% The initial conditions for this study are taken from the NASA JPL
% Horizons website. the Horizons system is based on Barycentric Dynamical
% Time (TDB). This crucial information is clearly explained in the
% following comment that is included with vectors downloaded from the
% Horizons system.
%
% Barycentric Dynamical Time ("TDB" or T_eph) output was requested.
% This continuous coordinate time is equivalent to the relativistic
% proper time of a clock at rest in a reference frame co-moving with
% the solar system barycenter but outside the system's gravity well.
% It is the independent variable in the solar system relativistic
% equations of motion.
%
% TDB runs at a uniform rate of one SI second per second and is
% independent of irregularities in Earth's rotation.
%
% According to this information, the Horizons system output for time,
% length, velocity, and gravitational mass (GM) are based on a convention
% where the time standard location is set at infinity or outside the
% gravitational well of the solar system.

c_inf = 299792458.0

disp(['c = ' num2str(c_inf,15) ' m/s, away from the Solar Systems influence consistent to NASA JPL Horizons TDB' ])

% Reset scalars to the number of simulated bodies

mb = mb(1,1:nb);
rx = rx(:,1:nb);
ry = ry(:,1:nb);
rz = rz(:,1:nb);

for n = 1:(nps+1)
    rx(n,1:nb) = rx(1,1:nb);
    ry(n,1:nb) = ry(1,1:nb);
    rz(n,1:nb) = rz(1,1:nb);
end

vx = vx(1,1:nb);
vy = vy(1,1:nb);
vz = vz(1,1:nb);

vx = zd.*vx; vy = zd.*vy; vz = zd.*vz;

disp(['Number of bodies simulated = ' num2str(nb,5) ])

tic

TMG_nb_initialization % This script initializes the solution
tstart(1,1:nb) = 0; % Starting time, in seconds
t1(1,1:nb) = 0; % Starting time, in seconds

plot_limit = plot_option_2*max(Dr(:,1)); % Limits for the orbital plot
zprecession = 0; % Initialize the precession for the body of interest
tp = 0; % Initialize the perihelion time for the body of interest
tp_count = 1;

for n = 1:np % Integrate the orbits

    TMG_nb_path_dt_integration_dvdt; % This script calculates the arc path

    if plot_option_1 == 1
        % Save values for plotting

        rx(1:nps,1:nb) = rx(2:nps+1,1:nb); rx(nps+1,1:nb) = rx1';
        ry(1:nps,1:nb) = ry(2:nps+1,1:nb); ry(nps+1,1:nb) = ry1';
        rz(1:nps,1:nb) = rz(2:nps+1,1:nb); rz(nps+1,1:nb) = rz1';
        txyz(1:nps,1:nb) = txyz(2:nps+1,1:nb); txyz(nps+1,1:nb) = t1';
    end
end

```

```

if nbc == 0
    plot(rx(nps,:), ry(nps,:), 'o', 'MarkerSize', ms, 'MarkerFaceColor', 'w')
    if plot_option_2 == 0
        else
            xlim([-plot_gAR*plot_limit, plot_gAR*plot_limit])
            ylim([-plot_limit, plot_limit])
        end
        hold('on')
        plot(rx(2:nps,:), ry(2:nps,:), 'linewidth', lw)
        hold('off')
    else
        plot((rx(nps,1:nb)- rx(nps,nbc)), (ry(nps,1:nb)- ry(nps,nbc)), 'o', 'MarkerSize', ms, 'MarkerFaceColor', 'w') % plot the saved
        hold('on')
        if plot_option_2 == 0
            else
                xlim([-plot_gAR*plot_limit, plot_gAR*plot_limit])
                ylim([-plot_limit, plot_limit])
            end
            plot((rx(:,1:nb)-rx(:,nbc)), (ry(:,1:nb)-ry(:,nbc)), 'linewidth', lw) % Plot relative with line connecting to the origin
            hold('off')
        end
    end

    if simo_option == 0
        set(gca, 'Color', 'k')
        %axis equal
        drawnow
        if record == 1
            A = getframe(myplot);
            writeVideo(vid, A);
        end
    end
end
else
    plot(tp/(365.25*24*3600*100), zprecession * (3600*180/pi), 'color', 'b'); % precession per Julian century
    if simo_option == 0
        set(gca, 'Color', 'w')
        drawnow
    end
end
end
end

exportgraphics(ax, ['Plot_Ex_1_our_solar_system.png'])

if record == 1
    close(vid); % close video file
end

Example_1_post_process_results; % Post-process data
if simo_option == 0
    save(['Out_Ex_1_SS_' int2str(nb) '_' _nb_ ' int2str(zd*round(t1(1)/365.25/24/3600)) '_' _years_dt_ ' int2str(round(dt)) '_' _seconds']);
end
toc

```

Example_2_Shapiro_Time_delay.m

```
% Example_2_Shapiro_Time_delay.m
%
% This example simulates the Shapiro time delay for light passing near the Sun between Earth and another planet
%-----
% Created by Morris G. Anderson
% https://www.researchgate.net/profile/Morris-Anderson
%
% Units, nomenclature, and documentation links provided in the accompanying script file:
% Background_and_Nomenclature.m
%
% Theory provided in the paper: N-body Gravity Simulation by Curvature of Wave Propagation
%-----
clc % Clear the command window.
clear

simo_option = 0; % Set=0 if using MatLab, Set simo_option=1 if using the SIMO MATLAB programming app
record=0; % initialize value

if simo_option==0
    close ('all'); % Close all figures
    clearvars;
    simo_option = 0;
    record=0; % set record=1 to record a video of the simulation
    plot_AR = 16/9; % plot window aspect ratio
    plot_pix = 1000;
    myplot=figure('Position',[100 100 plot_AR*plot_pix plot_pix]);hold on; % figure window
    grid on
    ax = gca;
    ax.LineWidth = 2;
    ax.GridLineWidth = 2;
    ax.Color = ('k');
    lw = 2.0; % plot line width
    ms = 10; % plot symbol size
    %plot_gAR = 55.3/29.4; % plot grid aspect ratio
    plot_gAR = 55.3/32.25; % plot grid aspect ratio
    if record==1
        vid = VideoWriter('Movie_Ex_2_Shapiro_Time_delay.mp4','MPEG-4');
        vid.FrameRate = 30;
        open(vid);
    end
else
    lw = 1.0; % plot line width
    ms = 1; % plot symbol size
    plot_gAR = 49/29.1; % plot grid aspect ratio for ipad M2
end

g = 0.66743e-10; % Newton's universal gravitational constant. Units = m^3/(kg-s)
cs = 299792458.0; % The standard speed of light, see Time Matter and Gravity (TMG) section 2.4.1
tau = 2*g/cs/cs;
zlow = 1e-49; % zlow is a limiting value to prevent dividing by zero

disp(['Newton's G = ' num2str(g,6) ' m^3/(kg-s), Standard speed of light = ' num2str(cs,10) ' m/s'])

% n-body simulation inputs-----
plot_option_1 = 1; % If = 1, plot orbits.
plot_option_2 = 0; % If = 1, plot each path and connect the end of each path line to the origin. If = 0, plot each path.
plot_option_3 = 1.2; % If = 0, no limit. If =>1, plot limit = plot_option_3 * max distance from body 1

nb = 6; % The number of governing bodies. 2 = The Sun and Mercury. 7 = The Sun, Mercury, ..., and Jupiter.
b_pr = 2; % The body for tracking the number of orbits and orbital precession
%-----
%%% *** Use this input for comparison with the 1970 Shapiro time delay experiment.
% zd = 1 % If = 1, the rotation is positive (right-hand rule). If = -1, the rotation is negative.
% days = 25 % Set this to the desired number of days (0 to 300) from the Superior Conjunction
% nps = 100 % Points saved for plotting. This is sufficient for this problem. You can use more if desired
% tt = (days - zd*(0.5 + 71.488/360))*60*60*24 % The total path time, seconds.
% dt = tt / (20*nps) % This is a reasonable time step for this problem.
% tt_nps = tt; % The orbital path length that is stored for plotting
% shp1 = 4; % body from which light is emitted
% shp2 = 3; % body to which light is sent
% disp(['Shapiro time delay calculated ' num2str(days*zd,3) ' days from the January 25, 1970 Superior Conjunction with Venus'])
%-----
%%% *** Use this for the 1967 Shapiro time delay experiment documented in 1968 in Physical Review Letters Volume 20, number 22.
% zd = -1 % If = 1, the rotation is positive (right-hand rule). If = -1, the rotation is negative.
% for this example zd should be = -1 to roll the planets back to 1967 from the 1970 data
% days = 21.1825 % Set this to the desired number of days(0 to 30) after April 20, 1967 or August 10, 1967
% days = 23.0 % Set this to the desired number of days(0 to 30) after April 20, 1967 or August 10, 1967
% nps = 300 % Points saved for plotting. This is sufficient for this problem. You can use more if desired
% tt = (899 - days + zd*(0.5 + 71.488/360))*60*60*24 % The total path time, seconds. Use this for the August data.
% tt = (1011 - days + zd*(0.5 + 71.488/360))*60*60*24 % The total path time, seconds. Use this for the April data.
% dt = tt / (200*nps) % (paper results calculated with dt = tt / (2000*nps). This is a reasonable time step for this problem.
% tt_nps = 1.0*tt; % The orbital path length that is stored for plotting
% shp1 = 4; % body from which light is emitted
% shp2 = 2; % body to which light is sent
disp(['Shapiro time delay calculated ' num2str(days*zd,3) ' days from the January 25, 1970 Superior Conjunction with Mercury'])
%-----
nac = round(tt_nps/dt/nps) % The number of arc segments or time steps per point that are saved for plotting
np = round((tt / dt) / nac) % The total number of points that could be plotted
ni = np * nac; % The total number of time steps
dt = tt / ni; % The time step
disp(['Initial rotation dt = ' num2str(dt,10) ' seconds'])
%-----
% For plotting orbits and other parameters
rx = zeros(nps+1,nb); % rx = x position
ry = zeros(nps+1,nb); % ry = y position
rz = zeros(nps+1,nb); % rz = z position
rc = zeros(nps,nb); % rc = radius of curvature
```

```

b = zeros(nps,nb); % b = v/c
c = zeros(nps,nb); % c = the speed of light at the location of each body. The influence of the
% individual body is ignored when calculating the speed of light at its location.
%-----
% *** Read in initial starting conditions for each body
zSS_1970_01_25_downloaded_on_2023_06_26 % Use this for the 1968 and 1970 Shapiro time delay experiment
% You may add a scalar group velocity. For example, You can add the group velocity of the solar system
% relative to the galactic center to the individual velocities.
% Accuracy? - see paper.

zvg_scalar = 0.0;
zvg = zvg_scalar*230000.0; % Group velocity of the solar system relative to the galactic center.
vg(1) = zvg * sin(60*pi/180); % The simulation x group velocity.
vg(2) = 0.0; % The simulation y group velocity.
vg(3) = zvg * cos(60*pi/180); % The simulation z group velocity.

vx = vx + vg(1);
vy = vy + vg(2);
vz = vz + vg(3);

disp(['group velocity: vgx = ' num2str(vg(1),6) ' vgy = ' num2str(vg(2),6) ' vgz = ' num2str(vg(3),6) ' m/s' ])

% Calculate c away from the earth in the solar system. Also calculate c away from the solar system.

re = 6371008.7714; %Earth's IUGG mean radius (R1) see https://en.wikipedia.org/wiki/Earth_radius
reoa = 149.598e9; %Earth's Semi-major orbital axis, see https://nssdc.gsfc.nasa.gov/planetary/factsheet/earthfact.html

% c_inf = cs*exp((2*sum(mb(4))*g/cs/cs)/re); % Calculate c away from the earth with TMG eq 4-11.
%
% disp(['c = ' num2str(c_inf,15) ' m/s, away from Earth at ' num2str(re,8) ' meters from the Sun or the Earth's average orbital radius'])
%
% c_inf = c_inf*exp((2*(sum(mb)-mb(4))*g/cs/cs)/reoa); % Calculate c away from the solar system TMG eq 4-11.

% The initial conditions for this study are taken from the NASA JPL
% Horizons website. the Horizons system is based on Barycentric Dynamical
% Time (TDB). This crucial information is clearly explained in the
% following comment that is included with vectors downloaded from the
% Horizons system.
%
% Barycentric Dynamical Time ("TDB" or T_eph) output was requested.
% This continuous coordinate time is equivalent to the relativistic
% proper time of a clock at rest in a reference frame co-moving with
% the solar system barycenter but outside the system's gravity well.
% It is the independent variable in the solar system relativistic
% equations of motion.
%
% TDB runs at a uniform rate of one SI second per second and is
% independent of irregularities in Earth's rotation.
%
% According to this information, the Horizons system output for time,
% length, velocity, and gravitational mass (GM) are based on a convention
% where the time standard location is set at infinity or outside the
% gravitational well of the solar system.

c_inf = 299792458.0

disp(['c = ' num2str(c_inf,15) ' m/s, away from the Solar Systems influence consistent to NASA JPL Horizons TDB' ])

% Reset scalars to the number of simulated bodies

mb = mb(1,1:nb)';

rx = rx(1,1:nb);
ry = ry(1,1:nb);
rz = rz(1,1:nb);

for n = 1:(nps)
    rx(n,1:nb) = rx(1,1:nb);
    ry(n,1:nb) = ry(1,1:nb);
    rz(n,1:nb) = rz(1,1:nb);
end

if plot_option_2 == 1
    rx(nps+1,1:nb) = 0;
    ry(nps+1,1:nb) = 0;
    rz(nps+1,1:nb) = 0;
end

vx = vx(1,1:nb);
vy = vy(1,1:nb);
vz = vz(1,1:nb);

vx = zd.*vx; vy = zd.*vy; vz = zd.*vz;

disp(['Number of bodies simulated = ' num2str(nb,5) ])

tic

TMG_nb_initialization % This script initializes the solution

tstart(1,1:nb) = 0; % Starting time, in seconds
t1(1,1:nb) = 0; % Starting time, in seconds

plot_limit = plot_option_3*max(Dr(:,1)); % Limits for the orbital plot
zprecession = 0; % Initialize the precession for the body of interest
tp = 0; % Initialize the perihelion time for the body of interest
tp_count = 1;

for n = 1:nps % Integrate the orbits

```

```

TMG_nb_path_dt_integration; % This script calculates the arc path

if plot_option_1 == 1
    % Save values for plotting
    rx(1:(nps-1),1:nb) = rx(2:nps,1:nb);
    rx(nps,1:nb) = rx1';
    ry(1:(nps-1),1:nb) = ry(2:nps,1:nb);
    ry(nps,1:nb) = ry1';
    rz(1:(nps-1),1:nb) = rz(2:nps,1:nb);
    rz(nps,1:nb) = rz1';

    plot(rx(nps,:), ry(nps,:), 'o', 'MarkerSize', ms, 'MarkerFaceColor', 'w')
    if plot_option_3 == 0
    else
        xlim([-plot_gAR*plot_limit, plot_gAR*plot_limit])
        ylim([-plot_limit, plot_limit])
    end
    hold('on')
    plot(rx, ry, 'linewidth', lw)
    hold('off')
    if simo_option == 0
        set(gca, 'Color', 'k')
        drawnow
        if record == 1
            A = getframe(myplot);
            writeVideo(vid, A);
        end
    end
end
end

pl_save_x = rx(1:nps, 1:nb); % for plotting with the following light ray solution
pl_save_y = ry(1:nps, 1:nb); % for plotting with the following light ray solution

%-----
% Calculate the Shapiro Time Delay from body shp1 to body shp2
%-----
bx(1,:) = b1.*bul(1,:); % Initialize from previous solution
by(1,:) = b1.*bul(2,:);
bz(1,:) = b1.*bul(3,:);

rx = rx1';
ry = ry1';
rz = rz1';

nb = nb+1; % add in a photon
mb(nb) = 0; % photon rest mass = 0

% calculate photon starting point and direction vector

disp(['Calculating light passage between body ' num2str(shp1,3) ' and ' num2str(shp2,3)])

shp1rx = rx1(shp1);
shp1ry = ry1(shp1);
shp1rz = rz1(shp1);

shp2rx = rx1(shp2);
shp2ry = ry1(shp2);
shp2rz = rz1(shp2);
shpDr = sqrt((shp2rx-shp1rx)^2 + (shp2ry-shp1ry)^2 + (shp2rz-shp1rz)^2);
shpdt = 2*(shpDr - rb(shp1) - rb(shp2)) / (c1(shp1) + c1(shp2));

shp2rx = rx1(shp2) + shpdt*c1(shp2)*b1(shp2).*bul(1,shp2); % repeat a fee times to improve estimate.
shp2ry = ry1(shp2) + shpdt*c1(shp2)*b1(shp2).*bul(2,shp2);
shp2rz = rz1(shp2) + shpdt*c1(shp2)*b1(shp2).*bul(3,shp2);
shpDr = sqrt((shp2rx-shp1rx)^2 + (shp2ry-shp1ry)^2 + (shp2rz-shp1rz)^2);
shpdt = 2*(shpDr - rb(shp1) - rb(shp2)) / (c1(shp1) + c1(shp2));

shp2rx = rx1(shp2) + shpdt*c1(shp2)*b1(shp2).*bul(1,shp2);
shp2ry = ry1(shp2) + shpdt*c1(shp2)*b1(shp2).*bul(2,shp2);
shp2rz = rz1(shp2) + shpdt*c1(shp2)*b1(shp2).*bul(3,shp2);
shpDr = sqrt((shp2rx-shp1rx)^2 + (shp2ry-shp1ry)^2 + (shp2rz-shp1rz)^2);
shpdt = 2*(shpDr - rb(shp1) - rb(shp2)) / (c1(shp1) + c1(shp2));

shp_Drxu = (shp2rx-shp1rx)/shpDr;
shp_Dryu = (shp2ry-shp1ry)/shpDr;
shp_Drzu = (shp2rz-shp1rz)/shpDr;

rx(:,nb) = rx(:,shp1) + rb(shp1)*shp_Drxu;
ry(:,nb) = ry(:,shp1) + rb(shp1)*shp_Dryu;
rz(:,nb) = rz(:,shp1) + rb(shp1)*shp_Drzu;
bx(1,nb) = shp_Drxu;
by(1,nb) = shp_Dryu;
bz(1,nb) = shp_Drzu;

bu_shp_start = [shp_Drxu
    shp_Dryu
    shp_Drzu]; % photon starting point and direction vector

b1(nb) = 1;
beta_option(nb) = 1;

c(1,nb) = 0; % Initialize storage space for this value

%-----
dt = 0.15%0.25; % Time step, seconds. This is approximate, dt is recalculated below to match tt
%-----
tt = shpdt; % Approximate time required for light to pass from shp1 to shp2
tt_nps = tt; % Plot entire photon path
ni = tt/dt % The total number of time steps

```

```

nac    = 1 % (paper results calculated with nac = 1). The number of arc segments or time steps per point that are saved for plotting
np     = round(ni / nac); % The total number of points that could be plotted
nps    = np;
ni     = np * nac;        % For this example, nps must = np to capture the beginning and end
dt     = tt / ni         % The time step
%-----
nps = np; % Reset to compensate for roundoff error. For this example, nps must = np to capture the beginning and end

TMG_nb_initialization % This script initializes the solution

t1(nb) = mean(t1(1,nb-1));
tstart(1,1:nb) = t1; %Starting time, in seconds
t1(1,1:nb) = t1; %Starting time, in seconds

rx= rx(1,:) + zeros (nps+1,nb); %initialize all plotting points to starting positions
ry= ry(1,:) + zeros (nps+1,nb); %initialize all plotting points to starting positions
rz= rz(1,:) + zeros (nps+1,nb); %initialize all plotting points to starting positions

rbrs(1:nps+1)= Dr(1,nb)/rb(1); % Calculate distance in solar radii

zprecession = 0; % Initialize the precession for the body of interest
tp = 0; % Initialize the perihelion time for the body of interest
tp_count =1;

for n = 1:np % Integrate the orbits

    TMG_nb_path_dt_integration; % This script calculates the arc path

    % Save values for plotting
    rx(1:nps,1:nb) = rx(2:nps+1,1:nb);
    rx(nps+1,1:nb) = rx1';
    ry(1:nps,1:nb) = ry(2:nps+1,1:nb);
    ry(nps+1,1:nb) = ry1';
    rz(1:nps,1:nb) = rz(2:nps+1,1:nb);
    rz(nps+1,1:nb) = rz1';

    rbrs(1:nps) = rbrs(2:nps+1);
    rbrs(nps+1) = Dr(1,nb)/rb(1);
end

plot(rx(nps,:), ry(nps,:), 'o', 'MarkerSize',ms, 'MarkerFaceColor', 'w')
if plot_option_3 == 0
else
    xlim([-plot_gAR*plot_limit,plot_gAR*plot_limit])
    ylim([-plot_limit,plot_limit])
end
hold('on')
plot(rx, ry, 'linewidth', 2, 'lineStyle', ':')
plot(pl_save_x, pl_save_y, 'linewidth', lw) % plot previous solution
hold('off')
xlabel('x, meter')
ylabel('y, meter')
if simo_option ==0
    set(gca, 'Color', 'k')
    drawnow
    if record==1
        A=getframe(myplot);
        writeVideo(vid,A);
        close(vid); % close video file
    end
end
end

%disp(['The following value accuracy is best if nps = np'])
disp(['days = ' num2str( days,6) ])
disp(['Closest Approach to the Sun in terms solar photosphere radii = ' num2str( min(rbrs),6) ])

bu_shp_end = bu1(:,nb);
zdot = dot(bu_shp_start,bu_shp_end);
ab = acos(zdot)*180/pi;
disp(['Light Deflection = ' num2str( ab*3600,6) ' arcseconds' ])

delay = 2*((t1(nb)-tstart(nb)) - sqrt( (rx(nps+1,nb)-rx(1,nb))^2 + (ry(nps+1,nb)-ry(1,nb))^2 + (rz(nps+1,nb)-rz(1,nb))^2 ) / c_inf);

disp(['Shapiro time delay = ' num2str(delay,6) ' seconds' ])

disp('Calculation')

if simo_option ==0
    saveas(ax, 'Plot_Ex_2_Shapiro_Time_delay.png')
end
toc

```

Example_2_Shapiro_Time_delay_dvdt.m

