The Curvature of Motion in a Gravitational Field

Morris G. Anderson
email: morrisganderson@gmail.com

Abstract

MatLab script, with example files, for numerically integrating: Orbital motion; Orbital Precession; Natural Frequency Shift; Light and Cosmic Ray Deflection; Shapiro Time Delay; Apparent Black Hole Emissivity and Shadow Diameter in a Gravitational Field. This is accomplished by applying a simple and efficient curvature of motion method (based on wave-particle duality) to calculate the influence of a static gravitational field on the path of motion and the natural frequency of matter. The associated math is easy to understand and is limited to basic geometry, vector algebra, and simple calculus. This enables advanced high school and undergraduate college students to understand and solve these examples on a mobile phone or tablet with an app capable of running MatLab script files. Comparisons with data illustrate excellent agreement for both week and strong gravitational fields.

The MatLab TMG_solver.m and TMG_path.m script files apply a simple curvature of motion method (based on wave-particle duality) to calculate, by numerical integration, the influence of a static gravitational field on the path of motion and the natural frequency of matter. **Comparisons with data illustrate excellent agreement in both week and strong gravitational fields.** This is accomplished with the Wave Propagation Curvature of Motion method derived in the book, Time, Matter, and Gravity, (TMG) by Morris G. Anderson, copyright 2004 [1].

The wave-particle duality approach makes it possible to efficiently calculate the curvature of motion in a gravitational field. **This enables solving the examples presented in this paper on a mobile phone or tablet with an application capable of running MatLab script files.** Applying this method could help high school and undergraduate college students understand how wave propagation governs the motion of matter in a gravitational field.

The associated math is easy to understand and is limited to basic geometry, vector algebra, and simple calculus. The path of motion is numerically integrated by calculating the curvature of motion with TMG Equation 7-12 (derived based on wave propagation), in three-dimensional Euclidean Space. Numerical convergence is obtained with the Newton-Raphson method. The user verifies overall convergence by increasing the number of arc segments until changes in the result become asymptotic.

Example Script files are provided to calculate the following solutions:

Example 1: Simple trajectory motion.
Example 2: Earth satellite motion and clock rate frequency shift.
Example 3: Orbital motion and precession of planets in our solar system.
Example 4: Orbital motion, precession, and redshift of Star S2 in the galactic center.
Example 5: Strong gravitational field orbital precession.
Example 6: The curved path and Shapiro time delay of light passing near the Sun.
Example 7: The gravitational bending of light and cosmic rays passing near a neutron star.
Example 8: The emissivity, shadow diameter, and curved path of light escaping from or passing near an apparent black hole or quasar. This is done for Sgr A* at the galactic center and the central gravitational body M87*.

Results from these example solutions agree closely with data illustrating that the TMG Method enables calculating the natural frequency and the curvature of motion of all forms of matter (particle or wave) in a gravitational field based on wave propagation.

Note: This does not appear to be possible with general relativity.

"Therefore, it does not appear possible to introduce in a natural way the hypothesis of wave-particle duality into the framework of general relativity."
Mashhoon, B.: Wave propagation in a gravitational field. Phys. Lett. A 122, number 6, 7 page 299-304 (1987) [2]

Example 8 provides a prediction of the size of the apparent black hole shadow. Based on the numerical results presented in this paper, the size of the black hole shadow is $e2r_s$ where e is the base of the natural logarithm and r_s is the Schwarzschild radius. This result has been previously and independently verified by Stanley L. Robertson [3]. The resulting value (which agrees very well with data) is approximately 4.6% larger than the general relativity prediction [4] of $(3\sqrt{3} \div 2)2r_s$.

The ability to calculate the diameter of an apparent black hole shadow with these script files indicates the black hole concept that is based on general relativity could be misleading. It may be better described as an "Apparent Black Hole". This observation is based on the fact that the general relativity gravitational redshift is a first-order approximation of a natural logarithmic solution (see the discussion accompanying Example 8).

The TMG derivation also illustrates that the null result of the Michelson–Morley experiment stems from the wave properties of matter. It explains why the standard speed of light (defined as the change in position of light divided by its change in position as measured with a standard rod and clock) is independent of motion and position.

The book, Time, Matter, and Gravity, provides a full derivation of this method.

The equations applied in the MatLab script files provided herein were derived independently by the author. However, the natural logarithmic solution has also been postulated, derived, and/or utilized by others to simulate motion in a gravitational field [5], [6].