```
% Example_2_Shapiro_Time_delay_dvdt.m
%
% This script file applies a 4th order Runge-Kutta solver method
%
% This example simulates the Shapiro time delay for light passing near the Sun between Earth and another planet
%-----
% Created by Morris G. Anderson
% https://www.researchgate.net/profile/Morris-Anderson
%
% Units, nomenclature, and documentation links provided in the accompanying script file:
% Background_and_Nomenclature.m
%
% Theory provided in the paper: N-body Gravity Simulation by Curvature of Wave Propagation
%-----
clc % Clear the command window.
clear

simo_option = 0; % Set=0 if using MatLab, Set simo_option=1 if using the SIMO MATLAB programming app
record=0; % initialize value

if simo_option==0
    close ('all'); % Close all figures
    clearvars;
    simo_option = 0;
    record=0; % set record=1 to record a video of the simulation
    plot_AR = 16/9; % plot window aspect ratio
    plot_pix = 1000;
    myplot=figure('Position',[100 100 plot_AR*plot_pix plot_pix]);hold on; % figure window
    grid on
    ax = gca;
    ax.LineWidth = 2;
    ax.GridLineWidth = 2;
    ax.Color = ('k');
    lw = 2.0; % plot line width
    ms = 10; % plot symbol size
    %plot_gAR = 55.3/29.4; % plot grid aspect ratio
    plot_gAR = 55.3/32.25; % plot grid aspect ratio
    if record==1
        vid = VideoWriter('Movie_Ex_2_Shapiro_Time_delay.mp4','MPEG-4');
        vid.FrameRate = 30;
        open(vid);
    end
else
    lw = 1.0; % plot line width
    ms = 1; % plot symbol size
    plot_gAR = 49/29.1; % plot grid aspect ratio for iPad M2
end

g = 0.66743e-10; % Newton's universal gravitational constant. Units = m^3/(kg-s)
cs = 299792458.0; % The standard speed of light, see Time Matter and Gravity (TMG) section 2.4.1
tau = 2*g/cs/cs;
zlow = 1e-49; % zlow is a limiting value to prevent dividing by zero

disp(['Newton's G = ' num2str(g,6) ' m^3/(kg-s), Standard speed of light = ' num2str(cs,10) ' m/s'])

% n-body simulation inputs-----
plot_option_1 = 1; % If = 1, plot orbits.
plot_option_2 = 0; % If = 1, plot each path and connect the end of each path line to the origin. If = 0, plot each path.
plot_option_3 = 1.2; % If = 0, no limit. If =>1, plot limit = plot_option_3 * max distance from body 1

nb = 6; % The number of governing bodies. 2 = The Sun and Mercury. 7 = The Sun, Mercury, ..., and Jupiter.
b_pr = 2; % The body for tracking the number of orbits and orbital precession
%-----
%%% *** Use this input for comparison with the 1970 Shapiro time delay experiment.
% zd = 1 % If = 1, the rotation is positive (right-hand rule). If = -1, the rotation is negative.
% days = 25 % Set this to the desired number of days (0 to 300) from the Superior Conjunction
% nps = 100 % Points saved for plotting. This is sufficient for this problem. You can use more if desired
% tt = (days - zd*(0.5 + 71.488/360))*60*60*24 % The total path time, seconds.
% dt = tt / (20*nps) % This is a reasonable time step for this problem.
% tt_nps = tt; % The orbital path length that is stored for plotting
% shp1 = 4; % body from which light is emitted
% shp2 = 3; % body to which light is sent
% disp(['Shapiro time delay calculated ' num2str(days*zd,3) ' days from the January 25, 1970 Superior Conjunction with Venus'])
%-----
%%% *** Use this for the 1967 Shapiro time delay experiment documented in 1968 in Physical Review Letters Volume 20, number 22.
zd = -1 % If = 1, the rotation is positive (right-hand rule). If = -1, the rotation is negative.
% for this example zd should be = -1 to roll the planets back to 1967 from the 1970 data
% days = 21.1825 % Set this to the desired number of days(0 to 30) after April 20, 1967 or August 10, 1967
% days = 23.0 % Set this to the desired number of days(0 to 30) after April 20, 1967 or August 10, 1967
% nps = 300 % Points saved for plotting. This is sufficient for this problem. You can use more if desired
% tt = (899 - days + zd*(0.5 + 71.488/360))*60*60*24 % The total path time, seconds. Use this for the August data.
% tt = (1011 - days + zd*(0.5 + 71.488/360))*60*60*24 % The total path time, seconds. Use this for the April data.
% dt = tt / (200*nps) % (paper results calculated with dt = tt / (2000*nps). This is a reasonable time step for this problem.
% tt_nps = 1.0*tt; % The orbital path length that is stored for plotting
% shp1 = 4; % body from which light is emitted
% shp2 = 2; % body to which light is sent
% disp(['Shapiro time delay calculated ' num2str(days*zd,3) ' days from the January 25, 1970 Superior Conjunction with Mercury'])
%-----
nac = round(tt_nps/dt/nps) % The number of arc segments or time steps per point that are saved for plotting
np = round((tt / dt) / nac) % The total number of points that could be plotted
ni = np * nac; % The total number of time steps
dt = tt / ni; % The time step
disp(['Initial rotation dt = ' num2str(dt,10) ' seconds'])
%-----
% For plotting orbits and other parameters
rx = zeros(nps+1,nb); % rx = x position
ry = zeros(nps+1,nb); % ry = y position
```

```

rz = zeros(nps+1,nb); % rz = z position
rc = zeros(nps,nb); % rc = radius of curvature
b = zeros(nps,nb); % b = v/c
c = zeros(nps,nb); % c = the speed of light at the location of each body. The influence of the
% individual body is ignored when calculating the speed of light at its location.
%-----
% *** Read in initial starting conditions for each body
zSS 1970_01_25 downloaded on 2023_06_26 % Use this for the 1968 and 1970 Shapiro time delay experiment
% You may add a scalar group velocity. For example, You can add the group velocity of the solar system
% relative to the galactic center to the individual velocities.
% Accuracy? - see paper.

zvg_scalar = 0.0;
zvg = zvg_scalar*230000.0; % Group velocity of the solar system relative to the galactic center.
vg(1) = zvg * sin(60*pi/180); % The simulation x group velocity.
vg(2) = 0.0; % The simulation y group velocity.
vg(3) = zvg * cos(60*pi/180); % The simulation z group velocity.

vx = vx + vg(1);
vy = vy + vg(2);
vz = vz + vg(3);

disp(['group velocity: vgx = ' num2str(vg(1),6) ' vgy = ' num2str(vg(2),6) ' vgz = ' num2str(vg(3),6) ' m/s' ])

% Calculate c away from the earth in the solar system. Also calculate c away from the solar system.

re = 6371008.7714; %Earth's IUGG mean radius (R1) see https://en.wikipedia.org/wiki/Earth_radius
reoa = 149.598e9; %Earth's Semi-major orbital axis, see https://nssdc.gsfc.nasa.gov/planetary/factsheet/earthfact.html

% c_inf = cs*exp((2*sum(mb(4))*g/cs/cs)/re); % Calculate c away from the earth with TMG eq 4-11.
%
% disp(['c = ' num2str(c_inf,15) ' m/s, away from Earth at ' num2str(re,8) ' meters from the Sun or the Earth's average orbital radius'])
%
% c_inf = c_inf*exp((2*(sum(mb)-mb(4))*g/cs/cs)/reoa); % Calculate c away from the solar system TMG eq 4-11.

% The initial conditions for this study are taken from the NASA JPL
% Horizons website. the Horizons system is based on Barycentric Dynamical
% Time (TDB). This crucial information is clearly explained in the
% following comment that is included with vectors downloaded from the
% Horizons system.
%
% Barycentric Dynamical Time ("TDB" or T_eph) output was requested.
% This continuous coordinate time is equivalent to the relativistic
% proper time of a clock at rest in a reference frame co-moving with
% the solar system barycenter but outside the system's gravity well.
% It is the independent variable in the solar system relativistic
% equations of motion.
%
% TDB runs at a uniform rate of one SI second per second and is
% independent of irregularities in Earth's rotation.
%
% According to this information, the Horizons system output for time,
% length, velocity, and gravitational mass (GM) are based on a convention
% where the time standard location is set at infinity or outside the
% gravitational well of the solar system.

c_inf = 299792458.0

disp(['c = ' num2str(c_inf,15) ' m/s, away from the Solar Systems influence consistent to NASA JPL Horizons TDB' ])

% Reset scalars to the number of simulated bodies

mb = mb(1,1:nb)';

rx = rx(1,1:nb);
ry = ry(1,1:nb);
rz = rz(1,1:nb);

for n = 1:(nps)
    rx(n,1:nb) = rx(1,1:nb);
    ry(n,1:nb) = ry(1,1:nb);
    rz(n,1:nb) = rz(1,1:nb);
end

if plot_option_2 == 1
    rx(nps+1,1:nb) = 0;
    ry(nps+1,1:nb) = 0;
    rz(nps+1,1:nb) = 0;
end

vx = vx(1,1:nb);
vy = vy(1,1:nb);
vz = vz(1,1:nb);

vx = zd.*vx; vy = zd.*vy; vz = zd.*vz;

disp(['Number of bodies simulated = ' num2str(nb,5) ])

tic

TMG_nb_initialization % This script initializes the solution

tstart(1,1:nb) = 0; % Starting time, in seconds
t1(1,1:nb) = 0; % Starting time, in seconds

plot_limit = plot_option_3*max(Dr(:,1)); % Limits for the orbital plot
zprecession = 0; % Initialize the precession for the body of interest
tp = 0; % Initialize the perihelion time for the body of interest
tp_count = 1;

```

```

for n = 1:np      % Integrate the orbits

    TMG_nb_path_dt_integration_dvdt; % This script calculates the arc path

    if plot_option_1 == 1
        % Save values for plotting
        rx(1:(nps-1),1:nb) = rx(2:nps,1:nb);
        rx(nps,1:nb) = rx1';
        ry(1:(nps-1),1:nb) = ry(2:nps,1:nb);
        ry(nps,1:nb) = ry1';
        rz(1:(nps-1),1:nb) = rz(2:nps,1:nb);
        rz(nps,1:nb) = rz1';

        plot(rx(nps,:), ry(nps,:), 'o', 'MarkerSize', ms, 'MarkerFaceColor', 'w')
        if plot_option_3 == 0
            else
                xlim([-plot_gAR*plot_limit, plot_gAR*plot_limit])
                ylim([-plot_limit, plot_limit])
            end
            hold('on')
            plot(rx, ry, 'linewidth', lw)
            hold('off')
            if simo_option == 0
                set(gca, 'Color', 'k')
                drawnow
                if record==1
                    A=getframe(myplot);
                    writeVideo(vid,A);
                end
            end
        end
    end
end

pl_save_x=rx(1:nps,1:nb); % for plotting with the following light ray solution
pl_save_y=ry(1:nps,1:nb); % for plotting with the following light ray solution

%-----
% Calculate the Shapiro Time Delay from body shp1 to body shp2
%-----
bx(1,:) = b1.*bul(1,:); % Initialize from previous solution
by(1,:) = b1.*bul(2,:);
bz(1,:) = b1.*bul(3,:);

rx = rx1';
ry = ry1';
rz = rz1';

nb = nb+1; % add in a photon
mb(nb) = 0; % photon rest mass = 0

% calculate photon starting point and direction vector

disp(['Calculating light passage between body ' num2str( shp1,3) ' and ' num2str( shp2,3)])

shp1rx = rx1(shp1);
shp1ry = ry1(shp1);
shp1rz = rz1(shp1);

shp2rx = rx1(shp2);
shp2ry = ry1(shp2);
shp2rz = rz1(shp2);
shpDr = sqrt( (shp2rx-shp1rx)^2 + (shp2ry-shp1ry)^2 + (shp2rz-shp1rz)^2 );
shpdt = 2*(shpDr - rb(shp1) - rb(shp2) ) / (c1(shp1) + c1(shp2));

shp2rx = rx1(shp2) + shpdt*c1(shp2)*b1(shp2).*bul(1,shp2); % repeat a fee times to improve estimate.
shp2ry = ry1(shp2) + shpdt*c1(shp2)*b1(shp2).*bul(2,shp2);
shp2rz = rz1(shp2) + shpdt*c1(shp2)*b1(shp2).*bul(3,shp2);
shpDr = sqrt( (shp2rx-shp1rx)^2 + (shp2ry-shp1ry)^2 + (shp2rz-shp1rz)^2 );
shpdt = 2*(shpDr - rb(shp1) - rb(shp2) ) / (c1(shp1) + c1(shp2));

shp2rx = rx1(shp2) + shpdt*c1(shp2)*b1(shp2).*bul(1,shp2);
shp2ry = ry1(shp2) + shpdt*c1(shp2)*b1(shp2).*bul(2,shp2);
shp2rz = rz1(shp2) + shpdt*c1(shp2)*b1(shp2).*bul(3,shp2);
shpDr = sqrt( (shp2rx-shp1rx)^2 + (shp2ry-shp1ry)^2 + (shp2rz-shp1rz)^2 );
shpdt = 2*(shpDr - rb(shp1) - rb(shp2) ) / (c1(shp1) + c1(shp2));

shp_Drxu = (shp2rx-shp1rx)/shpDr;
shp_Dryu = (shp2ry-shp1ry)/shpDr;
shp_Drzu = (shp2rz-shp1rz)/shpDr;

rx(:,nb) = rx(:,shp1) +rb(shp1)*shp_Drxu;
ry(:,nb) = ry(:,shp1) +rb(shp1)*shp_Dryu;
rz(:,nb) = rz(:,shp1) +rb(shp1)*shp_Drzu;
bx(1,nb) = shp_Drxu;
by(1,nb) = shp_Dryu;
bz(1,nb) = shp_Drzu;

bu_shp_start = [shp_Drxu
                shp_Dryu
                shp_Drzu]; % photon starting point and direction vector

b1(nb) = 1;
beta_option(nb) = 1;

c(1,nb) = 0; % Initialize storage space for this value

%-----
dt = 0.025%0.25; % Time step, seconds. This is approximate, dt is recalculated below to match tt
%-----
tt = shpdt; % Approximate time required for light to pass from shp1 to shp2

```

```

tt_nps = tt; % Plot entire photon path
ni = tt/dt % The total number of time steps
nac = 1 % (paper results calculated with nac = 1). The number of arc segments or time steps per point that are saved for plotting
np = round(ni / nac); % The total number of points that could be plotted
nps = np;
ni = np * nac; % For this example, nps must = np to capture the beginning and end
dt = tt / ni % The time step
%-----
nps = np; % Reset to compensate for roundoff error. For this example, nps must = np to capture the beginning and end

TMG_nb_initialization % This script initializes the solution

clear t1
clear tstart
t1 = 0;
tstart=0; %Starting time, in seconds

rx= rx(1,:) + zeros (nps+1,nb); %initialize all plotting points to starting positions
ry= ry(1,:) + zeros (nps+1,nb); %initialize all plotting points to starting positions
rz= rz(1,:) + zeros (nps+1,nb); %initialize all plotting points to starting positions

rbrs(1:nps+1)= Dr(1,nb)/rb(1); % Calculate distance in solar radii

zprecession = 0; % Initialize the precession for the body of interest
tp = 0; % Initialize the perihelion time for the body of interest
tp_count =1;

for n = 1:np % Integrate the orbits

    TMG_nb_path_dt_integration_dvdt; % This script calculates the arc path

    % Save values for plotting
    rx(1:nps,1:nb) = rx(2:nps+1,1:nb);
    rx(nps+1,1:nb) = rx1';
    ry(1:nps,1:nb) = ry(2:nps+1,1:nb);
    ry(nps+1,1:nb) = ry1';
    rz(1:nps,1:nb) = rz(2:nps+1,1:nb);
    rz(nps+1,1:nb) = rz1';

    rbrs(1:nps) = rbrs(2:nps+1);
    rbrs(nps+1) = Dr(1,nb)/rb(1);
end

plot(rx(nps,:), ry(nps,:), 'o', 'MarkerSize',ms, 'MarkerFaceColor', 'w')
if plot_option_3 == 0
else
    xlim([-plot_gAR*plot_limit,plot_gAR*plot_limit])
    ylim([-plot_limit,plot_limit])
end
hold('on')
plot(rx, ry, 'linewidth', 2, 'lineStyle', ':')
plot(pl_save_x, pl_save_y, 'linewidth', lw) % plot previous solution
hold('off')
xlabel('x, meter')
ylabel('y, meter')
if simo_option ==0
    set(gca, 'Color', 'k')
    drawnow
    if record==1
        A=getframe(myplot);
        writeVideo(vid,A);
        close(vid); % close video file
    end
end

disp(['The following value accuracy is best if nps = np'])
disp(['days = ' num2str( days,6) ])
disp(['Closest Approach to the Sun in terms solar photosphere radii = ' num2str( min(rbrs),6) ])

bu_shp_end = bu1(:,nb);
zdot = dot(bu_shp_start,bu_shp_end);
ab = acos(zdot)*180/pi;
disp(['Light Deflection = ' num2str( ab*3600,6) ' arcseconds' ])

delay = 2*((t1-tstart) - sqrt( (rx(nps+1,nb)-rx(1,nb))^2 + (ry(nps+1,nb)-ry(1,nb))^2 + (rz(nps+1,nb)-rz(1,nb))^2 ) / c_inf);

disp(['Shapiro time delay = ' num2str(delay,6) ' seconds' ])

disp('Calculation')

if simo_option ==0
    saveas(ax, 'Plot_EX_2_Shapiro_Time_delay.png')
end
toc

```

Example_3_Apparent_black_hole_shadow.m

```
% Example_3_Apparent_black_hole_shadow.m
%
% This example simulates the motion of light passing an apparent black hole.
%-----
% Created by Morris G. Anderson
% https://www.researchgate.net/profile/Morris-Anderson
%
% Units, nomenclature, and documentation links provided in the accompanying script file:
% Background_and_Nomenclature.m
%
% Theory provided in the paper: N-body Gravity Simulation by Curvature of Wave Propagation
%-----
clc % Clear the command window.
clear

simo_option = 0; % Set = 0 if using MatLab, Set simo_option = 1 if using the SIMO MATLAB programming app

record=0; % initialize value

if simo_option ==0
    close ('all'); % Close all figures
    clearvars;
    simo_option = 0;
    solar=char(9737);
    record=0; % set record=1 to record a video of the simulation
    lw = 2.0; % plot line width
    ms = 7; % plot symbol size
    plot_gAR = 55.3/29.4; % plot grid aspect ratio for mac
    plot_gAR = 55.3/32.25; % plot grid aspect ratio
else
    lw = 1.0; % plot line width
    ms = 1; % plot symbol size
    plot_gAR = 65.6/37.3; % plot grid aspect ratio for ipad
end

zlow = 1e-49; % zlow is a limiting value to prevent dividing by zero

if record==1
    vid = VideoWriter('Movie_Ex_3_Apparent_black_hole_shadow.mp4','MPEG-4');
    vid.FrameRate = 30;
    open(vid);
end

if simo_option ==0
    plot_AR = 16/9; % plot window aspect ratio
    plot_pix = 1000;
    myplot=figure('Position',[100 100 plot_AR*plot_pix plot_pix]);hold on; % figure window
    grid on
    ax = gca;
    ax.LineWidth = 2;
    ax.GridLineWidth = 2;
    ax.Color = ('k');
end

g = 0.66743e-10; % Newton's universal gravitational constant. Units = m^3/(kg-s)
cs = 299792458.0; % The standard speed of light, see Time Matter and Gravity (TMG) section 2.4.1
tau = 2*g/cs/cs;
light_year = 365 * 24 * 3600 * cs; % meters, set this as the max initial semi-major axis

disp(['Newton's G = ' num2str(g,6) ' m^3/(kg-s), Standard speed of light = ' num2str(cs,10) ' m/s'])
%-----
% n-body simulation inputs -----
%
% The following input values are recommended for the SIMO app.
%-----
% nb = 26 % use 25 for Simo. This is The number of governing bodies and simulated photos.
% % At least two of these bodies must have rest mass.
% np = 600 % The total number of points that could be plotted
% nac = 50 % The number of arc segments or time steps per point that are saved for plotting
% nps = np % The number of points saved for plotting
% d_theta_max = 0.25 % This is the Max rotation per step, degree.
% tmax = 4.0*365.25*24*3600 % Maximum simulation time, seconds
%-----
%
% The following input values are recommended for MATLAB.
%-----
% nb = 51 % This is The number of governing bodies and simulated photos.
% % At least two of these bodies must have rest mass.
% np = 2000 % The total number of points that could be plotted
% nac = 100 % The number of arc segments or time steps per point that are saved for plotting
% nps = np % The number of points saved for plotting
% d_theta_max = 0.05 % This is the Max rotation per step, degree.
% tmax = 4.0*365.25*24*3600 % Maximum simulation time, seconds
%-----
dt_min = 0 % Minimum dt allowed for simulation. This can override d_theta_max
%-----
d_theta_max = d_theta_max*pi/180; % Max rotation per step, radians.
% For plotting orbits and other parameters
rx = zeros(nps+1,nb); % rx = x position
ry = zeros(nps+1,nb); % ry = y position
rz = zeros(nps+1,nb); % rz = z position
b = zeros(nps,nb); % b = v/c
%-----
```

```

beta_option(1:nb) = 1; % Initial motion input as v/c

% Define m, This is the mass of the governing body in kilograms
m_Sun = 1.32712e20/g; % from https://nssdc.gsfc.nasa.gov/planetary/factsheet/sunfact.html

disp('This example illustrates a light wavefront passing the central mass of M87*')

m = 6.5e9; % M87* mass in multiples of our Sun - from Information source 8

disp(['M87* galactic center mass = ' num2str((m),3) ' Suns' ])

mb(1) = m*m_Sun; % central mass in multiples of our Sun's mass, kg
mb(2) = 0; % photon orbiting the central mass
mb(3) = m_Sun; % 1st sun orbiting the central mass
mb(4) = m_Sun; % 2nd sun orbiting the central mass
mb(5:nb) = 0.0; % light wave approaching a central mass
rs = mb(1) * tau; % Schwartzchild radius
r_shadow = rs*exp(1);

% define beta for each body
bx(1,1) = 1e-9; % central body
bx(1,5:nb) = -1.0; % photons
by(1,1:nb) = 0.0;
by(1,2) = 1.0; % photon orbiting a central mass
by(1,3) = 0.4; % sun orbiting the central mass
by(1,4) = 0.35; % sun orbiting the central mass
bz(1,1:nb) = 0.0;

b1 = sqrt(bx.^2 + by.^2 + bz.^2);

% Calculate the radius for the circular orbit of body 2
z_rb2 = (1+1/(b1(2)^2))*mb(1)*g/cs^2 % Radius at witch body 2 will orbit body 1
phi1 = (-tau) * mb(1) ./ z_rb2;
c1 = cs * exp(phi1);
dt_plot = (2*pi*z_rb2/c1)/(2*pi/d_theta_max) % time between frames saved for plotting

xrs = 52; %scalar for setting light wavefront starting point.

rx(1:nps+1,2:nb) = xrs * rs + rx(1:nps+1,2:nb); % initial position of body nb(1)
rx(:,2) = z_rb2; % Set body 2 in position to orbit body 1
rx(:,3) = 5 * z_rb2; % Set body 3 through nb in position to orbit body 1
rx(:,4) = (1+1/(b1(4)^2))*mb(1)*g/cs^2; % Radius at witch body 4 will orbit body 1

y_scalar = 6; % sets the initial y values in terms of r_shadow
for zi=1:nps+1
    ry(zi,5:nb)=-y_scalar*r_shadow:((y_scalar*2*r_shadow)/(nb-5)):y_scalar*r_shadow;
end

ry(:,2) = 1e-50*ry(:,5);
ry(:,3) = 1e-50*ry(:,5);
ry(:,4) = 1e-50*ry(:,5);

mb = mb'; % Transpose matrix for initialization

c_inf = cs; % Set speed of light away from group to cs

disp(['c = ' num2str(c_inf,15) ' m/s, away from the simulation influence' ])

disp(['Number of bodies simulated = ' num2str(nb,5) ])

tic

TMG_nb_initialization % This script initializes the solution
tstart(1,1:nb) = 0; % Starting time, in seconds
t1(1,1:nb) = 0; % Starting time, in seconds

plot_limit = xrs*rs/light_year;

xmin = -plot_limit;
xmax = -xmin;
ymin = -plot_limit/plot_gAR;
ymax = -ymin;

shadow_limit_x = [xmin xmin
    xmax xmax];

shadow_limit_y = [-r_shadow/light_year r_shadow/light_year
    -r_shadow/light_year r_shadow/light_year];

hold('off')
zprecession = 0; % Initialize the precession for the body of interest
tp = 0; % Initialize the perihelion time for the body of interest
tp_count = 1;
ndt_plot = 1;
n = 0;

while mean(t1) < tmax % Intagrate the path
    n=n+1;

    TMG_nb_path_dtheta_integration % Integrate the orbits

    % Save values for plotting
    rx(1:(nps-1),1:nb) = rx(2:nps,1:nb); rx(nps,1:nb) = rx1';
    ry(1:(nps-1),1:nb) = ry(2:nps,1:nb); ry(nps,1:nb) = ry1';
    wavefront_x(1:nb-4,1) = rx1(5:nb);
    wavefront_y(1:nb-4,1) = ry1(5:nb);

    plot(shadow_limit_x, shadow_limit_y, 'lineStyle','--', 'linewidth', lw, 'color',[1,1,1])
    xlabel('x, light year')
    ylabel('y, light year')

```

```

xlim([xmin,xmax])
ylim([ymin,ymax])

hold('on')
plot(rx(nps,1)./light_year, ry(nps,1)./light_year,'o','MarkerSize',ms,'color','r','MarkerFaceColor','r') % central mass
plot(rx(nps,2)./light_year, ry(nps,2)./light_year,'o','MarkerSize',ms,'color','y','MarkerFaceColor','y') % light orbiting a central
mass
plot((rx(1:nps,2)./light_year), (ry(1:nps,2)./light_year),'color','y','linewidth', lw) % light orbiting central mass
plot(rx(nps,3)./light_year, ry(nps,3)./light_year,'o','MarkerSize',ms,'color',[0,1,1], 'MarkerFaceColor',[0,1,1]) % star orbiting a
central mass
plot((rx(1:nps,3)./light_year), (ry(1:nps,3)./light_year),'color',[0,1,1],'linewidth', 2) % star orbiting central mass
plot(rx(nps,4)./light_year, ry(nps,4)./light_year,'o','MarkerSize',ms,'color',[1,0.5,0], 'MarkerFaceColor',[1,0.5,0]) % star orbiting a
central mass
plot((rx(1:nps,4)./light_year), (ry(1:nps,4)./light_year),'color',[1,0.5,0],'linewidth', 2) % star orbiting central mass

plot(wavefront_x(1:nb-4,1)./light_year, wavefront_y(1:nb-4,1)./light_year,'color',[0,1,0],'linewidth', 2) % wavefront
plot((rx(1:nps,5:nb)./light_year), (ry(1:nps,5:nb)./light_year),'color','y','linewidth', lw/2, 'lineStyle',':') % wavefront light path
% za = annotation('doublearrow', [0.8 0.8], [0.35 0.5] );
% za.Color = [1 1 1];
hold('off')

if simo_option ==0
    set(gca,'Color','k')
    ztitle = ['n-body = ' num2str(nb) ' ', total mass = ' num2str(round(sum(mb)/(132712440041.279419*1000^3/g)),3) ' M_',solar,' ', time
step = ' num2str(round(dt)) ' seconds, t = ' num2str(round((t1(1)/365.25/24/3600),2)) ' years'];
    title(ztitle,'Color',[1 1 1], 'Units','normalized', 'Position',[0.65, 0.95, 0], 'FontSize', 14)
    text(0.15, 0.02, 'M87 apparent black hole shadow N-body Gravity Simulation by Curvature of Wave Propagation', 'Units',
'Normalized', 'Color','White', 'FontSize', 14)

    legend('shadow limit', '', 'central mass','photon orbiting the central mass','', '1 solar mass star orbiting the central mass', '',
'1 solar mass star orbiting the central mass', '', 'light wavefront', 'light ray path','location','northwest', 'TextColor',[1 1 1],
'FontSize', 14)
    drawnow
    if record==1
        A=getframe(myplot);
        writeVideo(vid,A);
    end
end

% if dt_check==1
%     break
% end
ndt_plot = ndt_plot + 1;
end
if simo_option ==0
    if record==1
        close(vid); % close video file
    end
end
% if simo_option ==0
%     close ('all')
% end
toc

```

Example_4_Simple_trajectory.m

```
% Example_4_Simple_trajectory.m
%
% This example calculates a Simple Trajectory of several small objects
%-----
% Created by Morris G. Anderson
% https://www.researchgate.net/profile/Morris-Anderson
%
% Units, nomenclature, and documentation links provided in the accompanying script file:
% Background_and_Nomenclature.m
%
% Theory provided in the paper: N-body Gravity Simulation by Curvature of Wave Propagation
%-----
clc % Clear the command window.
clear

simo_option = 0; % Set = 0 if using MatLab, Set simo_option = 1 if using the SIMO MATLAB programming app

if simo_option == 0
    close('all'); % Close all figures
    clearvars; %
    clear
    simo_option = 0;
end

zlow = 1e-49; % zlow is a limiting value to prevent dividing by zero

if simo_option == 0
    lw = 2.0; % plot line width
    ms = 10; % plot symbol size
    plot_AR = 0.55; % plot window aspect ratio
    plot_pix = 1000
    myplot=figure('Position',[100 100 plot_AR*plot_pix plot_pix]);hold on; % figure window
    grid on
    ax = gca;
    ax.LineWidth = 2;
    ax.GridLineWidth = 2;
else
    lw = 2.0; % plot line width
    ms = 1; % plot symbol size
end

g = 0.66743e-10; % Newton's universal gravitational constant. Units = m^3/(kg-s)
cs = 299792458.0; % The standard speed of light, see Time Matter and Gravity (TMG) section 2.4.1
tau = 2*g/cs/cs;

disp(['Newton's G = ' num2str(g,6) ' m^3/(kg-s), Standard speed of light = ' num2str(cs,10) ' m/s'])
% n-body simulation inputs-----
plot_option = 1; % 1 = plot orbits, 2 = plot precession

nb = 6 % The number of governing bodies
b_pr = 1 % The body for calculating orbital precession (does not impact this simple trajectory solution)

tt = 10 % The total path time

dt = 0.05 % Time step, seconds. This is approximate, dt is recalculated below to match tt
nac = 10 % Number of time steps or arc segments per point that are saved for plotting

%-----
np = round((tt / dt) / nac) % The total number of points that could be plotted
ni = np * nac % Total number of time steps
dt = tt / ni % The time step

% For plotting orbits and other parameters
rx = zeros(np,nb); % rx = x position
ry = zeros(np,nb); % ry = y position
rz = zeros(np,nb); % rz = z position

% Initial conditions-----
beta_option = 0; % Initial motion input as velocity in m/s

znb = 1; % Earth
mb(znb) = 398600.435436*1000^3/g;
rx(1,znb) = 0.0; ry(1,znb) = 0; rz(1,znb) = 0;
vx(1,znb) = 1e-50; vy(1,znb) = 1e-50; vz(1,znb) = 1e-50
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

re = 6371008.7714; %Earth's IUGG mean radius (R1) see https://en.wikipedia.org/wiki/Earth_radius
c_inf = cs*exp((2*sum(mb(1))*g/cs/cs)/re); % calculate c with TMG eq 4-11.
disp(['c = ' num2str(c_inf,15) ' m/s, away from Earth at ' num2str(re,8) ' meters from the Sun at the average orbital radius of the Earth'])

znb = 2; % This is a object number 1
mb(znb) = 0.25;
rx(1,znb) = 0; ry(1,znb) = re; rz(1,znb) = 0;

alpha = 80; %degrees
v = 40; % m/s

vx(1,znb) = v*cos(alpha*pi/180);
vy(1,znb) = v*sin(alpha*pi/180);
vz(1,znb) = 0;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c
```

```

disp(['Initial velocity,  vx = ' num2str(vx(1,znb),4) ' m/s, vy = ' num2str(vy(1,znb),4) ' m/s, vz = ' num2str(vz(1,znb),4) ' m/s'])

znb = 3; % This is a object number 2
mb(znb) = 0.25;
rx(1,znb) = 0; ry(1,znb) = re; rz(1,znb) = 0;

alpha = 70; %degrees
v = 40; % m/s

vx(1,znb) = v*cos(alpha*pi/180)
vy(1,znb) = v*sin(alpha*pi/180)
vz(1,znb) = 0;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

disp(['Initial velocity,  vx = ' num2str(vx(1,znb),4) ' m/s, vy = ' num2str(vy(1,znb),4) ' m/s, vz = ' num2str(vz(1,znb),4) ' m/s'])

znb = 4; % This is a object number 3
mb(znb) = 0.25;
rx(1,znb) = 0; ry(1,znb) = re; rz(1,znb) = 0;
vx(1,znb) = 0; vy(1,znb) = 0; vz(1,znb) = 0;

alpha = 60; %degrees
v = 40; % m/s

vx(1,znb) = v*cos(alpha*pi/180);
vy(1,znb) = v*sin(alpha*pi/180);
vz(1,znb) = 0;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

disp(['Initial velocity,  vx = ' num2str(vx(1,znb),4) ' m/s, vy = ' num2str(vy(1,znb),4) ' m/s, vz = ' num2str(vz(1,znb),4) ' m/s'])

znb = 5; % This is a object number 4
mb(znb) = 0.25;
rx(1,znb) = 0; ry(1,znb) = re; rz(1,znb) = 0;
vx(1,znb) = 0; vy(1,znb) = 0; vz(1,znb) = 0;

alpha = 45; %degrees
v = 40; % m/s

vx(1,znb) = v*cos(alpha*pi/180);
vy(1,znb) = v*sin(alpha*pi/180);
vz(1,znb) = 0;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

disp(['Initial velocity,  vx = ' num2str(vx(1,znb),4) ' m/s, vy = ' num2str(vy(1,znb),4) ' m/s, vz = ' num2str(vz(1,znb),4) ' m/s'])

znb = 6; % This is a object number 4
mb(znb) = 0.25;
rx(1,znb) = 0; ry(1,znb) = re; rz(1,znb) = 0;
vx(1,znb) = 0; vy(1,znb) = 0; vz(1,znb) = 0;

alpha = 30; %degrees
v = 40; % m/s

vx(1,znb) = v*cos(alpha*pi/180);
vy(1,znb) = v*sin(alpha*pi/180);
vz(1,znb) = 0;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

disp(['Initial velocity,  vx = ' num2str(vx(1,znb),4) ' m/s, vy = ' num2str(vy(1,znb),4) ' m/s, vz = ' num2str(vz(1,znb),4) ' m/s'])

mb = mb(1,1:nb)';

tic

TMG_nb_initialization % This script file initializes the starting conditions

tstart(1,1:nb) = 0; %Starting time, in seconds
t1(1,1:nb) = 0; %Starting time, in seconds

for n = 1:np % Intagrate the trajectories

    TMG_nb_path_dt_integration; % This script calculates the arc path

    % Save values for plotting
    rx(n,:) = rx1';
    ry(n,:) = ry1';

end

plot(rx(:,2:nb), ry(:,2:nb) - re,'o','MarkerSize',ms,'linewidth', 1.5) % plot the saved positions for each body

hold('on')
% Calculate results using Newtons method based on a constant acceleration of gravity for comparison
ag= mb(1)*g/re/re
xn(1,1:znb) = 0;
yn(1,1:znb) = 0;
tn = 0;
for n = 2:ni
    tn = tn + dt;
    xn(n,:) = xn(1,:) + vx(1,:) * tn;
    yn(n,:) = yn(1,:) + vy(1,:) * tn - 0.5 * ag * tn * tn;
end
plot(xn(:,2:nb),yn(:,2:nb),'linewidth', lw)

hold('off') %format('long g');

```

```
if simo_option ==0
    set(gca,'fontsize',12)
end
xlabel('x, meter')
ylabel('y, meter')
title('The Curvature of Motion (symbols) versus Newtons method (curves)')

if simo_option ==0
    ax = gca;
    saveas(ax,['Plot_Ex_4_curvature_vs_newton.png'])
end

toc
```

Example_5_Star_cluster_random.m

```
% Example_5_Star_cluster_random.m
%
% This example simulates a random cluster.
%-----
% Created by Morris G. Anderson
% https://www.researchgate.net/profile/Morris-Anderson
%
% Units, nomenclature, and documentation links provided in the accompanying script file:
% Background_and_Nomenclature.m
%
% Theory provided in the paper: N-body Gravity Simulation by Curvature of Wave Propagation
%-----
clc % Clear the command window.
clear

simo_option = 0; % Set = 0 for MatLab, Set = 1 for the SIMO MATLAB programming app

record=0; % initialize value

if simo_option ==0
    close('all'); % Close all figures
    clearvars;
    simo_option = 0;
    record = 1; % set record=1 to record a video of the simulation
    solar=char(9737);
    plot_AR = 16/9; % plot window aspect ratio
    plot_pix = 1000;
    myplot=figure('Position',[100 100 plot_AR*plot_pix plot_pix]);hold on; % figure window
    grid on
    ax = gca;
    ax.LineWidth = 2;
    ax.GridLineWidth = 2;
    ax.Color = ('k');
    lw = 1.0; % plot line width
    ms = 5; % plot symbol size
    %plot_gAR = 55.3/23.3; % plot grid aspect ratio
    plot_gAR = 55.3/25.4; % plot grid aspect ratio
    if record==1
        vid = VideoWriter('Movie_Ex_5_Star_cluster_random.mp4','MPEG-4');
        vid.FrameRate = 30;
        open(vid);
    end
else
    lw = 1.0; % plot line width
    ms = 1; % plot symbol size
    plot_gAR = 49/29.1; % plot grid aspect ratio for ipad M2
end

g = 0.66743e-10; % Newton's universal gravitational constant. Units = m^3/(kg-s)
cs = 299792458.0; % The standard speed of light, see Time Matter and Gravity (TMG) section 2.4.1
tau = 2*g/cs/cs;
zlow = 1e-49;

disp(['Newton's G = ' num2str(g,6) ' m^3/(kg-s), Standard speed of light = ' num2str(cs,10) ' m/s'])
%-----
% n-body simulation inputs-----
%-----
nb = 3 % The number of governing bodies.
tmax = 36e6*365.25*24*3600 % Maximum simulation time, second

if simo_option ==0
    % suggested values for trying this out with MATLAB
    nps = 5000 % The number of points saved for plotting
    nac = 10 % 25 % 500 % The number of arc segments or time steps per point that are saved for plotting
    d_theta_max = 0.05 % 0.0025 % Max rotation per step, degree.
    dt_plot = 10000*365.25*24*60*60;% Time step for plotting results
else
    % suggested values for trying this out with SIMO
    nps = 200 % The number of points saved for plotting
    nac = 20 % The number of arc segments or time steps per point that are saved for plotting
    d_theta_max = 0.05 % Max rotation per step, degree.
    dt_plot = 20000*365.25*24*60*60;% Time step for plotting results
end

%dt_min = 1*365.25*24*3600 % Minimum dt allowed for simulation. This can override d_theta_max
dt_min = 0 % Minimum dt allowed for simulation. This can override d_theta_max

zvg = 0; % set zvg = 0 for approximately zero group velocity, set zvg = 1 for random group velocity
xy_only = 0; % set xy_only = 1 for xy plane only, else xyz solution is calculated
m_same = 0; % define m_same = 1 to set all bodies to the same mass, else random values will be applied
b_scalar = 0.55; % multiplier applied to circular orbit value for initial condition
%-----
d_theta_max = d_theta_max*pi/180; % Max rotation per step, radians.
% For plotting orbits and other parameters
rx = zeros(nps+1,nb); % rx = x position
ry = zeros(nps+1,nb); % ry = y position
rz = zeros(nps+1,nb); % rz = z position
b = zeros(nps,nb); % b = v/c
c = zeros(nps,nb); % c = the speed of light at the location of each body. The influence of the
pe = zeros(nps,nb);
ke = zeros(nps,nb);
t_vid= zeros(nps,1);
% individual body is ignored when calculating the speed of light at its location.
%-----
% Apply random numbers to define initial mass, v/c, and position for each body
beta_option(1:nb) = 1; % Initial motion input as v/c
```

```

% to simulate a solar system size group
% mb_total = 132712440041.279419*1000^3/g; % Total rest mass of the cluster, kg
% rmax = 1e10; % max initial semi-major axis

% To simulate a globular cluster
star_mass = 4.0 * 132712440041.279419*1000^3/g % average star mass in multipls of our Sun's mass, kg
mb_total = nb * star_mass % Total rest mass of the cluster, kg
star_per_cubic_ly = 1.5 % Stars per cubic light year % See Wikipedia "Open star cluster" density for Pleiades
light_year = 365.25 * 24 * 3600 * cs % meters, set this as the max initial semi-major axis
cluster_volume = (nb / star_per_cubic_ly) * light_year^3
rmax = ( 3 * cluster_volume / pi / 4 )^(1/3)

if m_same == 1
    mb(1:nb) = 1; % for equal mass simulation
else
    mb = rand(1,nb);
end

mb = mb_total * (mb/sum(mb));

position = rmax*randn(nb, 3);

for n = 1:(nps+1)

    rx(n,1:nb) = position(:,1)';
    ry(n,1:nb) = position(:,2)';
    rz(n,1:nb) = position(:,3)';
    if xy_only == 1
        rz(n,1:nb) = 0; % set xy_only = 1 for xy plane only, else xyz solution is calculated
    end
end

% set the center of mass equal to 0,0,0
rx = rx - sum(rx(1,:).*mb)/sum(mb);
ry = ry - sum(ry(1,:).*mb)/sum(mb);
rz = rz - sum(rz(1,:).*mb)/sum(mb);

r_semi_major = sqrt(rx(1,1:nb).^2 + ry(1,1:nb).^2 + rz(1,1:nb).^2 );

% Start with the circular orbital velocity corresponding to the semi-major axis. Calculate beta with TMG equation 7-12.
b_zz(1,1:nb) = b_scalar * sqrt( 1 ./ ( cs*cs.*r_semi_major ./ (sum(mb)-mb) / g) -1 ) );

% calculate beta unit vector directions
bx(1,1:nb) = randn(nb, 1)';
by(1,1:nb) = randn(nb, 1)';
bz(1,1:nb) = randn(nb, 1)';
if xy_only == 1
    bz(1,1:nb) = 0; % set xy_only = 1 for xy plane only, else xyz solution is calculated
end

b1 = sqrt(bx.^2 + by.^2 + bz.^2);

bx = b_zz .* bx./b1;
by = b_zz .* by./b1;
bz = b_zz .* bz./b1;

if zvg == 0
    % set the group momentum = 0,0,0
    bx = bx - sum(bx.*mb)/sum(mb);
    by = by - sum(by.*mb)/sum(mb);
    bz = bz - sum(bz.*mb)/sum(mb);
end

%bx = bx + 0.000005;

mb = mb'; % Transpose matrix for initialization

c_inf = cs; % Set speed of light away from group to cs

disp(['c = ' num2str(c_inf,15) ' m/s, away from the simulation influence' ])

disp(['Number of bodies simulated = ' num2str(nb,5) ])

tic

TMG_nb_initialization % This script initializes the solution
tstart(1,1:nb) = 0; % Starting time, in seconds
t1(1,1:nb) = 0; % Starting time, in seconds

plot_limit = 1.0*max(Dr(:)); % Limits for the orbital plot

zmo = mb'.*sqrt(1-b1.^2); % calculate the rest mass of each body
zmo_inf = zmo.*sqrt(c_inf./c1); % calculate the rest mass of each body a the standard location
ke_o(1,1:nb) = 100*(sum(zmo./sqrt(1-b1.^2)) - sum(zmo))/sum(mb); % Initial kinetic energy
pe_o(1,1:nb) = 100*(0.5*(sum(mb)-sum(zmo))/sum(mb)); % Initial potential energy

for n = 1:nps

    ke(n,1:nb) = ke_o(1,1:nb);
    pe(n,1:nb) = pe_o(1,1:nb);

end

ndt_plot = 1;
n = 0;
dt = 0;

```

```

while t1 < tmax % Integrate the path
    n=n+1;

    TMG_nb_path_dtheta_integration

    % Save values for plotting
    rx(1:(nps-1),1:nb) = rx(2:nps,1:nb); rx(nps,1:nb) = rx1';
    ry(1:(nps-1),1:nb) = ry(2:nps,1:nb); ry(nps,1:nb) = ry1';
    rz(1:(nps-1),1:nb) = rz(2:nps,1:nb); rz(nps,1:nb) = rz1';
    pe(1:(nps-1),1:nb) = pe(2:nps,1:nb);
    ke(1:(nps-1),1:nb) = ke(2:nps,1:nb);
    t_vid(1:(nps-1)) = t_vid(2:nps);

    zmo = zmo_inf.*sqrt(c1./c_inf); % calculate the rest mass of each body
    ke(nps,1:nb) = 100*(sum(zmo./sqrt(1-b1.^2)) - sum(zmo))/sum(mb); % calculate the kinetic energy in % of total mass
    pe(nps,1:nb) = 100*(0.5*(sum(mb)-sum(zmo))/sum(mb)); % calculate the potential energy in % of total mass

    t_vid(nps) = t1(1)/(3600*24*365.25);

    if t1(1) >= dt_plot*ndt_plot

        if simo_option ==0
            subplot(5,1,1:4)
            plot(rx(nps,1:nb)/light_year, ry(nps,1:nb)/light_year,'o','MarkerSize',ms,'MarkerFaceColor','w','MarkerEdgeColor','k') % plot
the saved positions for each body
            hold('on')
            if zvg ==0
                xlim([-plot_gAR*plot_limit/light_year,plot_gAR*plot_limit/light_year])
                ylim([-plot_limit/light_year,plot_limit/light_year])
            end
            % plot((rx(1:nps,1:nb)/light_year), (ry(1:nps,1:nb)/light_year),'linewidth', lw, 'lineStyle','-') % Plot the saved path
for each body
            ztitle = ['n-body = ' num2str(nb) ', total mass = ' num2str(round(sum(mb)/(132712440041.279419*1000^3/g)),3) ' M_',solar,
sprintf('dt = %.3g',(dt/365.25/24/3600)), ' years, t = ' num2str(round(t1(1)/365.25/24/3600)) ' years'];
            title(ztitle,'Color',[1 1 1], 'Units','normalized', 'Position',[0.5, 0.95, 0], 'FontSize', 14)
            ylabel('light years')
            xlabel('light years')
            hold('off')
            set(gca,'Color','k')
            text(0.1, 0.015, 'Calculated with an N-body Gravity Simulation by Curvature of Wave Propagation. 3-D solution viewed
in x-y plane','Units','Normalized','Color','White','FontSize', 14)

            subplot(5,1,5)
            plot (t_vid,ke-ke_o(1), 'Color',[1 0 0])
            hold('on')
            plot (t_vid,(pe_o(1)-pe), 'Color',[0 1 0])
            plot (t_vid,(ke-ke_o(1))+(pe_o(1)-pe), 'Color',[0 0 1])
            %xlim([t_vid(1),t_vid(nps)])
            xlim([t_vid(nps)-0.5e7,t_vid(nps)])
            text(0.02, 0.85, 'ke - ke_0', 'Units','Normalized','Color',[1 0 0], 'FontSize', 14)
            text(0.02, 0.15, 'pe - pe_0', 'Units','Normalized','Color',[0 1 0], 'FontSize', 14)
            text(0.02, 0.5, '(ke - ke_0) + (pe - pe_0)', 'Units','Normalized','Color',[0 0 1], 'FontSize', 14, 'BackgroundColor',
[1,1,1])

            hold('off')
            xlabel('time, years')
            ylabel('% of total mass')

            drawnow
            if record==1
                A=getframe(myplot);
                writeVideo(vid,A);
            end
            ndt_plot = ndt_plot + 1;
        else
            plot(rx(nps,1:nb)/light_year, ry(nps,1:nb)/light_year,'o','MarkerSize',ms,'MarkerFaceColor','w','MarkerEdgeColor','k') % plot
the saved positions for each body
            hold('on')
            if zvg ==0
                xlim([-plot_gAR*plot_limit/light_year,plot_gAR*plot_limit/light_year])
                ylim([-plot_limit/light_year,plot_limit/light_year])
            end
            plot((rx(1:nps,1:nb)/light_year), (ry(1:nps,1:nb)/light_year),'linewidth', lw, 'lineStyle','-') % Plot the saved path for each
body
            hold('off')
            end
            ndt_plot = ndt_plot + 1;
        end
        if dt_check==1
            break
        end
    end

    if record==1
        close(vid); % close video file
    end
    toc

```

Example_5_Star_cluster_symmetric.m