Numerical Approach For Calculating The Path Of Motion

Figure 1 illustrates the curvature of motion of matter in a gravitational field. This is derived from the wavelength, λ, of the object and the properties of the space through which it is changing position. The important variables are the velocity of the object, the speed of light, and the gradient of the speed of light in the gravitational field.

Note: The speed of light is a function of position. However, (as demonstrated by derivation in the book Time, Matter, and Gravity) the length of a measuring rod and the natural frequency of a clock are both functions of the square root of the speed of light. Hence, the standard speed of light (defined as the change in position of light divided by its change in position when measured with a standard rod and clock) is independent of position and velocity.

The Curvature of Motion illustrated in Figure 1 is valid for all forms of matter in both week and strong gravitational fields.

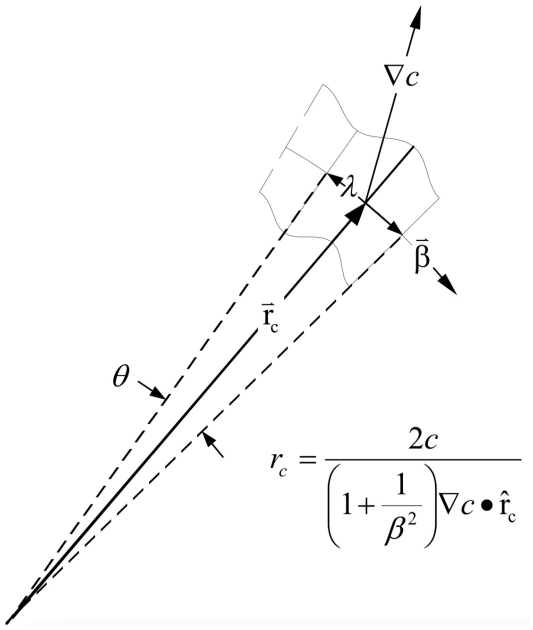


Figure 1 Curvature of motion derived from wave propagation and expressed as TMG equation 7-9

For a single non-spinning governing body, this reduces to the following (see TMG equation 7-12)

$$r_c = \frac{c_s^2 r^2}{(1 + \frac{1}{\beta^2})MG(\hat{r} \cdot \hat{r}_c)}$$

The Matlab script TMG_solver.m drives Matlab function TMG_path.m to calculate an object's path of motion by numerical methods. This is only valid for an object of constant matter (independent of other forces such as drag, magnetic, etc.) along the path of motion in the gravitational field of a single static governing body. However, the TMG_path function could be modified to include the influence of external forces and multiple governing bodies in motion.

According to the TMG derivation, the natural frequency of matter (and therefore of a clock) is a function of position and motion in a gravitational field. This influence is accounted for in Matlab function TMG_path.m by calculating the frequency shift with TMG equation 5-29.

This script utilizes Vectors in a Cartesian coordinate system to keep track of the math. The speed of light, as a function of position in a gravitational field, is calculated with TMG equation (4-12) in the form of TMG equation (4-14). The overall path is numerically integrated by dividing the total path into small arcs, calculating the radius of curvature for these segments with TMG equation (7-12), and adding together the individual arc lengths.

Figure 2 presents a particle that is moving through space along a curved path. Starting with the direction and curvature of motion at point 1, the direction of motion at point 2 is estimated by rotating the curvature vector, r_{c1}, through the angle α. The radius of curvature at point 2, r_{c2}, is then calculated and compared with r_{c1}. If there is a significant difference, the averaged value is used to re-calculate the location of point 2. Once the location and direction of motion of point 2 are known with sufficient accuracy, the process is repeated to calculate the next position along the path.