```
% Example_5_Star_cluster_symmetric.m
%
% This example simulates a symmetrical cluster. Numerical truncation errors cause it to lose symmetry after several orbits.
%-----
% Created by Morris G. Anderson
% https://www.researchgate.net/profile/Morris-Anderson
%
% Units, nomenclature, and documentation links provided in the accompanying script file:
% Background_and_Nomenclature.m
%
% Theory provided in the paper: N-body Gravity Simulation by Curvature of Wave Propagation
%-----
clc % Clear the command window.
clear

simo_option = 0; % Set = 0 for MatLab, Set = 1 for the SIMO MATLAB programming app

record=0; % initialize value

if simo_option ==0
    close ('all'); % Close all figures
    clearvars;
    simo_option = 0;
    record = 1; % set record=1 to record a video of the simulation
    solar=char(9737);
    plot_AR = 16/9; % plot window aspect ratio
    plot_pix = 1000;
    myplot=figure('Position',[100 100 plot_AR*plot_pix plot_pix]);hold on; % figure window
    grid on
    ax = gca;
    ax.LineWidth = 2;
    ax.GridLineWidth = 2;
    ax.Color = ('k');
    lw = 2.0; % plot line width
    ms = 10; % plot symbol size
    %plot_gAR = 55.3/23.3; % plot grid aspect ratio
    plot_gAR = 55.3/25.4; % plot grid aspect ratio
    if record==1
        vid = VideoWriter('Movie_Ex_5_Star_cluster_symmetric.mp4','MPEG-4');
        vid.FrameRate = 30;
        open(vid);
    end
else
    lw = 1.0; % plot line width
    ms = 1; % plot symbol size
    plot_gAR = 49/29.1; % plot grid aspect ratio for ipad M2
end

g = 0.66743e-10; % Newton's universal gravitational constant. Units = m^3/(kg-s)
cs = 299792458.0; % The standard speed of light, see Time Matter and Gravity (TMG) section 2.4.1
tau = 2*g/cs/cs;
zlow = 1e-49;

disp(['Newton's G = ' num2str(g,6) ' m^3/(kg-s), Standard speed of light = ' num2str(cs,10) ' m/s'])
%-----
% n-body simulation inputs-----
%-----
nb = 3 % The number of governing bodies.
tmax = 36e6*365.25*24*3600 % Maximum simulation time, second

if simo_option ==0
    % suggested values for trying this out with MATLAB
    nps = 5000 % The number of points saved for plotting
    nac = 10 % 500 % The number of arc segments or time steps per point that are saved for plotting
    d_theta_max = 0.05 % 0.0025 % Max rotation per step, degree.
    dt_plot = 10000*365.25*24*60*60;% Time step for plotting results
else
    % suggested values for trying this out with SIMO
    nps = 200 % The number of points saved for plotting
    nac = 20 % The number of arc segments or time steps per point that are saved for plotting
    d_theta_max = 0.05 % Max rotation per step, degree.
    dt_plot = 20000*365.25*24*60*60;% Time step for plotting results
end

dt_min = 0.0001*365.25*24*3600 % Minimum dt allowed for simulation. This can override d_theta_max

zvg = 0.0; % set zvg = 0 for approximately zero group velocity, set zvg = 1 for random group velocity
m_same = 1; % define m_same = 1 to set all bodies to the same mass, else random values will be applied
b_scalar = 0.35; % Used to calculate the initial beta. This multiplier is
%b_scalar = 0.6; %starting with nb = 10 % Used to calculate the initial beta. This multiplier is
%-----
d_theta_max = d_theta_max*pi/180; % Max rotation per step, radians.
% For plotting orbits and other parameters
rx = zeros(nps+1,nb); % rx = x position
ry = zeros(nps+1,nb); % ry = y position
rz = zeros(nps+1,nb); % rz = z position
b = zeros(nps,nb); % b = v/c
c = zeros(nps,nb); % c = the speed of light at the location of each body. The influence of the
pe = zeros(nps,nb);
ke = zeros(nps,nb);
t_vid= zeros(nps,1);
% individual body is ignored when calculating the speed of light at its location.
%-----
% Apply random numbers to define initial mass, v/c, and position for each body
beta_option(1:nb) = 1; % Initial motion input as v/c

% to simulate a solar system size group
```

```

% mb_total = 132712440041.279419*1000^3/g; % Total rest mass of the cluster, kg
% rmax = 1e10; % max initial semi-major axis

% To simulate a globular cluster
star_mass = 4.0 * 132712440041.279419*1000^3/g % average star mass in multiples of our Sun's mass, kg
mb_total = nb * star_mass % Total rest mass of the cluster, kg
star_per_cubic_ly = 1.5 % Stars per cubic light year
light_year = 365.25 * 24 * 3600 * cs % meters, set this as the max initial semi-major axis
cluster_volume = (nb / star_per_cubic_ly) * light_year^3
rmax = ( 4 * cluster_volume / pi / 4 )^(1/3)

if m_same == 1
    mb(1:nb) = 1; % for equal mass simulation
else
    mb = rand(1,nb);
end

mb = mb_total * (mb/sum(mb));

%----- symmetrical input -----
zn=1:nb;
angular_separation=(2*pi*zn/nb)'; % polar angles for each object's initial position

for n=1:(nps+1)
    rx(n,1:nb) = rmax*cos(angular_separation)'; % Cartesian coordinates of the objects
    ry(n,1:nb) = rmax*sin(angular_separation)';
    rz(n,1:nb) = 0.*angular_separation';
end
%-----

% set the center of mass equal to 0,0,0
rx = rx - sum(rx(1,:).*mb)/sum(mb);
ry = ry - sum(ry(1,:).*mb)/sum(mb);
rz = rz - sum(rz(1,:).*mb)/sum(mb);

r_semi_major = sqrt(rx(1,1:nb).^2 + ry(1,1:nb).^2 + rz(1,1:nb).^2);

% Start with the circular orbital velocity corresponding to the semi-major axis as if all off the mass is located at the center.
% Calculate beta with TMG equation 7-12.
b_zz(1,1:nb) = b_scalar * sqrt( 1 ./ ( (cs*cs.*r_semi_major ./ (sum(mb)-mb) / g) -1 ) );

%----- symmetrical velocities -----
bx(1,1:nb) = -b_zz.*sin(angular_separation)';
by(1,1:nb) = b_zz.*cos(angular_separation)';
bz(1,1:nb) = b_zz.*0.*(angular_separation)';
%-----

b1 = sqrt(bx.^2 + by.^2 + bz.^2);

bx = b_zz .* bx./b1;
by = b_zz .* by./b1;
bz = b_zz .* bz./b1;

if zvg ==0
    % set the group momentum = 0,0,0
    bx = bx - sum(bx.*mb)/sum(mb);
    by = by - sum(by.*mb)/sum(mb);
    bz = bz - sum(bz.*mb)/sum(mb);
end

mb = mb'; % Transpose matrix for initialization

c_inf = cs; % Set speed of light away from group to cs

disp(['c = ' num2str(c_inf,15) ' m/s, away from the simulation influence' ])

disp(['Number of bodies simulated = ' num2str(nb,5) ])

tic

TMG_nb_initialization % This script initializes the solution
tstart(1,1:nb) = 0; % Starting time, in seconds
t1(1,1:nb) = 0; % Starting time, in seconds

plot_limit = 1.0*max(Dr(:)); % Limits for the orbital plot

zmo = mb'.*sqrt(1-b1.^2); % calculate the rest mass of each body
zmo_inf = zmo.*sqrt(c_inf./cl); % calculate the rest mass of each body at the standard location
ke_o(1,1:nb) = 100*(sum(zmo./sqrt(1-b1.^2)) - sum(zmo))/sum(mb); % Initial kinetic energy
pe_o(1,1:nb) = 100*(0.5*(sum(mb)-sum(zmo))/sum(mb)); % Initial potential energy

for n = 1:nps

    ke(n,1:nb) = ke_o(1,1:nb);
    pe(n,1:nb) = pe_o(1,1:nb);

end

ndt_plot = 1;
n= 0;
dt = 0;

while t1 < tmax % Integrate the path
    n=n+1;

    TMG_nb_path_dtheta_integration

    % Save values for plotting
    rx(1:(nps-1),1:nb) = rx(2:nps,1:nb); rx(nps,1:nb) = rx1';

```

```

ry(1:(nps-1),1:nb) = ry(2:nps,1:nb); ry(nps,1:nb) = ry1';
rz(1:(nps-1),1:nb) = rz(2:nps,1:nb); rz(nps,1:nb) = rz1';
pe(1:(nps-1),1:nb) = pe(2:nps,1:nb);
ke(1:(nps-1),1:nb) = ke(2:nps,1:nb);
t_vid(1:(nps-1)) = t_vid(2:nps);

zmo = zmo_inf.*sqrt(c1./c_inf); % calculate the rest mass of each body
ke(nps,1:nb) = 100*(sum(zmo./sqrt(1-b1.^2)) - sum(zmo))/sum(mb); % calculate the kinetic energy in % of total mass
pe(nps,1:nb) = 100*(0.5*(sum(mb)-sum(zmo))/sum(mb)); % calculate the potential energy in % of total mass

t_vid(nps) = t1(1)/(3600*24*365.25);

if t1(1) > dt_plot*ndt_plot

    if simo_option ==0
        subplot(5,1,4)
        plot(rx(nps,1:nb)/light_year, ry(nps,1:nb)/light_year,'o','MarkerSize',ms,'MarkerFaceColor','w','MarkerEdgeColor','k') % plot
the saved positions for each body
        hold('on')
        if zvg ==0
            xlim([-plot_gAR*plot_limit/light_year,plot_gAR*plot_limit/light_year])
            ylim([-plot_limit/light_year,plot_limit/light_year])
        end
        plot((rx(1:nps,1:nb)/light_year), (ry(1:nps,1:nb)/light_year),'linewidth', lw, 'lineStyle','-') % Plot the saved path for each
body
        ztitle(['n-body = ' num2str(nb) ', total mass = ' num2str(round(sum(mb)/(132712440041.279419*1000^3/g)),3) ' M_',solar,', dt
=' num2str(round(dt/365.25/24/3600),3) ' years, t = ' num2str(round(t1(1)/365.25/24/3600)) ' years'];
        title(ztitle,'Color',[1 1 1], 'Units','normalized', 'Position',[0.5, 0.95, 0], 'FontSize', 14)
        ylabel('light years')
        xlabel('light years')
        hold('off')
        set(gca,'Color','k')
        text(0.1, 0.015, 'Calculated with an N-body Gravity Simulation by Curvature of Wave Propagation. 3-D solution viewed
in x-y plane','Units','Normalized','Color','White','FontSize', 14)

        subplot(5,1,5)
        plot (t_vid,ke-ke_o(1), 'Color', [1 0 0])
        hold('on')
        plot (t_vid,(pe_o(1)-pe), 'Color', [0 1 0])
        plot (t_vid,(ke-ke_o(1))+(pe_o(1)-pe), 'Color', [0 0 1])
        %xlim([t_vid(1),t_vid(nps)])
        xlim([t_vid(nps)-1e7,t_vid(nps)])
        text(0.02, 0.85, 'ke - ke_0', 'Units', 'Normalized', 'Color', [1 0 0], 'FontSize', 14)
        text(0.02, 0.15, 'pe - pe_0', 'Units', 'Normalized', 'Color', [0 1 0], 'FontSize', 14)
        text(0.02, 0.5, '(ke - ke_0) + (pe - pe_0)', 'Units', 'Normalized', 'Color', [0 0 1], 'FontSize', 14, 'BackgroundColor',
[1,1,1])

        hold('off')
        xlabel('time, years')
        ylabel('% of total mass')

        drawnow
        if record==1
            A=getframe(myplot);
            writeVideo(vid,A);
        end
        ndt_plot = ndt_plot + 1;
    else
        plot(rx(nps,1:nb)/light_year, ry(nps,1:nb)/light_year,'o','MarkerSize',ms,'MarkerFaceColor','w','MarkerEdgeColor','k') % plot
the saved positions for each body
        hold('on')
        if zvg ==0
            xlim([-plot_gAR*plot_limit/light_year,plot_gAR*plot_limit/light_year])
            ylim([-plot_limit/light_year,plot_limit/light_year])
        end
        plot((rx(1:nps,1:nb)/light_year), (ry(1:nps,1:nb)/light_year),'linewidth', lw, 'lineStyle','-') % Plot the saved path for each
body
        hold('off')
        end
        ndt_plot = ndt_plot + 1;
    end
    if dt_check==1
        break
    end
end

if record==1
    close(vid); % close video file
end
toc

```

Example_6_Pioneer_10_11_Voyager_1_2.m

```
% Example_6_Pioneer_10_11_Voyager_1_2.m
%
% This example simulates the Pioneer 10, 11, Voyager 1 and 2 spacecraft.
%-----
% Created by Morris G. Anderson
% https://www.researchgate.net/profile/Morris-Anderson
%
% Units, nomenclature, and documentation links provided in the accompanying script file:
% Background_and_Nomenclature.m
%
% Theory provided in the paper: N-body Gravity Simulation by Curvature of Wave Propagation
%-----
% SIMO MATLAB app notes
% Simo is not as capable as MATLAB. Therefore, the following variable
% settings are recommended
%
% on line 24 set simo_option = 1
% on line 72 set d_theta_max = 0.25
% on line 75 set dgps = 120
%-----
clc % Clear the command window.
clear

simo_option = 0; % Set=0 if using MatLab, Set simo_option=1 if using the SIMO MATLAB programming app

record=0; % initialize value

if simo_option ==0
    close('all'); % Close all figures
    clearvars;
    simo_option = 0;
    record=0; % set record=1 to record a video of the simulation
    plot_AR = 16/9; % plot window aspect ratio
    plot_pix = 1000;
    myplot=figure('Position',[100 100 plot_AR*plot_pix plot_pix]);%hold on; % figure window
    grid on
    ax = gca;
    ax.LineWidth = 2;
    ax.GridLineWidth = 2;
    ax.Color = ('k');
    lw = 2.0; % plot line width
    ms = 5; % plot symbol size
    %plot_gAR = 55.3/29.4; % plot grid aspect ratio
    plot_gAR = 55.3/32.25; % plot grid aspect ratio
    if record==1
        vid = VideoWriter('Movie_Ex_6_Pioneer_and_voyager_probes.mp4','MPEG-4');
        vid.FrameRate = 30;
        open(vid);
    end
else
    lw = 1.0; % plot line width
    ms = 1; % plot symbol size
    plot_gAR = 49/29.1; % plot grid aspect ratio for ipad M2
end

dt_plot = 5*24*3600; % Time step for saving plotting and saving frames for video

g = 0.66743e-10; % Newton's universal gravitational constant. Units = m^3/(kg-s)
cs = 299792458.0; % The standard speed of light, see Time Matter and Gravity (TMG) section 2.4.1
tau = 2*g/cs/cs;
zlow = 1e-49; % zlow is a limiting value to prevent dividing by zero

disp(['Newton's G = ' num2str(g,6) ' m^3/(kg-s), Standard speed of light = ' num2str(cs,10) ' m/s'])

% n-body simulation inputs-----
plot_option_1 = 1; % If = 1, plot orbits. If = 0, plot precession.
plot_limit_scalar = 0.65; % plot limit = plot_limit_scalar * max distance from body 1
plot_scalar_2 = 1.0 %0.6; % this scalar is combined with the previous one for the initial part of the solution
zd = 1; % If = 1, rotation is positive (right-hand rule). If = -1, rotation is negative.
nb = 13 % The number of governing bodies.
d_theta_max = 0.0025 %0.00125 % 0.0025 for video % Max rotation per step, deg.
dv_add = 1 % Set = 1 to apply coarse corrections. Set = 0 for no corrections.
%-----
dgps = 5 % degrees per point saved (set dgps<=5 for more accurate distance to planet calculation)
nac = round(dgps/d_theta_max) % The number of arc segments per point that saved for plotting
nps = round(30*(365.25/88)*360/d_theta_max/nac) % The number of points saved for plotting.
d_theta_max = d_theta_max*pi/180; % Max rotation per step, radians.
%-----
dt_min = 0 % Minimum dt allowed for simulation. This can override d_theta_max
%-----
% For plotting orbits and other parameters
rx = zeros(nps+1,nb); % rx = x position
ry = zeros(nps+1,nb); % ry = y position
rz = zeros(nps+1,nb); % rz = z position
rc = zeros(nps,nb); % rc = radius of curvature
b = zeros(nps,nb); % b = v/c
c = zeros(nps,nb); % c = the speed of light at the location of each body. The influence of the
% individual body is ignored when calculating the speed of light at its location.
tb = zeros(nps,nb); % tb = path time for each body, s
zDr = zeros(nps,1); % for calculating distances between specific bodies
%-----
% Read in initial starting conditions for each body

zSS_PS_1973_04_12 % This script initializes for the pioneer spacecraft missions

vg(1:3) = 0; % Set group velocity = 0. This example is not setup for adding in a group velocity.
```

```

% Calculate c away from the earth in the solar system. Also calculate c away from the solar system.

re = 6371008.7714; % Earth's IUGG mean radius (R1) see https://en.wikipedia.org/wiki/Earth_radius
reoa = 149.598e9; % Earth's Semimajor orbital axis, see https://nssdc.gsfc.nasa.gov/planetary/factsheet/earthfact.html

% c_inf = cs*exp((2*sum(mb(4))*g/cs/cs)/re); % Calculate c away from the earth with TMG eq 4-11.
%
% disp(['c = ' num2str(c_inf,15) ' m/s, away from Earth at ' num2str(re,8) ' meters from the Sun or the Earth's average orbital radius'])
%
% c_inf = c_inf*exp((2*(sum(mb)-mb(4))*g/cs/cs)/reoa); % Calculate c away from the solar system TMG eq 4-11.

% The initial conditions for this study are taken from the NASA JPL
% Horizons website. the Horizons system is based on Barycentric Dynamical
% Time (TDB). This crucial information is clearly explained in the
% following comment that is included with vectors downloaded from the
% Horizons system.
%
% Barycentric Dynamical Time ("TDB" or Teph) output was requested.
% This continuous coordinate time is equivalent to the relativistic
% proper time of a clock at rest in a reference frame co-moving with
% the solar system barycenter but outside the system's gravity well.
% It is the independent variable in the solar system relativistic
% equations of motion.
%
% TDB runs at a uniform rate of one SI second per second and is
% independent of irregularities in Earth's rotation.
%
% According to this information, the Horizons system output for time,
% length, velocity, and gravitational mass (GM) are based on a convention
% where the time standard location is set at infinity or outside the
% gravitational well of the solar system.

c_inf = 299792458.0

disp(['c = ' num2str(c_inf,15) ' m/s, away from the Solar Systems influence consistent to NASA JPL Horizons TDB' ])

% Reset scalars to the number of simulated bodies

mb = mb(1,1:nb)';

rx = rx(1,1:nb);
ry = ry(1,1:nb);
rz = rz(1,1:nb);

for n = 1:(nps)
    rx(n,1:nb) = rx(1,1:nb);
    ry(n,1:nb) = ry(1,1:nb);
    rz(n,1:nb) = rz(1,1:nb);
end

vx = vx(1,1:nb);
vy = vy(1,1:nb);
vz = vz(1,1:nb);

vx = zd.*vx; vy = zd.*vy; vz = zd.*vz;

disp(['Number of bodies simulated = ' num2str(nb,5) ])

% Calculate initial group velocity and re-set it to 0.0

vgx_initial = sum(vx(1,:).*mb')/sum(mb)
vgy_initial = sum(vy(1,:).*mb')/sum(mb)
vgz_initial = sum(vz(1,:).*mb')/sum(mb)

vx = vx - vgx_initial
vy = vy - vgy_initial
vz = vz - vgz_initial

vgx_new = sum(vx(1,:).*mb')/sum(mb)
vgy_new = sum(vy(1,:).*mb')/sum(mb)
vgz_new = sum(vz(1,:).*mb')/sum(mb)

tic

TMG_nb_initialization % This script initializes the solution
tstart(1,1:nb) = 0; % Starting time, in seconds
t1(1,1:nb) = 0; % Starting time, in seconds

plot_limit = plot_limit_scalar*max(Dr(:,1)); % Limits for the orbital plot
tp = 0; % Initialize the perihelion time for the body of interest
tp_count = 1;

r_j = 69911*1000; % The radius of Jupiter, meters
r_s = 58232*1000; % The radius of Saturn, meters
r_u = 25362*1000; % The radius of Uranus, meters
r_n = 24624*1000; % The radius of Neptune, meters

% define body for saving Dr, the closest approach to the planet
for n = 1:(nps)
    zDr_12_7(n) = Dr(12,7); % 12 = pioneer 10, 7 = Jupiter
    zDr_13_7(n) = Dr(13,7); % 13 = pioneer 11, 7 = Jupiter
    zDr_13_8(n) = Dr(13,8); % 13 = pioneer 11, 8 = Saturn
end
ndt_plot = 1;
n = 0;
%-----

tmax = 372*24*3600; % Calculate the solution to 1974-04-19

```

```

while mean(t1) < tmax % Integrate the path
    n=n+1;
    TMG_nb_path_dtheta_integration % This script calculates the path for each body
    % Save values for plotting
    rx(1:(nps-1),1:nb) = rx(2:nps,1:nb); rx(nps,1:nb) = rx1';
    ry(1:(nps-1),1:nb) = ry(2:nps,1:nb); ry(nps,1:nb) = ry1';
    rz(1:(nps-1),1:nb) = rz(2:nps,1:nb); rz(nps,1:nb) = rz1';
    zDr_12_7(1:nps-1) = zDr_12_7(2:nps); zDr_12_7(nps) = Dr(12,7);

    if t1(1) > dt_plot*ndt_plot

        plot(-ry(nps,1:11), rx(nps,1:11), 'o','MarkerSize',ms, 'MarkerFaceColor', 'w')
        hold('on')
        xlim([-plot_gAR*plot_limit*plot_scalar_2,plot_gAR*plot_limit*plot_scalar_2])
        ylim([-plot_limit*plot_scalar_2,plot_limit*plot_scalar_2])
        plot(-ry(:,1:11), rx(:,1:11),'linewidth', lw, 'lineStyle','-')
        plot(-ry(nps,12:13), rx(nps,12:13),'o','MarkerSize',ms,'color','y','MarkerFaceColor','y') % Pioneer spacecraft
        plot(-ry(:,12:13), rx(1:nps,12:13),'color','y','linewidth', lw, 'lineStyle',':') % Pioneer spacecraft
        if simo_option ==0
            text(2.0e12*plot_scalar_2, 2e12*plot_scalar_2, num2str(1973.277 + t1(1)/24/3600/365.25,5), 'Color', 'White', 'FontSize',16)
            text(2.0e12, 2e12, num2str(1973.277 + t1(1)/24/3600/365.25,5), 'Color', 'White', 'FontSize',16)
            text(-ry(nps,12)-0.05*plot_limit, rx(nps,12)+0.04*plot_limit,'P10','Color', 'Yellow', 'FontSize',12)
            text(-ry(nps,13)-0.1*plot_limit*plot_scalar_2, rx(nps,13)+0.02*plot_limit,'P11','Color', 'Yellow', 'FontSize',12)
            text(-ry(nps,8)*1.05, rx(nps,8)-0.05*plot_limit,'Saturn','Color', 'white', 'FontSize',12)
            text(0.25, 0.95, 'NASA Pioneer 10, 11, and Voyager 1, 2 probe trajectories calculated with an', 'Units', 'Normalized', 'Color',
'White', 'FontSize',14)
            text(0.3, 0.92, 'N-body Gravity Simulation by Curvature of Wave Propagation', 'Units', 'Normalized', 'Color', 'White',
'FontSize',14)
            end
            xlabel('meter')
            ylabel('meter')
            hold('off')

        if simo_option ==0
            set(gca,'Color','k')
            drawnow
            if record==1
                A=getframe(myplot);
                writeVideo(vid,A);
            end
        end
        if dt_check==1
            break
        end
        ndt_plot = ndt_plot + 1;
    end
end
dt_check=0;
disp(['Approximately ' num2str(mean(t1)/24/3600,6) ' days, this is the point of the first Pioneer 11 course correction on 19 April 1974'])

disp(['Pioneer 10 closest approach to Jupiter = ' num2str((min(zDr_12_7) - r_j)/1000,6) ' km'])
disp(['dgps = ' num2str(dgps) ' degrees'])
disp('set dgps<=5 degrees for more accurate distance to planet calculation')

%*****
% Pioneer 11 Course correction on 19 April 1974, see NASA SP-446 page 77
%*****

dv_correction = 63.7 * dv_add % Course correction on 19 April 1974, see NASA SP-446 page 77
dv(2) = -13.9 * dv_add; % y velocity change
dv(3) = -1.35 * dv_add; % z velocity change
dv(1) = sqrt(dv_correction^2 - dv(2)^2 - dv(3)^2); % x velocity change

bul(:,13) = (dv' + b1(13).*c1(13).*bul(:,13))./c1(13);
b1(:,13) = ((bul(1,13).^2 + bul(2,13).^2 + bul(3,13).^2 ).^(1/2));
bul(:,13) = bul(:,13) ./ b1(:,13);

% save course correction points
ncp = 1;
corr_points_x (ncp) = 1.7e12; % lable position for correction points
corr_points_y (ncp) = -2.0e12; % lable position for correction points
corr_points_z (ncp) = 0; % lable position for correction points

ncp = ncp +1;
corr_points_x (ncp) = rx(nps,13);
corr_points_y (ncp) = ry(nps,13);
corr_points_z (ncp) = rz(nps,13);

%-----
tmax = (1918.0 -(tmax/24/3600))*24*3600 + tmax; % Calculate the solution to 1978-07-13

while mean(t1) < tmax % Integrate the path
    n=n+1;
    TMG_nb_path_dtheta_integration % This script calculates the path for each body
    % Save values for plotting
    rx(1:(nps-1),1:nb) = rx(2:nps,1:nb); rx(nps,1:nb) = rx1';
    ry(1:(nps-1),1:nb) = ry(2:nps,1:nb); ry(nps,1:nb) = ry1';
    rz(1:(nps-1),1:nb) = rz(2:nps,1:nb); rz(nps,1:nb) = rz1';
    zDr_13_7(1:nps-1) = zDr_13_7(2:nps); zDr_13_7(nps) = Dr(13,7);

    if t1(1) > dt_plot*ndt_plot

        plot(-ry(nps,1:11), rx(nps,1:11), 'o','MarkerSize',ms, 'MarkerFaceColor', 'w')
        hold('on')
        xlim([-plot_gAR*plot_limit*plot_scalar_2,plot_gAR*plot_limit*plot_scalar_2])
        ylim([-plot_limit*plot_scalar_2,plot_limit*plot_scalar_2])
        plot(-ry(:,1:11), rx(:,1:11),'linewidth', lw, 'lineStyle','-')
        plot(-ry(nps,12:13), rx(nps,12:13),'o','MarkerSize',ms,'color','y','MarkerFaceColor','y') % Pioneer spacecraft

```

```

plot(-ry(:,12:13), rx(1:nps,12:13), 'color', 'y', 'linewidth', lw, 'lineStyle', ':') % Pioneer spacecraft
plot(-corr_points_y(1)*plot_scalar_2, corr_points_x(1)*plot_scalar_2, 'h', 'MarkerSize', 2*ms, 'color', [1 0.0 0.0], 'MarkerFaceColor',
[1 0.0 0.0])
plot(-corr_points_y(2:npc), corr_points_x(2:npc), 'h', 'MarkerSize', 2*ms, 'color', [1 0.0 0.0], 'MarkerFaceColor', [1 0.0 0.0])
if simo_option == 0
    text(2.0e12*plot_scalar_2, 2e12*plot_scalar_2, num2str(1973.277 + t1(1)/24/3600/365.25,5), 'Color', 'White', 'FontSize', 16)
    text(2.1e12*plot_scalar_2, 1.7e12*plot_scalar_2, 'simulated course corrections', 'Color', 'White', 'FontSize', 14)
    text(-ry(nps,12)-0.05*plot_limit, rx(nps,12)+0.04*plot_limit, 'P10', 'Color', 'Yellow', 'FontSize', 12)
    text(-ry(nps,13)-0.1*plot_limit*plot_scalar_2, rx(nps,13)+0.02*plot_limit, 'P11', 'Color', 'Yellow', 'FontSize', 12)
    text(-ry(nps,8)*1.05, rx(nps,8)-0.05*plot_limit, 'Saturn', 'Color', 'White', 'FontSize', 12)
    text(0.25, 0.95, 'NASA Pioneer 10, 11, and Voyager 1, 2 probe trajectories calculated with an', 'Units', 'Normalized', 'Color',
'White', 'FontSize', 14)
    text(0.3, 0.92, 'N-body Gravity Simulation by Curvature of Wave Propagation', 'Units', 'Normalized', 'Color', 'White',
'FontSize', 14)
end
xlabel('meter')
ylabel('meter')
hold('off')

if simo_option == 0
    set(gca, 'Color', 'k')
    drawnow
    if record == 1
        A = getframe(myplot);
        writeVideo(vid, A);
    end
end
if dt_check == 1
    break
end
ndt_plot = ndt_plot + 1;
end
dt_check = 0;
disp(['Approximately ' num2str(mean(t1)/24/3600,6) ' days, 1978-07-13'])

disp(['Pioneer 11 closest approach to Jupiter = ' num2str((min(zDr_13_7) - r_j)/1000,6) ' km'])
disp(['dgps = ' num2str(dgps) ' degrees'])
disp('set dgps <= 5 degrees for more accurate distance to planet calculation')

%*****
% Pioneer 11 estimated course correction
%*****
dv(1) = 8.34 * dv_add; % For Pioneer 11. I use this to simulate the reported two corrections that were made.
dv(2) = dv(1); % y velocity change
dv(3) = -30.0 * dv_add; % z velocity change

dv_correction = sqrt(dv(1)^2 + dv(2)^2 + dv(3)^2)

bul(:,13) = (dv' + b1(13).*c1(13).*bul(:,13))./c1(13);
b1(:,13) = ((bul(1,13).^2 + bul(2,13).^2 + bul(3,13).^2).^^(1/2));
bul(:,13) = bul(:,13) ./ b1(:,13);

% save course correction points

ncp = ncp + 1;
corr_points_x(ncp) = rx(nps,13);
corr_points_y(ncp) = ry(nps,13);
corr_points_z(ncp) = rz(nps,13);

% -----
% Add in Voyager 1 and 2. Initialize from previous solution
nb_save = nb;
rca_save = rca;
rx_save = rx; ry_save = ry; rz_save = rz;
bx(1,:) = b1.*bul(1,:); by(1,:) = b1.*bul(2,:); bz(1,:) = b1.*bul(3,:);
beta_option(1:nb) = 1; % Initial motion input as v/c
rx = rx1'; ry = ry1'; rz = rz1';

nb = nb + 1; % add in Voyager 1
mb(nb) = 1500.0;
% *****
% Ephemeris / WWW_USER Mon Sep 4 14:38:19 2023 Pasadena, USA / Horizons
% *****
disp(['Body ' num2str(nb) ' = Voyager 1 2443702.500000000 = A.D. 1978-Jul-13 00:00:00.0000 TDB'])
%$$$SOE
%2443702.500000000 = A.D. 1978-Jul-13 00:00:00.0000 TDB
X = -2.025227086803138E+08; Y = 5.092445669207466E+08; Z = 7.736712045741200E+06;
VX = -1.519226753853006E+01; VY = 8.642735970035670E+00; VZ = 6.889172431985413E-02;
rx(1,nb) = X*1000; ry(1,nb) = Y*1000; rz(1,nb) = Z*1000;
vx(1,nb) = VX*1000; vy(1,nb) = VY*1000; vz(1,nb) = VZ*1000;
beta_option(nb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c
c(1,nb) = 0; % Initialize storage space for this value
% -----
nb = nb + 1; % add in Voyager 2
mb(nb) = 1500.0;
% *****
% Ephemeris / WWW_USER Tue Sep 5 12:10:30 2023 Pasadena, USA / Horizons
% *****
disp(['Body ' num2str(nb) ' = Voyager 2 2443702.500000000 = A.D. 1978-Jul-13 00:00:00.0000 TDB'])
%$$$SOE
%2443702.500000000 = A.D. 1978-Jul-13 00:00:00.0000 TDB
X = -2.065210492530237E+08; Y = 4.756052857129858E+08; Z = 2.468854971198025E+07;
VX = -1.501921326309367E+01; VY = 6.377489239436725E+00; VZ = -2.231079129994993E-01;
rx(1,nb) = X*1000; ry(1,nb) = Y*1000; rz(1,nb) = Z*1000;
vx(1,nb) = VX*1000; vy(1,nb) = VY*1000; vz(1,nb) = VZ*1000;
beta_option(nb) = 0; % Initial motion input as velocity in m/s

```

```

%beta_option(znb) = 1; % Initial motion input as v/c
c(1,nb) = 0; % Initialize storage space for this value

TMG_nb_initialization % This script initializes the solution

rca(1:nb_save) = rca_save(1:nb_save); % Combine Voyager conditions with previous solution.

z_temp = rx_save; z_temp(1:nps,nb-1) = rx1(nb-1); z_temp(1:nps,nb) = rx1(nb); z_temp(nps,1:nb) = rx'; rx = z_temp;
z_temp = ry_save; z_temp(1:nps,nb-1) = ry1(nb-1); z_temp(1:nps,nb) = ry1(nb); z_temp(nps,1:nb) = ry'; ry = z_temp;
z_temp = rz_save; z_temp(1:nps,nb-1) = rz1(nb-1); z_temp(1:nps,nb) = rz1(nb); z_temp(nps,1:nb) = rz'; rz = z_temp;

t1((nb-1):nb) = mean(t1);

% define body for saving Dr, the closest approach to the planet
for n = 1:(nps)
    zDr_14_8(n) = Dr(14,8); % 14 = Voyager 1, 8 = Saturn
    zDr_15_8(n) = Dr(15,8); % 15 = Voyager 2, 8 = Saturn
    zDr_15_9(n) = Dr(15,9); % 15 = Voyager 2, 9 = Uranus
    zDr_15_10(n) = Dr(15,10); % 15 = Voyager 2, 10 = Neptune
end

%-----
tmax = (2424.0 - (tmax/24/3600))*24*3600 + tmax; % Calculate the solution to 1979-12-01

while mean(t1) < tmax % Integrate the path
    n=n+1;
    TMG_nb_path_dtheta_integration % This script calculates the path for each body
    % Save values for plotting
    rx(1:(nps-1),1:nb) = rx(2:nps,1:nb); rx(nps,1:nb) = rx1';
    ry(1:(nps-1),1:nb) = ry(2:nps,1:nb); ry(nps,1:nb) = ry1';
    rz(1:(nps-1),1:nb) = rz(2:nps,1:nb); rz(nps,1:nb) = rz1';
    zDr_14_8(1:nps-1) = zDr_14_8(2:nps); zDr_14_8(nps) = Dr(14,8);
    zDr_15_8(1:nps-1) = zDr_15_8(2:nps); zDr_15_8(nps) = Dr(15,8);

    if t1(1) > dt_plot*ndt_plot

        plot(-ry(nps,1:11), rx(nps,1:11), 'o','MarkerSize',ms, 'MarkerFaceColor', 'w')
        hold('on')
        xlim([-plot_gAR*plot_limit*plot_scalar_2,plot_gAR*plot_limit*plot_scalar_2])
        ylim([-plot_limit*plot_scalar_2,plot_limit*plot_scalar_2])
        plot(-ry(1:11), rx(1:11),'linewidth', lw, 'lineStyle','-')
        plot(-ry(nps,12:13), rx(nps,12:13),'o','MarkerSize',ms,'color','y','MarkerFaceColor','y') % Pioneer spacecraft
        plot(-ry(1:12:13), rx(1:nps,12:13),'color','y','linewidth', lw, 'lineStyle',':') % Pioneer spacecraft
        plot(-ry(nps,14:15), rx(nps,14:15),'o','MarkerSize',ms,'color','y','MarkerFaceColor','g') % Voyager spacecraft
        plot(-ry(1:14:15), rx(1:nps,14:15),'color','g','linewidth', lw, 'lineStyle','--') % Voyager spacecraft
        plot(-corr_points_y(1)*plot_scalar_2, corr_points_x(1)*plot_scalar_2, 'h','MarkerSize',2*ms,'color',[1 0.0 0.0],'MarkerFaceColor',
[1 0.0 0.0])
        plot(-corr_points_y(2:npc), corr_points_x(2:npc), 'h','MarkerSize',2*ms,'color',[1 0.0 0.0],'MarkerFaceColor',[1 0.0 0.0])
        if simo_option ==0
            text(2.0e12*plot_scalar_2, 2e12*plot_scalar_2, num2str(1973.277 + t1(1)/24/3600/365.25,5), 'Color', 'White', 'FontSize',16)
            text(2.1e12*plot_scalar_2, 1.7e12*plot_scalar_2, 'simulated course corrections', 'Color', 'White', 'FontSize',14)
            text(-ry(nps,12)-0.05*plot_limit, rx(nps,12)+0.04*plot_limit,'P10','Color', 'Yellow', 'FontSize',12)
            text(-ry(nps,13)+0.02*plot_limit, rx(nps,13)+0.02*plot_limit,'P11','Color', 'Yellow', 'FontSize',12)
            text(-ry(nps,8)*1.05, rx(nps,8)-0.05*plot_limit,'Saturn','Color', 'white', 'FontSize',12)
            text(0.25, 0.95, 'NASA Pioneer 10, 11, and Voyager 1, 2 probe trajectories calculated with an', 'Units', 'Normalized', 'Color',
'White', 'FontSize',14)
            text(0.3, 0.92, 'N-body Gravity Simulation by Curvature of Wave Propagation', 'Units', 'Normalized', 'Color', 'White',
'FontSize',14)
        end
        xlabel('meter')
        ylabel('meter')
        hold('off')

        if simo_option ==0
            set(gca,'Color','k')
            drawnow
            if record==1
                A=getframe(myplot);
                writeVideo(vid,A);
            end
        end
        if dt_check==1
            break
        end
        ndt_plot = ndt_plot + 1;
    end
end
dt_check=0;
disp(['Approximately ' num2str(mean(t1)/24/3600,6) ' days, 1979-12-01'])

%*****
% Voyager 1 and 2 estimated course corrections
%*****

% Voyager 1 course correction
dv(1) = -30.0/1.2 * dv_add; % For Voyager 1. I use this to simulate a course correction
dv(2) = -26.5/1.2 * dv_add; % y velocity change
dv(3) = 11.1*0.94 * dv_add; % z velocity change

dv_correction = sqrt(dv(1)^2+dv(2)^2+dv(3)^2)

bul(:,14) = (dv' + b1(14).*c1(14).*bul(:,14))./c1(14);
b1(:,14) = ((bul(1,14).^2 + bul(2,14).^2 + bul(3,14).^2 ).^(1/2));
bul(:,14) = bul(:,14) ./ b1(:,14);

% save course correction points
ncp = ncp + 1;
corr_points_x(ncp) = rx(nps,14);
corr_points_y(ncp) = ry(nps,14);

```

```

corr_points_z (ncp) = rz(nps,14);

% Voyager 2 course correction

% The following values are for d_theta_max = 0.0025
dv(1)= 12.72*1.007902 * dv_add; % x velocity change
dv(2)= -91.10/1.007902 * dv_add; % y velocity change
dv(3)= -0.157325 * dv_add; % z velocity change

% The following values are for d_theta_max = 0.00125
dv(1)= 12.72 * dv_add; % x velocity change
dv(2)= -91.101675 * dv_add; % y velocity change
dv(3)= 0.04634 * dv_add % z velocity change

dv_correction = sqrt(dv(1)^2+dv(2)^2+dv(3)^2)

bul(:,15) = (dv' + b1(15).*c1(15).*bul(:,15))./c1(15);
b1(:,15) = ((bul(1,15).^2 + bul(2,15).^2 + bul(3,15).^2 ).^(1/2));
bul(:,15) = bul(:,15) ./ b1(:,15);

% save course correction points
ncp = ncp + 1;
corr_points_x (ncp) = rx(nps,15);
corr_points_y (ncp) = ry(nps,15);
corr_points_z (ncp) = rz(nps,15);

%-----
tmax = (3186.0 - (tmax/24/3600))*24*3600 + tmax; % Calculate the solution to 1982-01-01
%tmax = (4671-200-(tmax/24/3600))*24*3600 + tmax; % working to Uranus flyby

while mean(t1) < tmax % Integrate the path
    n=n+1;
    TMG_nb_path_dtheta_integration % This script calculates the path for each body
    % Save values for plotting
    rx(1:(nps-1),1:nb) = rx(2:nps,1:nb); rx(nps,1:nb) = rx1';
    ry(1:(nps-1),1:nb) = ry(2:nps,1:nb); ry(nps,1:nb) = ry1';
    rz(1:(nps-1),1:nb) = rz(2:nps,1:nb); rz(nps,1:nb) = rz1';
    zDr_13_8(1:nps-1) = zDr_13_8(2:nps); zDr_13_8(nps) = Dr(13,8);
    zDr_14_8(1:nps-1) = zDr_14_8(2:nps); zDr_14_8(nps) = Dr(14,8);
    zDr_15_8(1:nps-1) = zDr_15_8(2:nps); zDr_15_8(nps) = Dr(15,8);

    if t1(1) > dt_plot*ndt_plot

        plot(-ry(nps,1:11), rx(nps,1:11), 'o','MarkerSize',ms, 'MarkerFaceColor', 'w')
        hold('on')
        xlim([-plot_gAR*plot_limit*plot_scalar_2,plot_gAR*plot_limit*plot_scalar_2])
        ylim([-plot_limit*plot_scalar_2,plot_limit*plot_scalar_2])
        plot(-ry(:,1:11), rx(:,1:11),'linewidth', lw, 'lineStyle','--')
        plot(-ry(nps,12:13), rx(nps,12:13),'o','MarkerSize',ms,'color','y','MarkerFaceColor','y') % Pioneer spacecraft
        plot(-ry(:,12:13), rx(1:nps,12:13),'color','y','linewidth', lw, 'lineStyle',':') % Pioneer spacecraft
        plot(-ry(nps,14:15), rx(nps,14:15),'o','MarkerSize',ms,'color','y','MarkerFaceColor','g') % Voyager spacecraft
        plot(-ry(:,14:15), rx(1:nps,14:15),'color','g','linewidth', lw, 'lineStyle','--') % Voyager spacecraft
        plot(-corr_points_y(1)*plot_scalar_2, corr_points_x(1)*plot_scalar_2, 'h','MarkerSize',2*ms,'color',[1 0.0 0.0],'MarkerFaceColor',[1 0.0 0.0])
        plot(-corr_points_y(2:ncp), corr_points_x(2:ncp), 'h','MarkerSize',2*ms,'color',[1 0.0 0.0],'MarkerFaceColor',[1 0.0 0.0])
        if simo_option ==0
            text(2.0e12*plot_scalar_2, 2e12*plot_scalar_2, num2str(1973.277 + t1(1)/24/3600/365.25,5), 'Color', 'White', 'FontSize',16)
            text(2.1e12*plot_scalar_2, 1.7e12*plot_scalar_2, 'simulated course corrections', 'Color', 'White', 'FontSize',14)
            text(-ry(nps,12)+0.02*plot_limit, rx(nps,12)+0.02*plot_limit,'P10','Color', 'Yellow', 'FontSize',12)
            text(-ry(nps,13)+0.02*plot_limit, rx(nps,13)+0.02*plot_limit,'P11','Color', 'Yellow', 'FontSize',12)
            text(0.25, 0.95, 'NASA Pioneer 10, 11, and Voyager 1, 2 probe trajectories calculated with an', 'Units', 'Normalized', 'Color', 'White', 'FontSize',14)
            text(0.3, 0.92, 'N-body Gravity Simulation by Curvature of Wave Propagation', 'Units', 'Normalized', 'Color', 'White', 'FontSize',14)
        end
        xlabel('meter')
        ylabel('meter')
        hold('off')

        if simo_option ==0
            set(gca,'Color','k')
            drawnow
            if record==1
                A=getframe(myplot);
                writeVideo(vid,A);
            end
        end
        if dt_check==1
            break
        end
        ndt_plot = ndt_plot + 1;
    end
end
dt_check=0;
disp(['Approximately ' num2str(mean(t1)/24/3600,6) ' days, 1982-01-01'])

%*****
% Voyager 2 estimated course corrections
%*****
% Voyager 2 course correction
% The following values are for d_theta_max = 0.0025
dv(1)= 0.0615 * dv_add; % For Voyager 2. I use this to simulate a course correction
dv(2)= 0.0 * dv_add; % y velocity change
dv(3)= -0.29586 * dv_add; % z velocity change