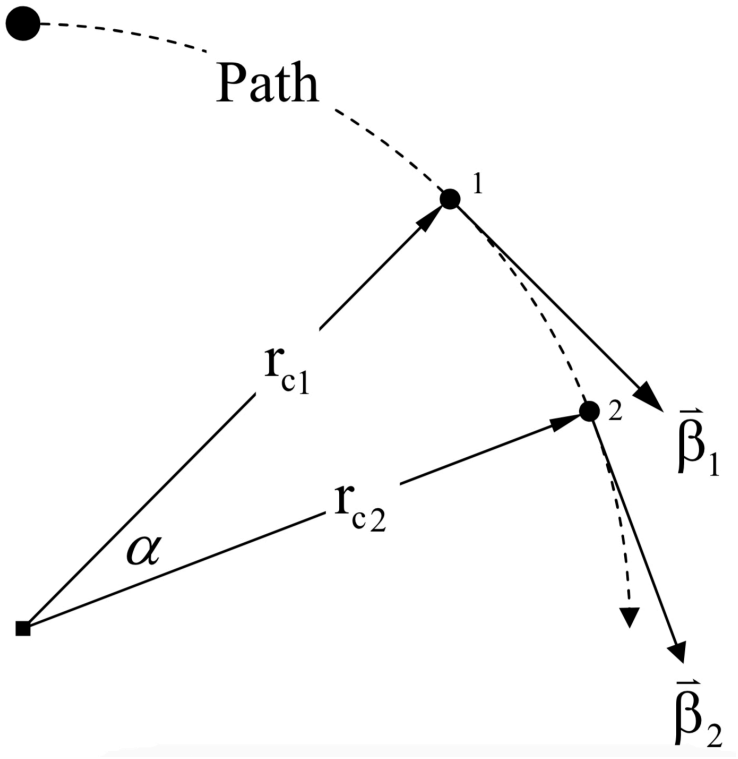


Figure 2 The radius of curvature, based on equation (7-12), is used to integrate the path

Example Script Files and Solutions

These MatLab script files are attached at the end of this paper. Each includes comments documenting the process. The equation derivations are provided in the book Time, Matter, and Gravity [1].

Example_1_Simple_Trajectory.m

Example_2_Earth_satellites.m This file calls the following to calculate results.
- Example_2a_input_for_INTELSAT_39.m
- Example_2b_input_for_MOLNIYA_1_36.m
- Example_2c_input_for_NTS_2.m
- Example_2d_input_for_ISS.m

Example_3_Solar_system.m This file calls the following to calculate results.
- Example_3a_input_for_Mercury.m
- Example_3b_input_for_Venus.m
- Example_3c_input_for_Earth.m
- Example_3d_input_for_Mars.m
- Example_3e_input_for_Jupiter.m
- Example_3f_input_for_Saturn.m
- Example_3g_input_for_Uranus.m
- Example_3h_input_for_Neptune.m
- Example_3i_input_for_Pluto.m

Example_4_S2_galactic_center.m

Example_5_strong_gravity_field_precession.m

Example_6_Sunlight_deflection_Shapiro_time_delay.m

Example_7_Neutron_star_light_deflection.m

Example_8_Apparent_Black_Hole_Shadow_Diameter_and_Emissivity.m

TMG_solver.m	This file drives the solution and is called by the example script file.
TMG_path.m	This file calculates the path of motion and is called by the TMG_solver.m file.

These files can be downloaded at: https://www.mathworks.com/matlabcentral/fileexchange/94835-the-curvature-of-motion-in-a-gravitational-field

Solutions can be obtained with:

MatLab	https://www.mathworks.com
GNU Octave	https://www.gnu.org/software/octave/index
SIMO	https://doc.simo.com.hk

All of the solutions presented in this document were calculated on an iPad Air (3rd generation) with the SIMO MATLAB Programming App version 2.20.0 (269D).

Example 1 Simple Trajectory Motion

Script file: Example_1_Simple_Trajectory.m

Figure 3 illustrates that the path of motion calculated with the TMG_path.m script is indistinguishable from Newtonian Mechanics for a simple trajectory at low velocities.

Initial velocity: vx = 20.00 m/s, vy = 34.64 m/s, vz = 0.000 m/s
Total rotation angle for calculation = 135.0 degrees

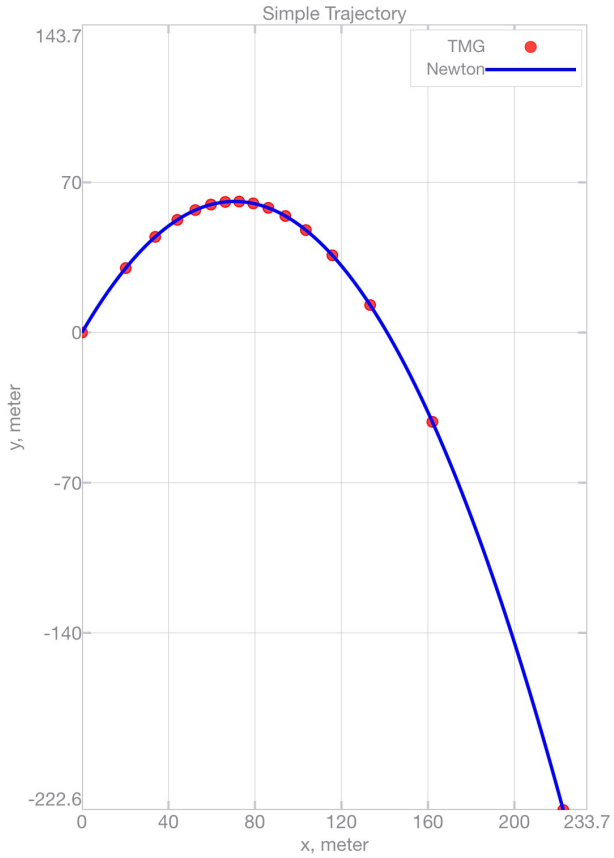


Figure 3 The TMG_path.m script results simulate Newtonian Mechanics at low speed.

Example_1_Simple_Trajectory.m Output

```
--- Inputs ---
Example_1_Simple_Trajectory

--- Outputs ---
Newtons G = 6.67430e-11 m^3/(kg-s),  Standard speed of light = 299792458.0 m/s

Earth mass = 5.97217e24 kg,  Earth volumetric mean radius = 6371010.00 m

Reference speed of light  = 299792458.000 m/s at 6371010.0 meters from the Earths center of mass

Initial Position x = 0.00000000000000 y = 6371010.00000000 z = 0.00000000000000 meters

Initial velocity,  vx = 20.00 m/s, vy = 34.64 m/s, vz = 0.000 m/s

Total rotation angle for calculation = 135.0 degrees

Number of solution points saved in memory for plotting = 15.00

Number of arc segments calculated for each point       = 30.00

*** Calculating a simple trajectory ***

>>> Calculating path with 450.0 equal arc angle segments

>>> Solution complete for path_option 3

Final Position x = 222.575627086640 y = 6370787.41647279 z = 0.00000000000000 meters

Euclidian distance between beginning and end = 314.775056758135 meters

Path length = 432.359144450279 meters

Path   time = 11.1287331500200 seconds

Angle between beginning and ending directions = 135.0 degrees

Trajectory radius: Min = 6370787.42036084, Max = 6371071.04439759 meters

Plot results relative to starting point

Plot is generated.

Plot is held.

Plot is generated.

Plot is unheld.

Legend is shown.
```

Example 2 Earth Satellites

Script file: Example_2_Earth_satellites.m

This example illustrates the orbital motion and clock rate shift of four man-made Earth Satellites. The satellite atomic clock rate shift is caused by the influence of motion and gravity on the natural frequency of matter. The calculated orbits match Keplerian elements, and the theoretical clock rate fractional frequency shift is within data uncertainty.

Figure 4 illustrates the orbits as viewed from above the equatorial plane. The side view is presented in Figure 5. Earth is illustrated by the blue circle. The international space station by the yellow line with the other satellites as indicated in the caption of each figure. The solid symbol identifies the orbit Perigee, the open symbol indicates the Apogee. The dotted portion of the orbit is below the equatorial plane, the solid portion is above.

Figure 6 presents a curve of the calculated fractional frequency shift with orbital averaged results for the satellites indicated. Measured data for the NTS-2 satellite is plotted illustrating close agreement [7].

The initial satellite Two-line element set data for this example was obtained from https://celestrak.com