% % Voyager 2 course correction
% % The following values are for d_theta_max = 0.00125
% dv(1)= 3.48875 * dv_add; % For Voyager 2. I use this to simulate a course correction

```

```

% dv(2)= 0.0 * dv_add; % y velocity change
% dv(3)= -2.37189 * dv_add; % z velocity change

dv_correction = sqrt(dv(1)^2+dv(2)^2+dv(3)^2)

bul(:,15) = (dv' + b1(15).*c1(15).*bul(:,15))./c1(15);
b1(:,15) = ((bul(1,15).^2 + bul(2,15).^2 + bul(3,15).^2).^^(1/2));
bul(:,15) = bul(:,15) ./ b1(:,15);

% save course correction points
ncp = ncp + 1;
corr_points_x(ncp) = rx(nps,15);
corr_points_y(ncp) = ry(nps,15);
corr_points_z(ncp) = rz(nps,15);

%-----
% tmax = (5979.0 - (tmax/24/3600))*24*3600 + tmax; % Calculate the solution to 1989-08-25
% tmax = (6473.0 - (tmax/24/3600))*24*3600 + tmax; % Calculate the solution to 1991-01-01
tmax = (7934.0 - (tmax/24/3600))*24*3600 + tmax; % Calculate the solution to 1995-01-01
% tmax = (4671 + 200 - (tmax/24/3600))*24*3600 + tmax; % working to Uranus flyby
% tmax = (5979.0 + 200 - (tmax/24/3600))*24*3600 + tmax; % working to just after Neptune

%nac = 1000

while mean(t1) < tmax
    n=n+1;
    TMG_nb_path_dtheta_integration % This script calculates the path for each body
    % Save values for plotting
    rx(1:(nps-1),1:nb) = rx(2:nps,1:nb); rx(nps,1:nb) = rx1';
    ry(1:(nps-1),1:nb) = ry(2:nps,1:nb); ry(nps,1:nb) = ry1';
    rz(1:(nps-1),1:nb) = rz(2:nps,1:nb); rz(nps,1:nb) = rz1';
    zDr_15_9(1:nps-1) = zDr_15_9(2:nps); zDr_15_9(nps) = Dr(15,9);
    zDr_15_10(1:nps-1) = zDr_15_10(2:nps); zDr_15_10(nps) = Dr(15,10);

    if t1(1) > dt_plot*ndt_plot

        plot(-ry(nps,1:11), rx(nps,1:11), 'o','MarkerSize',ms, 'MarkerFaceColor', 'w')
        hold('on')
        xlim([-plot_gAR*plot_limit,plot_gAR*plot_limit])
        ylim([-plot_limit,plot_limit])
        plot(-ry(:,1:11), rx(:,1:11),'linewidth', lw, 'lineStyle','--')
        plot(-ry(nps,12:13), rx(nps,12:13),'o','MarkerSize',ms,'color','y','MarkerFaceColor','y') % Pioneer spacecraft
        plot(-ry(:,12:13), rx(1:nps,12:13),'color','y','linewidth', lw, 'lineStyle',':') % Pioneer spacecraft
        plot(-ry(nps,14:15), rx(nps,14:15),'o','MarkerSize',ms,'color','y','MarkerFaceColor','g') % Voyager spacecraft
        plot(-ry(:,14:15), rx(1:nps,14:15),'color','g','linewidth', lw, 'lineStyle','--') % Voyager spacecraft
        plot(-corr_points_y, corr_points_x, 'h','MarkerSize',2*ms,'color',[1 0.0 0.0],'MarkerFaceColor',[1 0.0 0.0])
        if simo_option ==0
            text(2.0e12, 2e12, num2str(1973.277 + t1(1)/24/3600/365.25,5), 'Color', 'White', 'FontSize',16)
            text(2.1e12, 1.7e12, 'simulated course corrections', 'Color', 'White', 'FontSize',14)
            text(-ry(nps,9)*0.95, rx(nps,9)+0.05*plot_limit,'Uranus','Color', 'white', 'FontSize',12)
            text(-ry(nps,10)*0.95, rx(nps,10)+0.05*plot_limit,'Neptune','Color', 'white', 'FontSize',12)
            text(-ry(nps,11)*0.95, rx(nps,11)+0.05*plot_limit,'Pluto','Color', 'white', 'FontSize',12)
            text(-ry(nps,12)+0.02*plot_limit, rx(nps,12)+0.02*plot_limit,'P10','Color', 'Yellow', 'FontSize',12)
            text(-ry(nps,13)+0.02*plot_limit, rx(nps,13)+0.02*plot_limit,'P11','Color', 'Yellow', 'FontSize',12)
            text(-ry(nps,14)+0.02*plot_limit, rx(nps,14)+0.02*plot_limit,'V1','Color', 'Green', 'FontSize',12)
            text(-ry(nps,15)+0.02*plot_limit, rx(nps,15)+0.02*plot_limit,'V2','Color', 'Green', 'FontSize',12)
            text(0.25, 0.95, 'NASA Pioneer 10, 11, and Voyager 1, 2 probe trajectories calculated with an', 'Units', 'Normalized', 'Color',
'White', 'FontSize',14)
            text(0.3, 0.92, 'N-body Gravity Simulation by Curvature of Wave Propagation', 'Units', 'Normalized', 'Color', 'White',
'FontSize',14)
        end
        xlabel('meter')
        ylabel('meter')
        hold('off')

        if simo_option ==0
            set(gca,'Color','k')
            drawnow
            if record==1
                A=getframe(myplot);
                writeVideo(vid,A);
            end
        end
        if dt_check==1
            break
        end
        ndt_plot = ndt_plot + 1;
    end
end
dt_check=0;

disp(['Pioneer 11 closest approach to Saturn = ' num2str((min(zDr_13_8) - r_s)/1000,6) ' km'])
disp(['Voyager 1 closest approach to Saturn = ' num2str((min(zDr_14_8) - r_s)/1000,6) ' km'])
disp(['Voyager 2 closest approach to Saturn = ' num2str((min(zDr_15_8) - r_s)/1000,6) ' km'])
disp(['Voyager 2 closest approach to Uranus = ' num2str((min(zDr_15_9) - r_u)/1000,6) ' km'])
disp(['Voyager 2 closest approach to Neptune = ' num2str((min(zDr_15_10) - r_n)/1000,6) ' km'])
disp(['dgps = ' num2str(dgps) ' degrees'])
disp('set dgps<=5 degrees for more accurate distance to planet calculation')

disp(['Total simulation = approximately ' num2str(mean(t1)/24/3600,6) ' days'])

if simo_option ==0
    if record==1
        close(vid); % close video file
    end
end
end

% Read in JPL Horizons data
z_horizons_p10 = csvread('Pioneer_10_Spacecraft_horizons_xyz_2024_12_17.csv');
z_horizons_p11 = csvread('Pioneer_11_Spacecraft_horizons_xyz_2024_12_17.csv');

```

```

z_horizons_v1 = csvread('Voyager_1_Spacecraft_horizons_xyz_2024_12_17.csv');
z_horizons_v2 = csvread('Voyager_2_Spacecraft_horizons_xyz_2024_12_17.csv');

plot(-ry(nps,1:11), rx(nps,1:11), 'o','MarkerSize',ms, 'MarkerFaceColor', 'w')
hold('on')
xlim([-plot_gAR*plot_limit,plot_gAR*plot_limit])
ylim([-plot_limit,plot_limit])
plot(-ry(:,1:11), rx(:,1:11),'linewidth', lw, 'lineStyle','-')
plot(-ry(nps,12:13), rx(nps,12:13),'o','MarkerSize',ms,'color','y','MarkerFaceColor','y') % Pioneer spacecraft
plot(-ry(:,12:13), rx(1:nps,12:13),'color','y','linewidth', lw, 'lineStyle',':') % Pioneer spacecraft
plot(-ry(nps,14:15), rx(nps,14:15),'o','MarkerSize',ms,'color','y','MarkerFaceColor','g') % Voyager spacecraft
plot(-ry(:,14:15), rx(1:nps,14:15),'color','g','linewidth', lw, 'lineStyle','--') % Voyager spacecraft
plot(-corr_points_y, corr_points_x, 'h','MarkerSize',2*ms,'color',[1 0.0 0.0],'MarkerFaceColor',[1 0.0 0.0])
if simo_option ==0
    text(2.0e12, 2e12, num2str(1973.277 + t1(1)/24/3600/365.25,5), 'Color', 'White', 'FontSize',16)
    text(2.1e12, 1.7e12, 'simulated course corrections', 'Color', 'White', 'FontSize',14)
    text(-ry(nps,8), rx(nps,8)+0.05*plot_limit,'Saturn','Color', 'white', 'FontSize',12)
    text(-ry(nps,9), rx(nps,9)+0.05*plot_limit,'Uranus','Color', 'white', 'FontSize',12)
    text(-ry(nps,10), rx(nps,10)+0.05*plot_limit,'Neptune','Color', 'white', 'FontSize',12)
    text(-ry(nps,11), rx(nps,11)+0.05*plot_limit,'Pluto','Color', 'white', 'FontSize',12)
    text(-ry(nps,12)+0.02*plot_limit, rx(nps,12)+0.02*plot_limit,'P10','Color', 'Yellow', 'FontSize',12)
    text(-ry(nps,13)+0.02*plot_limit, rx(nps,13)+0.02*plot_limit,'P11','Color', 'Yellow', 'FontSize',12)
    text(-ry(nps,14)+0.02*plot_limit, rx(nps,14)+0.02*plot_limit,'V1','Color', 'Green', 'FontSize',12)
    text(-ry(nps,15)+0.02*plot_limit, rx(nps,15)+0.02*plot_limit,'V2','Color', 'Green', 'FontSize',12)
end
plot(-z_horizons_p10(:,2)*1000,z_horizons_p10(:,1)*1000,'color','w','linewidth', 0.25, 'lineStyle','-')
plot(-z_horizons_p11(:,2)*1000,z_horizons_p11(:,1)*1000,'color','w','linewidth', 0.25, 'lineStyle','-')
plot(-z_horizons_v1(:,2)*1000,z_horizons_v1(:,1)*1000,'color','w','linewidth', 0.25, 'lineStyle','-')
plot(-z_horizons_v2(:,2)*1000,z_horizons_v2(:,1)*1000,'color','w','linewidth', 0.25, 'lineStyle','-')
hold('off')

if simo_option ==0
    set(gca,'Color','k')
    drawnow
    ax = gca;
    exportgraphics(ax,'Plot_Ex_6_probes_myx.png')
end

plot(rx(nps,1:11), rz(nps,1:11), 'o','MarkerSize',ms, 'MarkerFaceColor', 'w')
hold('on')
xlim([-plot_gAR*plot_limit,plot_gAR*plot_limit])
ylim([-plot_limit,plot_limit])
plot(rx(:,1:11), rz(:,1:11),'linewidth', lw, 'lineStyle','-')
plot(rx(nps,12:13), rz(nps,12:13),'o','MarkerSize',ms,'color','y','MarkerFaceColor','y') % Pioneer spacecraft
plot(rx(:,12:13), rz(1:nps,12:13),'color','y','linewidth', lw, 'lineStyle',':') % Pioneer spacecraft
plot(rx(nps,14:15), rz(nps,14:15),'o','MarkerSize',ms,'color','y','MarkerFaceColor','g') % Voyager spacecraft
plot(rx(:,14:15), rz(1:nps,14:15),'color','g','linewidth', lw, 'lineStyle','--') % Voyager spacecraft

plot(z_horizons_p10(:,1)*1000,z_horizons_p10(:,3)*1000,'color','w','linewidth', 0.25, 'lineStyle','-')
plot(z_horizons_p11(:,1)*1000,z_horizons_p11(:,3)*1000,'color','w','linewidth', 0.25, 'lineStyle','-')
plot(z_horizons_v1(:,1)*1000,z_horizons_v1(:,3)*1000,'color','w','linewidth', 0.25, 'lineStyle','-')
plot(z_horizons_v2(:,1)*1000,z_horizons_v2(:,3)*1000,'color','w','linewidth', 0.25, 'lineStyle','-')
hold('off')

if simo_option ==0
    set(gca,'Color','k')
    drawnow
    ax = gca;
    exportgraphics(ax,'Plot_Ex_6_probes_xz.png')
    %close ('all')
    save('Out_Ex_6_Pioneer_10_11_Voyager_1_2');
end
toc

vgx_final = sum(b1(1,:).*bul(1,:).*c1.*mb)/sum(mb)
vgx_final = sum(b1(1,:).*bul(2,:).*c1.*mb)/sum(mb)
vgx_final = sum(b1(1,:).*bul(3,:).*c1.*mb)/sum(mb)

```

Data File: zSS_1970_01_25_downloaded_on_2023_06_26.m

```
% zSS_1970_01_25_downloaded_on_2023_06_26.m
%
% Planetary System Masses are from
% The JPL Planetary and Lunar Ephemerides DE440 and DE441 Table 2
% https://iopscience.iop.org/article/10.3847/1538-3881/abd414
% https://ssd.jpl.nasa.gov/astro_par.html
% The Planetary and Lunar Ephemerides DE430 and DE431 Table 12 - Use this source for Pallas and Juno
% http://ipnpr.jpl.nasa.gov/progress_report/42-196/196C.pdf
%
% Initial position and velocity relative to the Solar System Barycenter are from
% https://ssd.jpl.nasa.gov/horizons/app.html#/

disp(['Solar System bodies for 2440611.500000000 = A.D. 1970-Jan-25 00:00:00.0000 TDB'])
disp(['Ephemeris / WWW_USER Mon Jun 26 20:49:03 2023 Pasadena, USA / Horizons'])
g = 0.66743e-10 % Newton's universal gravitational constant. Units = m^3/(kg-s)

znb = 0;

znb = znb+1; % Sun
rb(znb) = 696500 * 1000; % Radius (photosphere), m
mb(znb) = 132712440041.279419*1000^3/g; % Rest mass, kg
%*****
%Ephemeris / WWW_USER Mon Jun 26 20:49:03 2023 Pasadena, USA / Horizons
%*****
%Target body name: Sun (10) {source: DE441}
%Center body name: Solar System Barycenter (0) {source: DE441}
%Center-site name: BODY CENTER
disp(['Body ' num2str(znb,3) ' = Sun 2440611.500000000 = A.D. 1970-Jan-25 00:00:00.0000 TDB'])
%$$$OE
% 2440611.500000000 = A.D. 1970-Jan-25 00:00:00.0000 TDB
X = 6.372991721565161E+05; Y = 2.937637542261903E+05; Z = -8.992258913619749E+03;
VX=-3.483068968377863E-03; VY= 9.046429242754891E-03; VZ= 4.047218717945006E-05;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

znb = znb+1; % Mercury
rb(znb) = 2440 * 1000; % radius, m
mb(znb) = 22031.868551*1000^3/g; % Rest mass, kg
%*****
%Ephemeris / WWW_USER Mon Jun 26 20:56:21 2023 Pasadena, USA / Horizons
%*****
%Target body name: Mercury (199) {source: DE441}
%Center body name: Solar System Barycenter (0) {source: DE441}
%Center-site name: BODY CENTER
disp(['Body ' num2str(znb,3) ' = Mercury 2440611.500000000 = A.D. 1970-Jan-25 00:00:00.0000 TDB'])
%*****
%$$$OE
% 2440611.500000000 = A.D. 1970-Jan-25 00:00:00.0000 TDB
X = -5.503879040460826E+07; Y = 7.637358563276584E+06; Z = 5.705217454928251E+06;
VX=-1.648877542482009E+01; VY=-4.618068342281613E+01; VZ=-2.257001367680303E+00;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

znb = znb+1; % Venus
rb(znb) = 6051.84 * 1000; % radius, m
mb(znb) = 324858.592*1000^3/g; % Rest mass, kg
%*****
%Ephemeris / WWW_USER Mon Jun 26 20:57:48 2023 Pasadena, USA / Horizons
%*****
%Target body name: Venus (299) {source: DE441}
%Center body name: Solar System Barycenter (0) {source: DE441}
%Center-site name: BODY CENTER
disp(['Body ' num2str(znb,3) ' = Venus 2440611.500000000 = A.D. 1970-Jan-25 00:00:00.0000 TDB'])
%$$$OE
% 2440611.500000000 = A.D. 1970-Jan-25 00:00:00.0000 TDB
X = 6.325335392822134E+07; Y = -8.872594306892014E+07; Z = -4.834105963581923E+06;
VX= 2.840528969073845E+01; VY= 2.003369242057543E+01; VZ=-1.368401487687009E+00;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

znb = znb+1; % Earth
rb(znb) = 6371.01 * 1000; % radius, m
mb(znb) = 398600.435507*1000^3/g; % Rest mass, kg
%*****
%Ephemeris / WWW_USER Mon Jun 26 20:59:13 2023 Pasadena, USA / Horizons
%*****
%Target body name: Earth (399) {source: DE441}
%Center body name: Solar System Barycenter (0) {source: DE441}
%Center-site name: BODY CENTER
disp(['Body ' num2str(znb,3) ' = Earth 2440611.500000000 = A.D. 1970-Jan-25 00:00:00.0000 TDB'])
%$$$OE
% 2440611.500000000 = A.D. 1970-Jan-25 00:00:00.0000 TDB
X = -8.388566864675777E+07; Y = 1.209003834599575E+08; Z = -1.493910087369382E+03;
VX=-2.487736065868874E+01; VY=-1.718677228094805E+01; VZ=-1.886196855505062E-04;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c
```

```

znb = znb+1; % Earth moon
rb(znb) = 1737.53 * 1000; % radius, m
mb(znb) = 4902.800118*1000^3/g; % Rest mass, kg
%*****
%Ephemeris / WWW_USER Mon Jun 26 21:00:28 2023 Pasadena, USA / Horizons
%*****
%Target body name: Moon (301) {source: DE441}
%Center body name: Solar System Barycenter (0) {source: DE441}
%Center-site name: BODY CENTER
disp(['Body ' num2str(znb,3) ' = Earth Moon 2440611.500000000 = A.D. 1970-Jan-25 00:00:00.0000 TDB'])
%$$$SOE
% 2440611.500000000 = A.D. 1970-Jan-25 00:00:00.0000 TDB
X = -8.424185424313150E+07; Y = 1.210927824804784E+08; Z = 5.508081829056144E+03;
VX = -2.532467376749580E+01; VY = -1.804714210500794E+01; VZ = -8.695467804848089E-02;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

znb = znb+1; % Mars
rb(znb) = 3389.92 * 1000; % radius, m
mb(znb) = 42828.375214*1000^3/g; % Rest mass, kg
%*****
%Ephemeris / WWW_USER Sat Jul 8 18:47:46 2023 Pasadena, USA / Horizons
%*****
%Target body name: Mars (499) {source: mar097}
%Center body name: Solar System Barycenter (0) {source: DE441}
%Center-site name: BODY CENTER
disp(['Body ' num2str(znb,3) ' = Mars 2440611.500000000 = A.D. 1970-Jan-25 00:00:00.0000 TDB'])
%$$$SOE
% 2440611.500000000 = A.D. 1970-Jan-25 00:00:00.0000 TDB
X = 1.776720447359633E+08; Y = 1.231874310156784E+08; Z = -1.798712260728620E+06;
VX = -1.288951287530203E+01; VY = 2.197979260226854E+01; VZ = 7.776709205366652E-01;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

znb = znb+1; % Jupiter
rb(znb) = 69911 * 1000; % Vol. Mean Radius, m
mb(znb) = 126686531.900*1000^3/g; % Rest mass, kg
%*****
%Ephemeris / WWW_USER Sat Jul 8 19:33:05 2023 Pasadena, USA / Horizons
%*****
%Target body name: Jupiter (599) {source: jup365_merged}
%Center body name: Solar System Barycenter (0) {source: DE441}
%Center-site name: BODY CENTER
disp(['Body ' num2str(znb,3) ' = Jupiter 2440611.500000000 = A.D. 1970-Jan-25 00:00:00.0000 TDB'])
%$$$SOE
% 2440611.500000000 = A.D. 1970-Jan-25 00:00:00.0000 TDB
X = -7.382892198014171E+08; Y = -3.433800206202698E+08; Z = 1.795894220545726E+07;
VX = 5.356722179397164E+00; VY = -1.124006541027567E+01; VZ = -7.381409116879212E-02;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

```

Data File: zSS_PS_1973_04_12.m

```
% zSS_PS_1973_04_12.m downloaded on 2023-08-22
%
% Planetary System Masses are from
% The JPL Planetary and Lunar Ephemerides DE440 and DE441 Table 2
% https://iopscience.iop.org/article/10.3847/1538-3881/abd414
% https://ssd.jpl.nasa.gov/astro_par.html
% The Planetary and Lunar Ephemerides DE430 and DE431 Table 12 - Use this source for Pallas and Juno
% http://ipnpr.jpl.nasa.gov/progress_report/42-196/196C.pdf
%
% Initial position and velocity relative to the Solar System Barycenter are from
% https://ssd.jpl.nasa.gov/horizons/app.html#/

disp(['Solar System bodies for April 12, 1973 with planetary system velocity and mass'])

znb = 0;

disp(['Solar System bodies for 2441784.500000000 = A.D. 1973-Apr-12 00:00:00.0000 TDB'])
disp(['Ephemeris / WWW_USER Tue Aug 22 19:32:52 2023 Pasadena, USA / Horizons'])

znb = znb+1; % Sun
rb(znb) = 696500 * 1000; % Radius (photosphere), m
mb(znb) = 132712440041.279419*1000^3/g; % Rest mass, kg
% *****
% Ephemeris / WWW_USER Tue Aug 22 19:32:52 2023 Pasadena, USA / Horizons
% *****
% Target body name: Sun (10) {source: DE441}
% Center body name: Solar System Barycenter (0) {source: DE441}
% Center-site name: BODY CENTER
disp(['Body ' num2str(znb,3) ' = Sun 2441784.500000000 = A.D. 1973-Apr-12 00:00:00.0000 TDB'])
% $$$OE
% 2441784.500000000 = A.D. 1973-Apr-12 00:00:00.0000 TDB
X = -1.923392302128695E+05; Y = 5.209918913468969E+05; Z = 6.061603157293692E+03;
VX = -8.339913902147377E-03; VY = -6.434694011379806E-03; VZ = 1.653807444119671E-04;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

znb = znb+1; % Mercury
mb(znb) = 22031.868551*1000^3/g; % Rest mass, kg
% *****
% Ephemeris / WWW_USER Tue Aug 22 19:36:05 2023 Pasadena, USA / Horizons
% *****
% Target body name: Mercury (199) {source: DE441}
% Center body name: Solar System Barycenter (0) {source: DE441}
% Center-site name: BODY CENTER
disp(['Body ' num2str(znb,3) ' = Mercury 2441784.500000000 = A.D. 1973-Apr-12 00:00:00.0000 TDB'])
% $$$OE
% 2441784.500000000 = A.D. 1973-Apr-12 00:00:00.0000 TDB
X = -6.239854884672338E+06; Y = -6.867923625198168E+07; Z = -5.088854587889042E+06;
VX = 3.875148202114654E+01; VY = -1.748121543909687E+00; VZ = -3.702299163804894E+00;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

znb = znb+1; % Venus
mb(znb) = 324858.592*1000^3/g; % Rest mass, kg
% *****
% Ephemeris / WWW_USER Tue Aug 22 19:43:20 2023 Pasadena, USA / Horizons
% *****
% Target body name: Venus (299) {source: DE441}
% Center body name: Solar System Barycenter (0) {source: DE441}
% Center-site name: BODY CENTER
disp(['Body ' num2str(znb,3) ' = Venus 2441784.500000000 = A.D. 1973-Apr-12 00:00:00.0000 TDB'])
% $$$OE
% 2441784.500000000 = A.D. 1973-Apr-12 00:00:00.0000 TDB
X = 9.899077831937718E+07; Y = 4.404482943802425E+07; Z = -5.129735886909358E+06;
VX = -1.419554792364713E+01; VY = 3.190745491599496E+01; VZ = 1.253113692220719E+00;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

znb = znb+1; % Earth
mb(znb) = 398600.435507*1000^3/g; % Rest mass, kg
% *****
% Ephemeris / WWW_USER Tue Aug 22 19:44:20 2023 Pasadena, USA / Horizons
% *****
% Target body name: Earth (399) {source: DE441}
% Center body name: Solar System Barycenter (0) {source: DE441}
% Center-site name: BODY CENTER
disp(['Body ' num2str(znb,3) ' = Earth 2441784.500000000 = A.D. 1973-Apr-12 00:00:00.0000 TDB'])
% $$$OE
% 2441784.500000000 = A.D. 1973-Apr-12 00:00:00.0000 TDB
X = -1.389204904643995E+08; Y = -5.644778869036563E+07; Z = 1.810627236131579E+03;
VX = 1.082927241663490E+01; VY = -2.766430037912495E+01; VZ = -5.234981476416323E-04;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

znb = znb+1; % Earth moon
mb(znb) = 4902.800118*1000^3/g; % Rest mass, kg
% *****
% Ephemeris / WWW_USER Tue Aug 22 19:46:11 2023 Pasadena, USA / Horizons
% *****
```