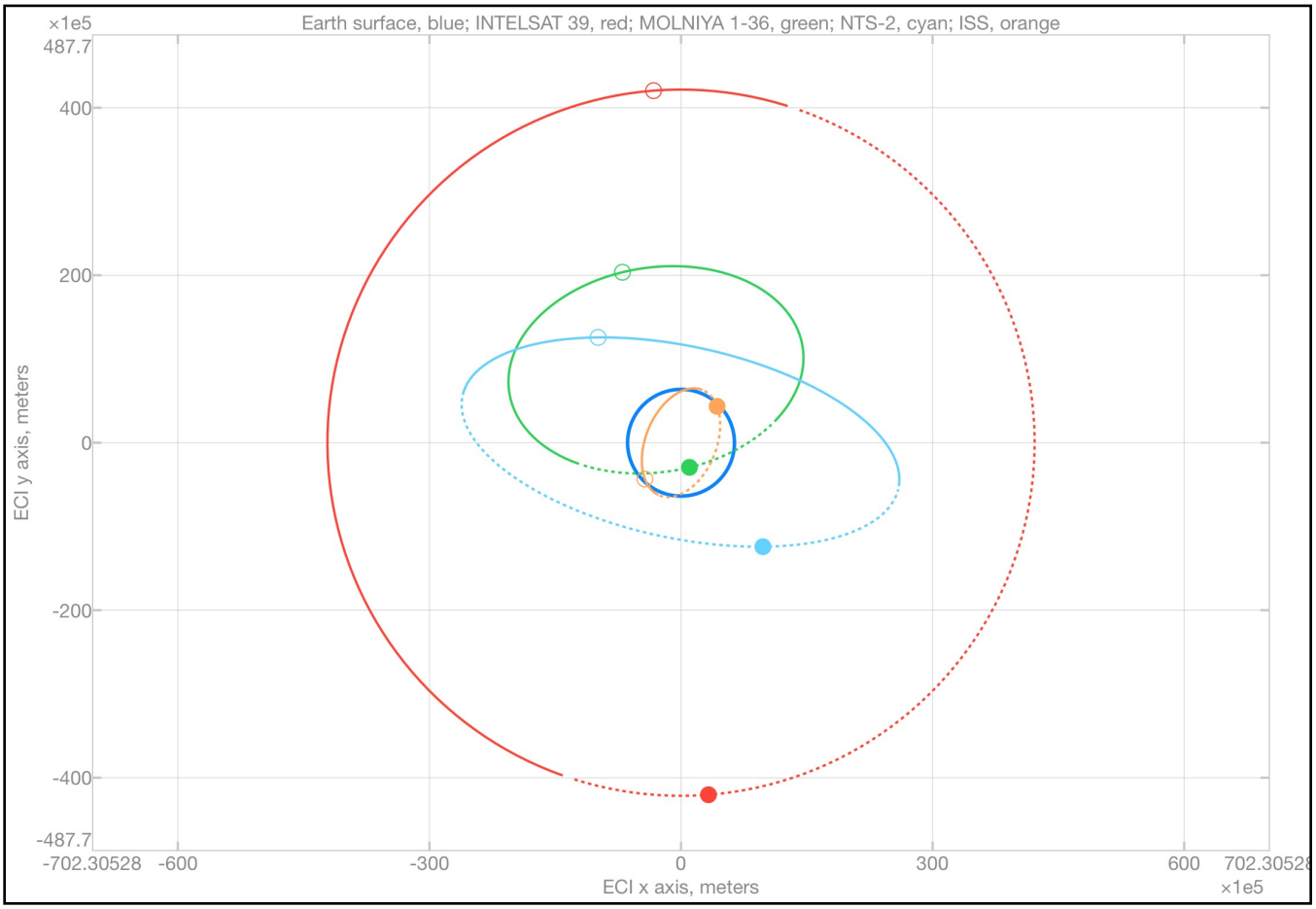


Figure 4 Earth satellite orbits looking down on the equatorial plane from North.

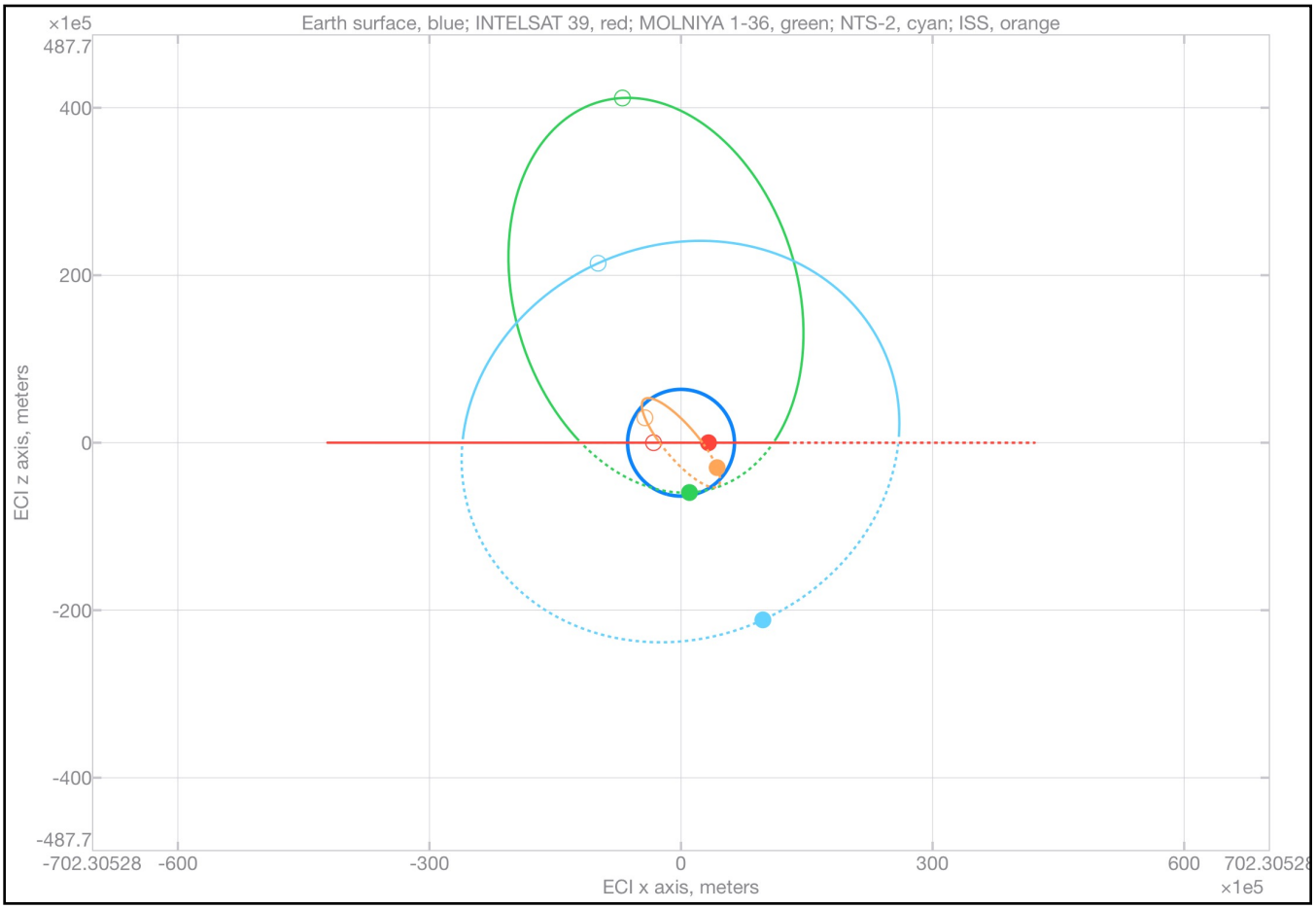


Figure 5 Earth satellite orbits viewed parallel to the equatorial plane