```

% Target body name: Moon (301)           {source: DE441}
% Center body name: Solar System Barycenter (0)   {source: DE441}
% Center-site name: BODY CENTER
disp(['Body ' num2str(znb,3) ' = Earth Moon      2441784.500000000 = A.D. 1973-Apr-12 00:00:00.0000'])
% $$$SOE
%
% 2441784.500000000 = A.D. 1973-Apr-12 00:00:00.0000 TDB
X = -1.391897271465424E+08; Y = -5.618233758732177E+07; Z = -1.664755858673155E+04;
VX = 1.007864904470174E+01; VY = -2.837411682918930E+01; VZ = -7.814943355182713E-02;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

znb = znb+1; % Mars system
mb(znb) = 42828.375816*1000^3/g; % Rest mass, kg
% *****
% Ephemeris / WWW_USER Tue Aug 22 19:45:24 2023 Pasadena, USA / Horizons
% *****
% Target body name: Mars Barycenter (4)           {source: DE441}
% Center body name: Solar System Barycenter (0)   {source: DE441}
% Center-site name: BODY CENTER
disp(['Body ' num2str(znb,3) ' = Mars system      2441784.500000000 = A.D. 1973-Apr-12 00:00:00.0000 TDB'])
% $$$SOE
%
% 2441784.500000000 = A.D. 1973-Apr-12 00:00:00.0000 TDB
X = 4.440578659011021E+06; Y = -2.166575586321789E+08; Z = -4.655462806515828E+06;
VX = 2.513778145738287E+01; VY = 2.590648909950933E+00; VZ = -5.649448591682305E-01;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

% znb = znb+1; % Vesta
% mb(znb) = 17.28824*1000^3/g; % Rest mass, kg
% *****
% Ephemeris / WWW_USER Thu Feb 23 08:22:43 2023 Pasadena, USA / Horizons
% *****
% Target body name: 4 Vesta (A807 FA)           {source: JPL#36}
% Center body name: Solar System Barycenter (0)   {source: DE441}
% Center-site name: BODY CENTER
% $$$SOE
%
% 2451544.500000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB
% disp(['Body ' num2str(znb,3) ' = Vesta      2451544.500000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB'])
% X = -2.042800442781562E+08; Y = -2.501607714438784E+08; Z = 3.225034236202550E+07;
% VX = 1.663991069548081E+01; VY = -1.287266001826995E+01; VZ = -1.632163374425416E+00;
% rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
% vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
% beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

% znb = znb+1; % Ceres
% mb(znb) = 62.6284*1000^3/g; % Rest mass, kg
% *****
% Ephemeris / WWW_USER Thu Feb 23 08:21:18 2023 Pasadena, USA / Horizons
% *****
% Target body name: 1 Ceres (A801 AA)           {source: JPL#48}
% Center body name: Solar System Barycenter (0)   {source: DE441}
% Center-site name: BODY CENTER
% $$$SOE
%
% 2451544.500000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB
% disp(['Body ' num2str(znb,3) ' = Ceres      2451544.500000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB'])
% X = -3.567415782604431E+08; Y = 1.193768483753398E+08; Z = 6.927039031044590E+07;
% VX = -6.233327667894318E+00; VY = -1.832962513060473E+01; VZ = 5.850329576204976E-01;
% rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
% vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
% beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

% znb = znb+1; % Pallas
% mb(znb) = 13.63*1000^3/g; % Rest mass, kg
% *****
% Ephemeris / WWW_USER Sat Feb 25 17:58:26 2023 Pasadena, USA / Horizons
% *****
% Target body name: 2 Pallas (A802 FA)           {source: JPL#56}
% Center body name: Solar System Barycenter (0)   {source: DE441}
% Center-site name: BODY CENTER
% $$$SOE
%
% 2451544.500000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB
% disp(['Body ' num2str(znb,3) ' = Pallas      2451544.500000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB'])
% X = -1.260221922690224E+08; Y = 2.472855609011113E+08; Z = -1.609056250737203E+08;
% VX = -2.033552571178631E+01; VY = -7.106826784592157E+00; VZ = 6.581658517884729E+00;
% rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
% vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
% beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

% znb = znb+1; % Juno
% mb(znb) = 13.63*(20/205)*1000^3/g; % Rest mass, kg
%
% scaled from Pallas with values from the Asteroid Fact Sheet
% https://nssdc.gsfc.nasa.gov/planetary/factsheet/asteroidfact.html
% *****
% Ephemeris / WWW_USER Mon Feb 27 09:20:57 2023 Pasadena, USA / Horizons
% *****
% Target body name: 3 Juno (A804 RA)           {source: JPL#130}
% Center body name: Solar System Barycenter (0)   {source: DE441}
% Center-site name: BODY CENTER
% $$$SOE
%
% 2451544.500000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB
% disp(['Body ' num2str(znb,3) ' = Juno      2451544.500000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB'])
% X = 5.577481915358917E+07; Y = -4.527662285472839E+08; Z = 1.004303681230918E+08;
% VX = 1.457661204807975E+01; VY = 4.595985058333277E+00; VZ = -1.618387715907128E+00;
% rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
% vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
% beta_option(znb) = 0; % Initial motion input as velocity in m/s

```

```

% %beta_option(znb) = 1; % Initial motion input as v/c

znb = znb+1; % Jupiter system
rb(znb) = 69911 * 1000; % Vol. Mean Radius, m
mb(znb) = 126712764.100000*1000^3/g; % Rest mass, kg
% *****
% Ephemeris / WWW_USER Tue Aug 22 19:40:02 2023 Pasadena, USA / Horizons
% *****
% Target body name: Jupiter Barycenter (5) (source: DE441)
% Center body name: Solar System Barycenter (0) (source: DE441)
% Center-site name: BODY CENTER
disp(['Body ' num2str(znb,3) ' = Jupiter system 2441784.500000000 = A.D. 1973-Apr-12 00:00:00.0000 TDB'])
% $$$OE
% 2441784.500000000 = A.D. 1973-Apr-12 00:00:00.0000 TDB
X = 3.620133265695466E+08; Y = -6.763026434565033E+08; Z = -5.324210149178475E+06;
VX = 1.136504920062007E+01; VY = 6.774769604438296E+00; VZ = -2.825069917736127E-01;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

znb = znb+1; % Saturn system
rb(znb) = 60268 * 1000; % Vol. Mean Radius, m
mb(znb) = 37940584.841800*1000^3/g; % Rest mass, kg
% *****
% Ephemeris / WWW_USER Tue Aug 22 19:41:05 2023 Pasadena, USA / Horizons
% *****
% Target body name: Saturn Barycenter (6) (source: DE441)
% Center body name: Solar System Barycenter (0) (source: DE441)
% Center-site name: BODY CENTER
disp(['Body ' num2str(znb,3) ' = Saturn system 2441784.500000000 = A.D. 1973-Apr-12 00:00:00.0000 TDB'])
% $$$OE
% 2441784.500000000 = A.D. 1973-Apr-12 00:00:00.0000 TDB
X = 1.855715314491706E+08; Y = 1.337100873808510E+09; Z = -3.076321856518877E+07;
VX = -1.009023380547906E+01; VY = 1.308378251082533E+00; VZ = 3.780901591793615E-01;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

znb = znb+1; % Uranus system
mb(znb) = 5794556.400000*1000^3/g; % Rest mass, kg
% *****
% Ephemeris / WWW_USER Tue Aug 22 19:47:10 2023 Pasadena, USA / Horizons
% *****
% Target body name: Uranus Barycenter (7) (source: DE441)
% Center body name: Solar System Barycenter (0) (source: DE441)
% Center-site name: BODY CENTER
disp(['Body ' num2str(znb,3) ' = Uranus system 2441784.500000000 = A.D. 1973-Apr-12 00:00:00.0000 TDB'])
% $$$OE
% 2441784.500000000 = A.D. 1973-Apr-12 00:00:00.0000 TDB
X = -2.562407243202391E+09; Y = -1.004122687805839E+09; Z = 2.952422209238613E+07;
VX = 2.434037190296007E+00; VY = -6.658769675415064E+00; VZ = -5.634471425889309E-02;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

znb = znb+1; % Neptune system
mb(znb) = 6836527.100580*1000^3/g; % Rest mass, kg
% *****
% Ephemeris / WWW_USER Tue Aug 22 19:48:44 2023 Pasadena, USA / Horizons
% *****
% Target body name: Neptune Barycenter (8) (source: DE441)
% Center body name: Solar System Barycenter (0) (source: DE441)
% Center-site name: BODY CENTER
disp(['Body ' num2str(znb,3) ' = Neptun system 2441784.500000000 = A.D. 1973-Apr-12 00:00:00.0000 TDB'])
% $$$OE
% 2441784.500000000 = A.D. 1973-Apr-12 00:00:00.0000 TDB
X = -1.832073500526248E+09; Y = -4.145810902005812E+09; Z = 1.275740007378149E+08;
VX = 4.937034316444592E+00; VY = -2.166055589456510E+00; VZ = -6.913592005881630E-02;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

znb = znb+1;
% Pluto system
mb(znb) = 975.500000*1000^3/g; % Rest mass, kg
% *****
% Ephemeris / WWW_USER Tue Aug 22 19:49:28 2023 Pasadena, USA / Horizons
% *****
% Target body name: Pluto Barycenter (9) (source: DE441)
% Center body name: Solar System Barycenter (0) (source: DE441)
% Center-site name: BODY CENTER
disp(['Body ' num2str(znb,3) ' = Pluto system 2441784.500000000 = A.D. 1973-Apr-12 00:00:00.0000 TDB'])
% $$$OE
% 2441784.500000000 = A.D. 1973-Apr-12 00:00:00.0000 TDB
X = -4.452528258427497E+09; Y = -2.769938240345988E+08; Z = 1.317499118896319E+09;
VX = 1.248298414008540E+00; VY = -5.745971070772510E+00; VZ = 2.539741557301916E-01;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

znb = znb+1;
% Pioneer 10
mb(znb) = 258.0^3/g; % Rest mass, kg
% *****
% Ephemeris / WWW_USER Tue Aug 22 19:52:55 2023 Pasadena, USA / Horizons

```

```

% *****
% Target body name: Pioneer 10 (spacecraft) (-23) {source: pioneer_10}
% Center body name: Solar System Barycenter (0) {source: DE441}
% Center-site name: BODY CENTER
disp(['Body ' num2str(znb,3) ' = Pioneer 10      2441784.500000000 = A.D. 1973-Apr-12 00:00:00.0000 TDB'])
% $$SOE
%
%                               2441784.500000000 = A.D. 1973-Apr-12 00:00:00.0000 TDB
X = 3.317142987235690E+08; Y =-4.923234621413164E+08; Z =-1.359181375527397E+07;
VX= 1.346622496900181E+01; VY=-2.560216346249997E+00; VZ= 5.684940805444272E-02;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

znb = znb+1;
% Pioneer 11
mb(znb) = 259.0^3/g; % Rest mass, kg
% *****
% Ephemeris / WWW_USER Tue Aug 22 19:53:39 2023 Pasadena, USA / Horizons
% *****
% Target body name: Pioneer 11 (spacecraft) (-24) {source: pioneer_11}
% Center body name: Solar System Barycenter (0) {source: DE441}
% Center-site name: BODY CENTER
disp(['Body ' num2str(znb,3) ' = Pioneer 11      2441784.500000000 = A.D. 1973-Apr-12 00:00:00.0000 TDB'])
% $$SOE
%
%                               2441784.500000000 = A.D. 1973-Apr-12 00:00:00.0000 TDB
X =-1.374802933398370E+08; Y =-6.089699986224020E+07; Z =-1.051510652365230E+06;
VX= 1.363907160668941E+01; VY=-3.635778202528599E+01; VZ=-2.071518963072091E+00;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

```

Data File: zSS_2000_01_01_PS_downloaded_on_2023_02_23.m

```
% zSS_2000_01_01_PS_downloaded_on_2023_02_23.m
%
% Planetary System Masses are from
% The JPL Planetary and Lunar Ephemerides DE440 and DE441 Table 2
% https://iopscience.iop.org/article/10.3847/1538-3881/abd414
% https://ssd.jpl.nasa.gov/astro_par.html
% The Planetary and Lunar Ephemerides DE430 and DE431 Table 12 - Use this source for Pallas and Juno
% http://ipnpr.jpl.nasa.gov/progress_report/42-196/196C.pdf
%
% Initial position and velocity relative to the Solar System Barycenter are from
% https://ssd.jpl.nasa.gov/horizons/app.html#/

disp(['Solar System bodies for January 1, 2000 with planetary system velocity and mass'])
disp(['Ephemeris / WWW_USER Thu Feb 23 08:00:55 2023 Pasadena, USA / Horizons'])
g = 0.66743e-10 % Newton's universal gravitational constant. Units = m^3/(kg-s)

znb = 0;

znb = znb+1; % Sun
rb(znb) = 696500 * 1000; % Radius (photosphere), m
mb(znb) = 132712440041.279419*1000^3/g; % Rest mass, kg
%*****
%Ephemeris / WWW_USER Thu Feb 23 08:00:55 2023 Pasadena, USA / Horizons
%*****
%Target body name: Sun (10) {source: DE441}
%Center body name: Solar System Barycenter (0) {source: DE441}
%Center-site name: BODY CENTER
%$$SOE
%
2451544.500000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB
disp(['Body ' num2str(znb,3) ' = Sun 2451544.500000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB'])
X = -1.068108951496322E+06; Y = -4.177210908491462E+05; Z = 3.086887010002915E+04;
VX = 9.305302656256911E-03; VY = -1.283177282717393E-02; VZ = -1.631700118015769E-04;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

znb = znb+1; % Mercury
mb(znb) = 22031.868551*1000^3/g; % Rest mass, kg
%*****
%Ephemeris / WWW_USER Thu Feb 23 08:03:59 2023 Pasadena, USA / Horizons
%*****
%Target body name: Mercury (199) {source: DE441}
%Center body name: Solar System Barycenter (0) {source: DE441}
%Center-site name: BODY CENTER
%$$SOE
%
2451544.500000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB
disp(['Body ' num2str(znb,3) ' = Mercury 2451544.500000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB'])
X = -2.212073002393702E+07; Y = -6.682435921338345E+07; Z = -3.461577076477692E+06;
VX = 3.666229234452722E+01; VY = -1.230266984222893E+01; VZ = -4.368336206255391E+00;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

znb = znb+1; % Venus
mb(znb) = 324858.592*1000^3/g; % Rest mass, kg
%*****
%Ephemeris / WWW_USER Thu Feb 23 08:06:49 2023 Pasadena, USA / Horizons
%*****
%Target body name: Venus (299) {source: DE441}
%Center body name: Solar System Barycenter (0) {source: DE441}
%Center-site name: BODY CENTER
%$$SOE
%
2451544.500000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB
disp(['Body ' num2str(znb,3) ' = Venus 2451544.500000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB'])
X = -1.085736592234813E+08; Y = -3.784241757371509E+06; Z = 6.190088659339075E+06;
VX = 8.984650886248794E-01; VY = -3.517203951420625E+01; VZ = -5.320225928762774E-01;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

znb = znb+1; % Earth
mb(znb) = 398600.435507*1000^3/g; % Rest mass, kg
%*****
%Ephemeris / WWW_USER Thu Feb 23 08:08:19 2023 Pasadena, USA / Horizons
%*****
%Target body name: Earth (399) {source: DE441}
%Center body name: Solar System Barycenter (0) {source: DE441}
%Center-site name: BODY CENTER
%$$SOE
%
2451544.500000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB
disp(['Body ' num2str(znb,3) ' = Earth 2451544.500000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB'])
X = -2.627903751048988E+07; Y = 1.445101984929515E+08; Z = 3.025245352813601E+04;
VX = -2.983052803412253E+01; VY = -5.220465675237847E+00; VZ = -1.01485599592612E-04;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

znb = znb+1; % Earth moon
mb(znb) = 4902.800118*1000^3/g; % Rest mass, kg
%*****
%Ephemeris / WWW_USER Thu Feb 23 08:09:00 2023 Pasadena, USA / Horizons
```

```

%*****
%Target body name: Moon (301)           {source: DE441}
%Center body name: Solar System Barycenter (0)   {source: DE441}
%Center-site name: BODY CENTER
%$$$SOE
%
disp(['Body ' num2str(znb,3) ' = Earth Moon      2451544.500000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB
X =-2.659668775178492E+07; Y = 1.442683153167126E+08; Z = 6.680827660505474E+04;
VX=-2.926974096801152E+01; VY=-6.020397935372383E+00; VZ=-1.740818643718001E-03;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

znb = znb+1; % Mars system
mb(znb) = 42828.375816*1000^3/g; % Rest mass, kg
%*****
%Ephemeris / WWW_USER Thu Feb 23 08:11:17 2023 Pasadena, USA / Horizons
%*****
%Target body name: Mars Barycenter (4)           {source: DE441}
%Center body name: Solar System Barycenter (0)   {source: DE441}
%Center-site name: BODY CENTE
%$$$SOE
%
disp(['Body ' num2str(znb,3) ' = Mars system      2451544.500000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB
X = 2.069269460321624E+08; Y =-3.560730804650453E+06; Z =-5.147912373401109E+06;
VX= 1.304308829086448E+00; VY= 2.628158891525453E+01; VZ= 5.188465896537870E-01;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

znb = znb+1; % Vesta
mb(znb) = 17.288245*1000^3/g; % Rest mass, kg
%*****
%Ephemeris / WWW_USER Thu Feb 23 08:22:43 2023 Pasadena, USA / Horizons
%*****
%Target body name: 4 Vesta (A807 FA)           {source: JPL#36}
%Center body name: Solar System Barycenter (0)   {source: DE441}
%Center-site name: BODY CENTER
%$$$SOE
%
disp(['Body ' num2str(znb,3) ' = Vesta            2451544.500000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB
X =-2.042800442781562E+08; Y =-2.501607714438784E+08; Z = 3.225034236202550E+07;
VX= 1.663991069548081E+01; VY=-1.287266001826995E+01; VZ=-1.632163374425416E+00;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

znb = znb+1; % Ceres
mb(znb) = 62.62890*1000^3/g; % Rest mass, kg
%*****
%Ephemeris / WWW_USER Thu Feb 23 08:21:18 2023 Pasadena, USA / Horizons
%*****
%Target body name: 1 Ceres (A801 AA)           {source: JPL#48}
%Center body name: Solar System Barycenter (0)   {source: DE441}
%Center-site name: BODY CENTER
%$$$SOE
%
disp(['Body ' num2str(znb,3) ' = Ceres            2451544.500000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB
X =-3.567415782604431E+08; Y = 1.193768483753398E+08; Z = 6.927039031044590E+07;
VX=-6.233327667894318E+00; VY=-1.832962513060473E+01; VZ= 5.850329576204976E-01;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

znb = znb+1; % Pallas
mb(znb) = 13.923011*1000^3/g; % Rest mass, kg
%*****
%Ephemeris / WWW_USER Sat Feb 25 17:58:26 2023 Pasadena, USA / Horizons
%*****
%Target body name: 2 Pallas (A802 FA)           {source: JPL#56}
%Center body name: Solar System Barycenter (0)   {source: DE441}
%Center-site name: BODY CENTER
%$$$SOE
%
disp(['Body ' num2str(znb,3) ' = Pallas           2451544.500000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB
X =-1.260221922690224E+08; Y = 2.472855609011113E+08; Z =-1.609056250737203E+08;
VX=-2.033552571178631E+01; VY=-7.106826784592157E+00; VZ= 6.581658517884729E+00;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0; % Initial motion input as velocity in m/s
%beta_option(znb) = 1; % Initial motion input as v/c

znb = znb+1; % Juno
mb(znb) = 1.622415*1000^3/g; % Rest mass, kg
%
scaled from Pallas with values from the Asteroid Fact Sheet
%
https://nssdc.gsfc.nasa.gov/planetary/factsheet/asteroidfact.html
%*****
%Ephemeris / WWW_USER Mon Feb 27 09:20:57 2023 Pasadena, USA / Horizons
%*****
%Target body name: 3 Juno (A804 RA)           {source: JPL#130}
%Center body name: Solar System Barycenter (0)   {source: DE441}
%Center-site name: BODY CENTER
%$$$SOE
%
disp(['Body ' num2str(znb,3) ' = Juno            2451544.500000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB
X = 5.577481915358917E+07; Y =-4.527662285472839E+08; Z = 1.004303681230918E+08;
VX= 1.457661204807975E+01; VY= 4.59598505833277E+00; VZ=-1.618387715907128E+00;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;

```