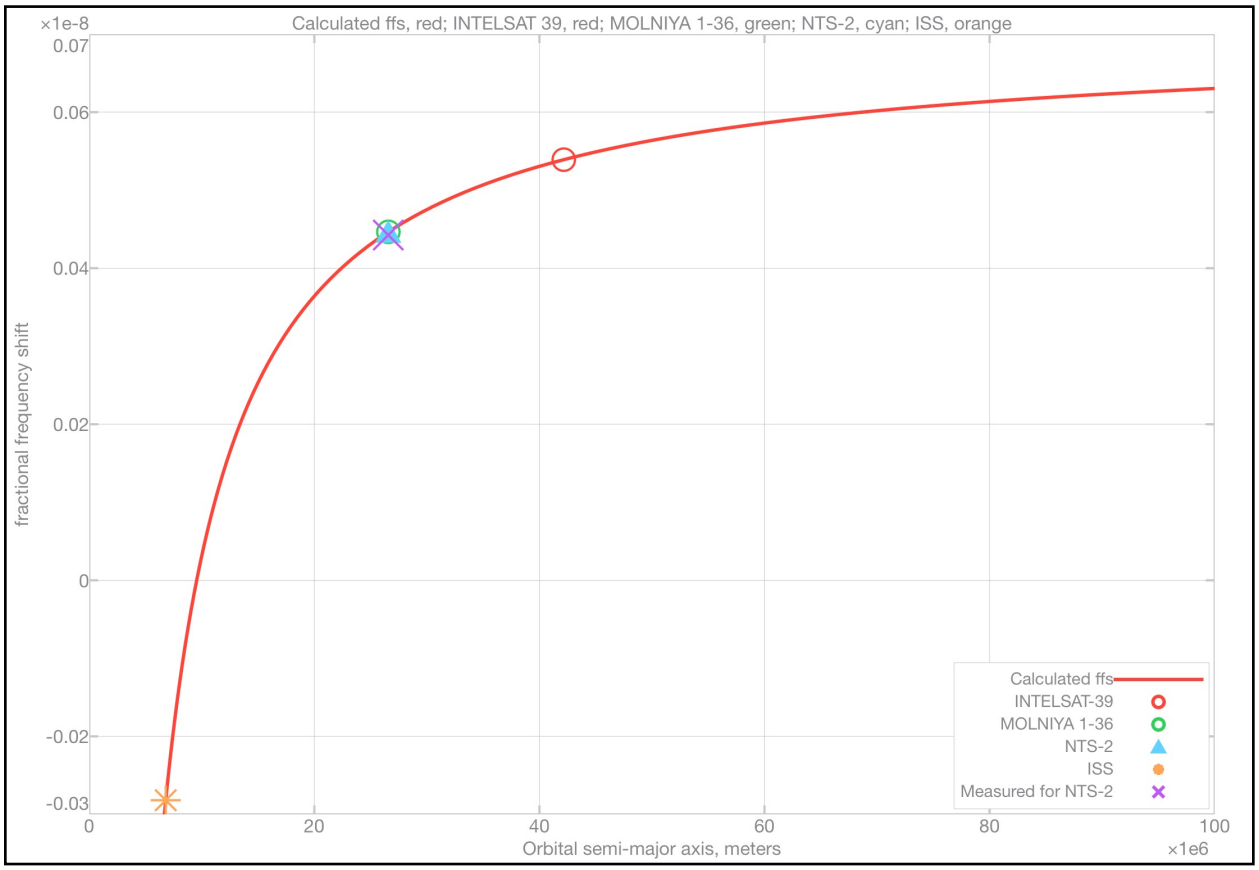


Figure 6 Calculated clock rate fractional frequency shift matches test data.