```

beta_option(znb) = 0;    % Initial motion input as velocity in m/s
%beta_option(znb) = 1;    % Initial motion input as v/c

znb = znb+1;    % Jupiter system
mb(znb) = 126712764.100000*1000^3/g;    % Rest mass, kg
%*****
%Ephemeris / WWW USER Thu Feb 23 08:13:59 2023 Pasadena, USA    / Horizons
%*****
%Target body name: Jupiter Barycenter (5)    {source: DE441}
%Center body name: Solar System Barycenter (0)    {source: DE441}
%Center-site name: BODY CENTER
%$$$SOE
%
2451544.5000000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB
disp(['Body ' num2str(znb,3) ' = Jupiter system 2451544.5000000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB'])
X = 5.978409986317567E+08; Y = 4.387049295899953E+08; Z = -1.520164045037362E+07;
VX=-7.892151773052870E+00; VY= 1.114945123278269E+01; VZ= 1.304865206209698E-01;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0;    % Initial motion input as velocity in m/s
%beta_option(znb) = 1;    % Initial motion input as v/c

znb = znb+1;    % Saturn system
mb(znb) = 37940584.841800*1000^3/g;    % Rest mass, kg
%*****
%Ephemeris / WWW USER Thu Feb 23 08:15:57 2023 Pasadena, USA    / Horizons
%*****
%Target body name: Saturn Barycenter (6)    {source: DE441}
%Center body name: Solar System Barycenter (0)    {source: DE441}
%Center-site name: BODY CENTER
%$$$SOE
%
2451544.5000000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB
disp(['Body ' num2str(znb,3) ' = Saturn system 2451544.5000000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB'])
X = 9.576380325384895E+08; Y = 9.821476973356705E+08; Z = -5.518987635363871E+07;
VX=-7.420583253571007E+00; VY= 6.725272206625776E+00; VZ= 1.779635492368175E-01;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0;    % Initial motion input as velocity in m/s
%beta_option(znb) = 1;    % Initial motion input as v/c

znb = znb+1;    % Uranus system
mb(znb) = 5794556.400000*1000^3/g;    % Rest mass, kg
%*****
%Ephemeris / WWW USER Thu Feb 23 08:17:19 2023 Pasadena, USA    / Horizons
%*****
%Target body name: Uranus Barycenter (7)    {source: DE441}
%Center body name: Solar System Barycenter (0)    {source: DE441}
%Center-site name: BODY CENTER
%$$$SOE
%
2451544.5000000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB
disp(['Body ' num2str(znb,3) ' = Uranus system 2451544.5000000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB'])
X = 2.157706581099027E+09; Y = -2.055242873665452E+09; Z = -3.559276745493364E+07;
VX= 4.646804612202742E+00; VY= 4.614387361015829E+00; VZ=-4.306282947090589E-02;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0;    % Initial motion input as velocity in m/s
%beta_option(znb) = 1;    % Initial motion input as v/c

znb = znb+1;    % Neptune system
mb(znb) = 6836527.100580*1000^3/g;    % Rest mass, kg
%*****
%Ephemeris / WWW USER Thu Feb 23 08:18:37 2023 Pasadena, USA    / Horizons
%*****
%Target body name: Neptune Barycenter (8)    {source: DE441}
%Center body name: Solar System Barycenter (0)    {source: DE441}
%Center-site name: BODY CENTER
%$$$SOE
%
2451544.5000000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB
disp(['Body ' num2str(znb,3) ' = Neptun system 2451544.5000000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB'])
X = 2.513785389044664E+09; Y = -3.739265139739250E+09; Z = 1.907031472560143E+07;
VX= 4.475372369556783E+00; VY= 3.063567642389680E+00; VZ=-1.662255108032890E-01;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0;    % Initial motion input as velocity in m/s
%beta_option(znb) = 1;    % Initial motion input as v/c

znb = znb+1;    % Pluto system
mb(znb) = 975.500000*1000^3/g;    % Rest mass, kg
%*****
%Ephemeris / WWW USER Thu Feb 23 08:20:31 2023 Pasadena, USA    / Horizons
%*****
%Target body name: Pluto Barycenter (9)    {source: DE441}
%Center body name: Solar System Barycenter (0)    {source: DE441}
%Center-site name: BODY CENTER
%$$$SOE
%
2451544.5000000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB
disp(['Body ' num2str(znb,3) ' = Pluto system 2451544.5000000000 = A.D. 2000-Jan-01 00:00:00.0000 TDB'])
X = -1.478626320473919E+09; Y = -4.182879133477592E+09; Z = 8.752983914521730E+08;
VX= 5.253372152934933E+00; VY=-2.675694808207576E+00; VZ=-1.233257158207600E+00;
rx(1,znb) = X*1000; ry(1,znb) = Y*1000; rz(1,znb) = Z*1000;
vx(1,znb) = VX*1000; vy(1,znb) = VY*1000; vz(1,znb) = VZ*1000;
beta_option(znb) = 0;    % Initial motion input as velocity in m/s
%beta_option(znb) = 1;    % Initial motion input as v/c

```

Appendix C

The following information is extracted from [Solar System Simulator – Technical Notes](#)
You can download the full report at <https://github.com/nicokuijpers/SolarSystemSimulator>

Solar System Simulator – Technical Notes

Author: Nico Kuijpers

Date: December 4, 2025

Introduction

The Solar System Simulator is written in Java. Positions and velocities of planets, moons, small solar system bodies, and spacecraft are continuously updated using either Newton Mechanics, General Relativity (PPN method) or the Curvature of Wave Propagation Method (CWPM). Initial positions and velocities are obtained from Nasa JPL's ephemerides. While simulating, both simulated positions and ephemeris data are visualised for comparison. Source code is made publicly available under the MIT licence.

Simulation

Position and velocity of each particle is updated each time step using a Runge-Kutta scheme (PPN and CWPM, time step 1 hour) or Adams-Bashforth-Moulton scheme (Newton Mechanics, time step 0.5 hour). Computing acceleration requires interaction between all pairs of particles, and therefore, the simulation is computationally expensive. To save computation time, smaller bodies such as comets and spacecraft do not apply forces to other particles. PPN or CWPM may be applied to obtain even more accurate simulation results. It should be noted that PPN requires much more computational effort compared to Newton Mechanics. However, computational effort required by CWPM is only a factor 3 compared to Newton Mechanics and is 5 times as fast as PPN with similar accuracy.

In case you want to run longer lasting simulations (say more than a century), it is possible to run the simulation without a graphical user interface and store intermediate results. See `CreateSolarSystemStateFiles.java` for example code. The files can be loaded into the simulator for visual inspection and/or continuation of the simulation.

Both the Runge-Kutta scheme and the Adams-Bashforth-Moulton scheme allow advancing with a negative timestep. Thus, it is also possible to simulate backward in time.

Ephemerides

Initial positions and velocities of the Solar System bodies are needed to run a simulation. In the simulator, ephemeris data is continuously computed such that simulation data (blue orbit lines) and ephemeris data (green orbit lines) can be compared at all times. Ephemerides data is computed such that the ecliptic plane corresponds to the x-y plane. In the Earth-to-Sun viewing mode, the ecliptic would be horizontal, whereas in reality the ecliptic plane would be tilted due to the fact that the Earth's axis is tilted. This should be noted when comparing a simulated Venus transit with drawings or pictures taken during the event.

Ephemerides for Sun, Moon, and major planets (including Pluto)

For dates between January 1, 1600 and January 31, 2200, DE405 is used. For other dates between 3000 BC and AD 3000, an approximate ephemeris is used. Ephemerides data for the

Moon before January 1, 1600 or after January 31, 2200 is approximated using DE405 ephemeris data (see `EphemerisSolarSystem.java` for the code).

.

.

.

Simulation accuracy

To obtain some insight in the accuracy of the simulation results, the major planets, the Sun, the Moon, and the Pluto system were simulated for 600 years. The simulation was started at January 1, 1600. During the simulation, the deviation in position compared to the DE405 Ephemeris was calculated each day. Average deviation was calculated each year. As may be expected, the deviation steadily increased over the years. In Table 2, the average deviation for each body is shown for the 100th simulated year and in Table 3, the average deviation for each body is shown for the 600th simulated year. It can be observed that using PPN or CWPM leads to more accurate results for the inner planets, whereas for the outer planets the difference is relatively small. The results shown in Tables 2 and 3 were obtained by running `SimulationAccuracyExperiment.java`. Computation times were 36 s for Newton Mechanics, 461 s for PPN, and 94 s for CWPM on a MacBook Pro with 2,6 GHz 6-Core Intel Core i7 processor.

Body	Newton Mechanics	General Relativity	CWPM
Mercury	15696 km	2 km	70 km
Venus	9229 km	2 km	4 km
Earth	5917 km	45 km	50 km
Moon	5923 km	46 km	52 km
Mars	2397 km	172 km	176 km
Jupiter	316 km	94 km	93 km
Saturn	25 km	137 km	137 km
Uranus	108 km	55 km	55 km
Neptune	34 km	73 km	72 km
Pluto System	55 km	43 km	43 km

Table 2: Deviation in position after 100 years of simulation.

Body	Newton Mechanics	General Relativity	CWPM
Mercury	96487 km	173 km	144 km
Venus	55471 km	8 km	26 km
Earth	35650 km	273 km	300 km
Moon	35642 km	2159 km	2114 km
Mars	15792 km	868 km	898 km
Jupiter	2034 km	636 km	630 km
Saturn	238 km	809 km	805 km
Uranus	724 km	373 km	374 km
Neptune	284 km	497 km	489 km
Pluto System	421 km	347 km	347 km

Table 3: Deviation in position after 600 years of simulation.

.

.

.

Precession of the perihelion of Mercury

The orbit of Mercury is not only affected by the gravitational forces of the other planets, but also by the fact that spacetime is disturbed by the Sun's mass. This leads to the precession of the perihelion of Mercury. In the Solar System Simulator, this effect can be observed when

simulating with General Relativity (PPN) or Curvature of Wave Propagation Method (CWPM) and comparing the results to a simulation with Newton Mechanics.

Two experiments have been defined to investigate the precession of the perihelion of Mercury. In one experiment, a two-particle system with the Sun and Mercury is simulated with Newton Mechanics, General Relativity, and CWPM for one hundred, one thousand, and ten thousand years (`MercuryPrecessionTwoParticleExperiment.java`). In another experiment, the entire Solar System is simulated for one hundred and one thousand years (`MercuryPrecessionSolarSystemExperiment.java`).

Perihelion precessions measured in the experiments are listed in Table 11. Observed precession of the perihelion of Mercury is 574.10 ± 0.65 arcsec/Julian century. Of the observed precession, 532.30 arcsec/century can be explained by gravitational pull of other bodies. For more information, see https://en.wikipedia.org/wiki/Tests_of_general_relativity.

Experiment	Duration	Newton Mechanics	General Relativity	CWPM
Two particles	100 years	0.1244	42.8942	42.8942
Two particles	1000 years	0.0088	42.9873	42.9873
Two particles	10000 years	0.0024	42.9830	42.9830
Solar system	100 years	527.46	570.38	570.38
Solar system	1000 years	533.64	576.49	576.49

Table 11: Precession of the perihelion of Mercury.

.
.
.

The following raw solution results are taken from: <https://github.com/nicokuijpers/SolarSystemSimulator/blob/master/src/main/java/experiments/SimulationAccuracyExperiment.java>

```
Running simulation using Newton Mechanics for 600 years
Year: 100 body: Mercury average deviation: 15696.344408740277 km
Year: 100 body: Venus average deviation: 9228.146075506156 km
Year: 100 body: Earth average deviation: 5916.612366864066 km
Year: 100 body: Moon average deviation: 5922.7420981041405 km
Year: 100 body: Mars average deviation: 2396.826905150947 km
Year: 100 body: Jupiter average deviation: 316.09479720192513 km
Year: 100 body: Saturn average deviation: 25.182541181543954 km
Year: 100 body: Uranus average deviation: 108.43526053082994 km
Year: 100 body: Neptune average deviation: 34.47117068562416 km
Year: 100 body: Pluto System average deviation: 55.14515805115282 km

Year: 200 body: Mercury average deviation: 31412.838970582376 km
Year: 200 body: Venus average deviation: 18421.432710679255 km
Year: 200 body: Earth average deviation: 11863.689599155057 km
Year: 200 body: Moon average deviation: 11871.83302058216 km
Year: 200 body: Mars average deviation: 4372.196872023489 km
Year: 200 body: Jupiter average deviation: 700.9589053923255 km
Year: 200 body: Saturn average deviation: 79.59395322377586 km
Year: 200 body: Uranus average deviation: 249.82190974105814 km
Year: 200 body: Neptune average deviation: 64.57753326455845 km
Year: 200 body: Pluto System average deviation: 163.98136337132297 km

Year: 300 body: Mercury average deviation: 47244.76944377595 km
Year: 300 body: Venus average deviation: 27787.07067992593 km
Year: 300 body: Earth average deviation: 17811.061061810804 km
Year: 300 body: Moon average deviation: 17817.826290829653 km
Year: 300 body: Mars average deviation: 6303.0617907412225 km
Year: 300 body: Jupiter average deviation: 855.7191210767357 km
Year: 300 body: Saturn average deviation: 65.72469669606453 km
Year: 300 body: Uranus average deviation: 387.11924427800545 km
Year: 300 body: Neptune average deviation: 148.16586248524357 km
Year: 300 body: Pluto System average deviation: 122.40852621087913 km

Year: 400 body: Mercury average deviation: 63326.31933716076 km
Year: 400 body: Venus average deviation: 36906.22324212632 km
Year: 400 body: Earth average deviation: 23757.319045108747 km
Year: 400 body: Moon average deviation: 23759.63999602375 km
```

Year: 400 body: Mars average deviation: 8842.541485263371 km
 Year: 400 body: Jupiter average deviation: 1450.9931557426466 km
 Year: 400 body: Saturn average deviation: 187.26296402207436 km
 Year: 400 body: Uranus average deviation: 486.7255721417548 km
 Year: 400 body: Neptune average deviation: 155.4683958456844 km
 Year: 400 body: Pluto System average deviation: 395.17673800677 km

 Year: 500 body: Mercury average deviation: 79734.46112997102 km
 Year: 500 body: Venus average deviation: 46291.40222342086 km
 Year: 500 body: Earth average deviation: 29705.498485231463 km
 Year: 500 body: Moon average deviation: 29702.642004979953 km
 Year: 500 body: Mars average deviation: 12205.582108296192 km
 Year: 500 body: Jupiter average deviation: 1431.2870059853458 km
 Year: 500 body: Saturn average deviation: 199.52213657731124 km
 Year: 500 body: Uranus average deviation: 581.699817023832 km
 Year: 500 body: Neptune average deviation: 238.25674047009062 km
 Year: 500 body: Pluto System average deviation: 231.75938519706827 km

 Year: 600 body: Mercury average deviation: 96487.19403592826 km
 Year: 600 body: Venus average deviation: 55470.825460066946 km
 Year: 600 body: Earth average deviation: 35650.3457572258 km
 Year: 600 body: Moon average deviation: 35641.69785507082 km
 Year: 600 body: Mars average deviation: 15791.579923819347 km
 Year: 600 body: Jupiter average deviation: 2033.8688697596333 km
 Year: 600 body: Saturn average deviation: 237.8991623604191 km
 Year: 600 body: Uranus average deviation: 723.5117772827939 km
 Year: 600 body: Neptune average deviation: 284.47840830165484 km
 Year: 600 body: Pluto System average deviation: 420.86375878060113 km
 Computation time Newton Mechanics : 36 s

 Running simulation using General Relativity for 600 years
 Year: 100 body: Mercury average deviation: 2.2160851425307246 km
 Year: 100 body: Venus average deviation: 1.6864158183041007 km
 Year: 100 body: Earth average deviation: 45.34697429913651 km
 Year: 100 body: Moon average deviation: 46.22053263098865 km
 Year: 100 body: Mars average deviation: 171.52349325056505 km
 Year: 100 body: Jupiter average deviation: 93.58293964033635 km
 Year: 100 body: Saturn average deviation: 137.27024487345452 km
 Year: 100 body: Uranus average deviation: 54.68083021595303 km
 Year: 100 body: Neptune average deviation: 73.20649981680536 km
 Year: 100 body: Pluto System average deviation: 43.650260944872805 km

 Year: 200 body: Mercury average deviation: 9.029623188774496 km
 Year: 200 body: Venus average deviation: 2.393530536751434 km
 Year: 200 body: Earth average deviation: 91.04458659847317 km
 Year: 200 body: Moon average deviation: 139.11680212169503 km
 Year: 200 body: Mars average deviation: 330.7015288375295 km
 Year: 200 body: Jupiter average deviation: 206.8820929492279 km
 Year: 200 body: Saturn average deviation: 319.53755048126027 km
 Year: 200 body: Uranus average deviation: 126.12072768464519 km
 Year: 200 body: Neptune average deviation: 121.18428635286209 km
 Year: 200 body: Pluto System average deviation: 146.25767797785446 km

 Year: 300 body: Mercury average deviation: 31.562177553581137 km
 Year: 300 body: Venus average deviation: 3.504136148261649 km
 Year: 300 body: Earth average deviation: 136.89304172046081 km
 Year: 300 body: Moon average deviation: 417.57202207169206 km
 Year: 300 body: Mars average deviation: 445.6642375682609 km
 Year: 300 body: Jupiter average deviation: 294.38966791024853 km
 Year: 300 body: Saturn average deviation: 413.0272246877849 km
 Year: 300 body: Uranus average deviation: 201.2507782264417 km
 Year: 300 body: Neptune average deviation: 257.1146797956943 km
 Year: 300 body: Pluto System average deviation: 112.89116007300497 km

 Year: 400 body: Mercury average deviation: 66.43096516972626 km
 Year: 400 body: Venus average deviation: 5.314413609489632 km
 Year: 400 body: Earth average deviation: 182.7379386220461 km
 Year: 400 body: Moon average deviation: 848.4580255543099 km
 Year: 400 body: Mars average deviation: 569.46897183995 km
 Year: 400 body: Jupiter average deviation: 423.636284038079 km
 Year: 400 body: Saturn average deviation: 607.3622384743568 km
 Year: 400 body: Uranus average deviation: 259.65169472135165 km
 Year: 400 body: Neptune average deviation: 286.0299313650391 km
 Year: 400 body: Pluto System average deviation: 323.38891998483734 km

 Year: 500 body: Mercury average deviation: 113.68586141294976 km
 Year: 500 body: Venus average deviation: 6.758164378218149 km
 Year: 500 body: Earth average deviation: 228.12955605878182 km
 Year: 500 body: Moon average deviation: 1422.7893536417675 km
 Year: 500 body: Mars average deviation: 732.5966358850064 km
 Year: 500 body: Jupiter average deviation: 500.4461049736107 km
 Year: 500 body: Saturn average deviation: 729.4851825742157 km
 Year: 500 body: Uranus average deviation: 310.1904652871179 km
 Year: 500 body: Neptune average deviation: 404.5642674027708 km
 Year: 500 body: Pluto System average deviation: 224.90006533057098 km

 Year: 600 body: Mercury average deviation: 173.4693348251751 km
 Year: 600 body: Venus average deviation: 7.712660570977729 km
 Year: 600 body: Earth average deviation: 273.067076327504 km
 Year: 600 body: Moon average deviation: 2158.52360762621 km
 Year: 600 body: Mars average deviation: 867.6119567994816 km
 Year: 600 body: Jupiter average deviation: 635.7367234863823 km
 Year: 600 body: Saturn average deviation: 808.6094386616495 km
 Year: 600 body: Uranus average deviation: 372.66956595628983 km
 Year: 600 body: Neptune average deviation: 496.530300102512 km
 Year: 600 body: Pluto System average deviation: 346.72740611799645 km
 Computation time General Relativity : 461 s

 Running simulation using Curvature of Wave Propagation Method (CWPM) for 600 years
 Year: 100 body: Mercury average deviation: 69.39295411325449 km

```

Year: 100 body: Venus average deviation: 3.7377374585809218 km
Year: 100 body: Earth average deviation: 49.70854995509628 km
Year: 100 body: Moon average deviation: 51.57942171469133 km
Year: 100 body: Mars average deviation: 176.23300009916647 km
Year: 100 body: Jupiter average deviation: 92.68439995408697 km
Year: 100 body: Saturn average deviation: 136.64043180308371 km
Year: 100 body: Uranus average deviation: 54.81094303604363 km
Year: 100 body: Neptune average deviation: 71.60031084725304 km
Year: 100 body: Pluto System average deviation: 42.82416118834642 km

Year: 200 body: Mercury average deviation: 91.58419955889808 km
Year: 200 body: Venus average deviation: 8.633020686340213 km
Year: 200 body: Earth average deviation: 100.02657868094153 km
Year: 200 body: Moon average deviation: 131.75228379075136 km
Year: 200 body: Mars average deviation: 339.70505164916545 km
Year: 200 body: Jupiter average deviation: 204.79002447306684 km
Year: 200 body: Saturn average deviation: 318.0960196961095 km
Year: 200 body: Uranus average deviation: 125.92665626691256 km
Year: 200 body: Neptune average deviation: 120.4725925489998 km
Year: 200 body: Pluto System average deviation: 145.95108725268562 km

Year: 300 body: Mercury average deviation: 122.9032713303424 km
Year: 300 body: Venus average deviation: 13.102855261561555 km
Year: 300 body: Earth average deviation: 150.33962453582208 km
Year: 300 body: Moon average deviation: 397.69921706922247 km
Year: 300 body: Mars average deviation: 459.22834908348665 km
Year: 300 body: Jupiter average deviation: 291.90630166823007 km
Year: 300 body: Saturn average deviation: 411.4434297576111 km
Year: 300 body: Uranus average deviation: 200.70741394687005 km
Year: 300 body: Neptune average deviation: 253.1600289897645 km
Year: 300 body: Pluto System average deviation: 114.05444361087015 km

Year: 400 body: Mercury average deviation: 150.56627650533093 km
Year: 400 body: Venus average deviation: 16.85208046690599 km
Year: 400 body: Earth average deviation: 200.60126637403917 km
Year: 400 body: Moon average deviation: 819.7501831581628 km
Year: 400 body: Mars average deviation: 587.9228597715771 km
Year: 400 body: Jupiter average deviation: 419.2951147647167 km
Year: 400 body: Saturn average deviation: 604.3943678385083 km
Year: 400 body: Uranus average deviation: 259.6175352071738 km
Year: 400 body: Neptune average deviation: 283.2222948398999 km
Year: 400 body: Pluto System average deviation: 320.26828218956246 km

Year: 500 body: Mercury average deviation: 132.7710272584092 km
Year: 500 body: Venus average deviation: 21.030145940220347 km
Year: 500 body: Earth average deviation: 250.56306526672265 km
Year: 500 body: Moon average deviation: 1385.8622336101607 km
Year: 500 body: Mars average deviation: 756.6372265996325 km
Year: 500 body: Jupiter average deviation: 496.1290612152379 km
Year: 500 body: Saturn average deviation: 726.5233584306108 km
Year: 500 body: Uranus average deviation: 311.2789437563982 km
Year: 500 body: Neptune average deviation: 400.8705324181377 km
Year: 500 body: Pluto System average deviation: 227.21669479939436 km

Year: 600 body: Mercury average deviation: 143.6523531802757 km
Year: 600 body: Venus average deviation: 25.603293823387503 km
Year: 600 body: Earth average deviation: 299.7956081887951 km
Year: 600 body: Moon average deviation: 2113.96050077713 km
Year: 600 body: Mars average deviation: 897.6909726757672 km
Year: 600 body: Jupiter average deviation: 629.6477415351101 km
Year: 600 body: Saturn average deviation: 804.9792291816616 km
Year: 600 body: Uranus average deviation: 374.17289177904286 km
Year: 600 body: Neptune average deviation: 489.17118210969545 km
Year: 600 body: Pluto System average deviation: 345.6900101200245 km
Computation time Curvature of Wave Propagation Method : 94 s

```