Example_2_Earth_satellites.m Output

```
--- Inputs ---
Example_2_Earth_satellites

--- Outputs ---
Newtons universal gravitational constant, G = 6.67430e-11 m^3/(kg-s)
The standard speed of light, cs = 299792458.0 m/s
Earth mass = 5.97217e24 kg,  Earth volumetric mean radius = 6371010.00 m
Reference speed of light  = 299792458.000 m/s at 6371010.0 meters from the Earths center of mass

------
This example calculates the orbital motion of the INTELSAT_39 satellite
Initial conditions taken from https://celestrak.com NORAD Two-Line Element Sets 2020-08-07
Position   x = 3.27425e6 m,   y = -4.20338e7 m,  z = -2532.68 m
Velocity  vx = 3065.65 m/s,  vy = 238.801 m/s,  vz = -4.34655e-1 m/s
Beta    vx/c = 1.02259e-5, vy/c = 7.96554e-7, vz/c = -1.44985e-9

--- Calculating INTELSAT_39 orbit ---
Total rotation angle for calculation = 360.0 degrees
Calculating solution with 180.0 points and 10.00 arc segments per point
>>> Calculating path with 1800 equal arc angle segments
>>> Solution complete for path_option 3

--- INTELSAT_39 Orbital Solution Results -----
Orbital Period = 1436.14 minutes, delta vs TLE = -1.064e-4 seconds
Circumference  = 264934236.877 meters
Perigee        = 42161107.0934 meters, delta vs TLE = -2.980e-8 meters
Apogee         = 42170079.9317 meters, delta vs TLE = -1.937e-6 meters
Eccentricity   = 1.0640000e-4, delta vs TLE = -2.275e-14
df/f           = 5.391e-10 fractional frequency shift relative to the USNO master clock

------
This example calculates the orbital motion of the MOLNIYA_1_36 satellite
Initial conditions taken from https://celestrak.com NORAD Two-Line Element Sets 2020-08-08
Position   x = 1.00893e6 m,   y = -2.94091e6 m,  z = -5.94687e6 m
Velocity  vx = 9886.65 m/s,  vy = 2412.58 m/s,  vz = 484.238 m/s
Beta    vx/c = 3.29783e-5, vy/c = 8.04751e-6, vz/c = 1.61525e-6

--- Calculating MOLNIYA_1_36 orbit ---
Total rotation angle for calculation = 360.0 degrees
Calculating solution with 180.0 points and 10.00 arc segments per point
>>> Calculating path with 1800 equal arc angle segments
>>> Solution complete for path_option 3

--- MOLNIYA_1_36 Orbital Solution Results -----
Orbital Period = 718.761 minutes, delta vs TLE = -2.225e-2 seconds
Circumference  = 140385387.879 meters
Perigee        = 6710600.14123 meters, delta vs TLE = 0.000 meters
Apogee         = 46448761.9331 meters, delta vs TLE = -32.86 meters
Eccentricity   = 7.4752894e-1, delta vs TLE = -1.561e-7
df/f           = 4.466e-10 fractional frequency shift relative to the USNO master clock

------
This example calculates the orbital motion of the NTS_2 satellite
Initial conditions taken from https://celestrak.com NORAD Two-Line Element Sets 2020-08-10
Position   x = 9.76189e6 m,   y = -1.24143e7 m,  z = -2.11532e7 m
Velocity  vx = 3547.77 m/s,  vy = 37.3210 m/s,  vz = 1615.34 m/s
Beta    vx/c = 1.18341e-5, vy/c = 1.24489e-7, vz/c = 5.38820e-6

--- Calculating NTS_2 orbit ---
Total rotation angle for calculation = 360.0 degrees
Calculating solution with 180.0 points and 10.00 arc segments per point
>>> Calculating path with 1800 equal arc angle segments
>>> Solution complete for path_option 3

--- NTS_2 Orbital Solution Results -----
Orbital Period = 718.387 minutes, delta vs TLE = -5.182e-5 seconds
Circumference  = 166945384.281 meters
Perigee        = 26398224.5535 meters, delta vs TLE = -3.725e-9 meters
Apogee         = 26742699.9640 meters, delta vs TLE = -4.535e-3 meters
Eccentricity   = 6.4822999e-3, delta vs TLE = -8.479e-11
df/f           = 4.465e-10 fractional frequency shift relative to the USNO master clock

------
This example calculates the orbital motion of the ISS satellite
Initial conditions taken from https://celestrak.com NORAD Two-Line Element Sets 2020-08-10
```

```
Position   x = 4.30421e6 m,   y = 4.33622e6 m,  z = -2.97661e6 m
Velocity  vx = -2042.03 m/s,  vy = 5446.89 m/s,  vz = 4982.04 m/s
Beta    vx/c = -6.81146e-6, vy/c = 1.81689e-5, vz/c = 1.66183e-5

--- Calculating ISS orbit ---
Total rotation angle for calculation = 360.0 degrees
Calculating solution with 180.0 points and 10.00 arc segments per point
>>> Calculating path with 1800 equal arc angle segments
>>> Solution complete for path_option 3

--- ISS Orbital Solution Results -----
Orbital Period = 92.9538 minutes, delta vs TLE = -2.305e-6 seconds
Circumference  = 42708866.8714 meters
Perigee        = 6796260.81761 meters, delta vs TLE = -6.519e-9 meters
Apogee         = 6798393.81892 meters, delta vs TLE = -6.771e-7 meters
Eccentricity   = 1.5690000e-4, delta vs TLE = -4.936e-14
df/f           = -2.819e-10 fractional frequency shift relative to the USNO master clock
```

Example 3 Solar system

Script file: Example_3_Solar_system.m

In this example, we apply Keplerian Mean Orbital Elements from NASA fact sheets (https://nssdc.gsfc.nasa.gov/planetary/planetfact.html) to calculate the orbit of each planet in the solar system. In doing so, we consider only the gravitational field of the Sun, ignoring the influence of other solar system matter. We also calculate the orbital precession caused by the Sun's gravitational field.

Figure 7 illustrates the orbits of the inner planets as viewed from above the ecliptic plane (upper plot) and from the side (lower plot). 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.

Figure 8 presents similar results for the outer planets.

Figure 9 illustrates the convergence history

Table 1 Orbital precession results are in close agreement with data.

Orbital precession caused by the gravitational field of the Sun		
Planet	arcseconds / century	
	Calculated	Observed
Mercury	42.97	42.56 +- 0.94
Venus	8.61	8.4 +- 4.8
Earth	3.83	4.6 +- 2.7
Mars	1.34	1.5 +- 0.04

Calculated values are averaged for solutions with 32 000, 64 000, 128 000, 256 000, and 512 000 arc segments.
For observed values, see V. M. Blanco and S. W. McCuskey (1961) Basic Physics Of The Solar System page 217 [8]

The results presented in Table 1 represent converged solutions as illustrated in Figure 9. Convergence is achieved with about 10 000.0 arc segments for Earth, Venus, and Mars. Around 30 000.0 iterations are required for Mercury predicting that the Sun's gravitational field causes a precession of about 42.97 arc seconds per century in close agreement with observations.

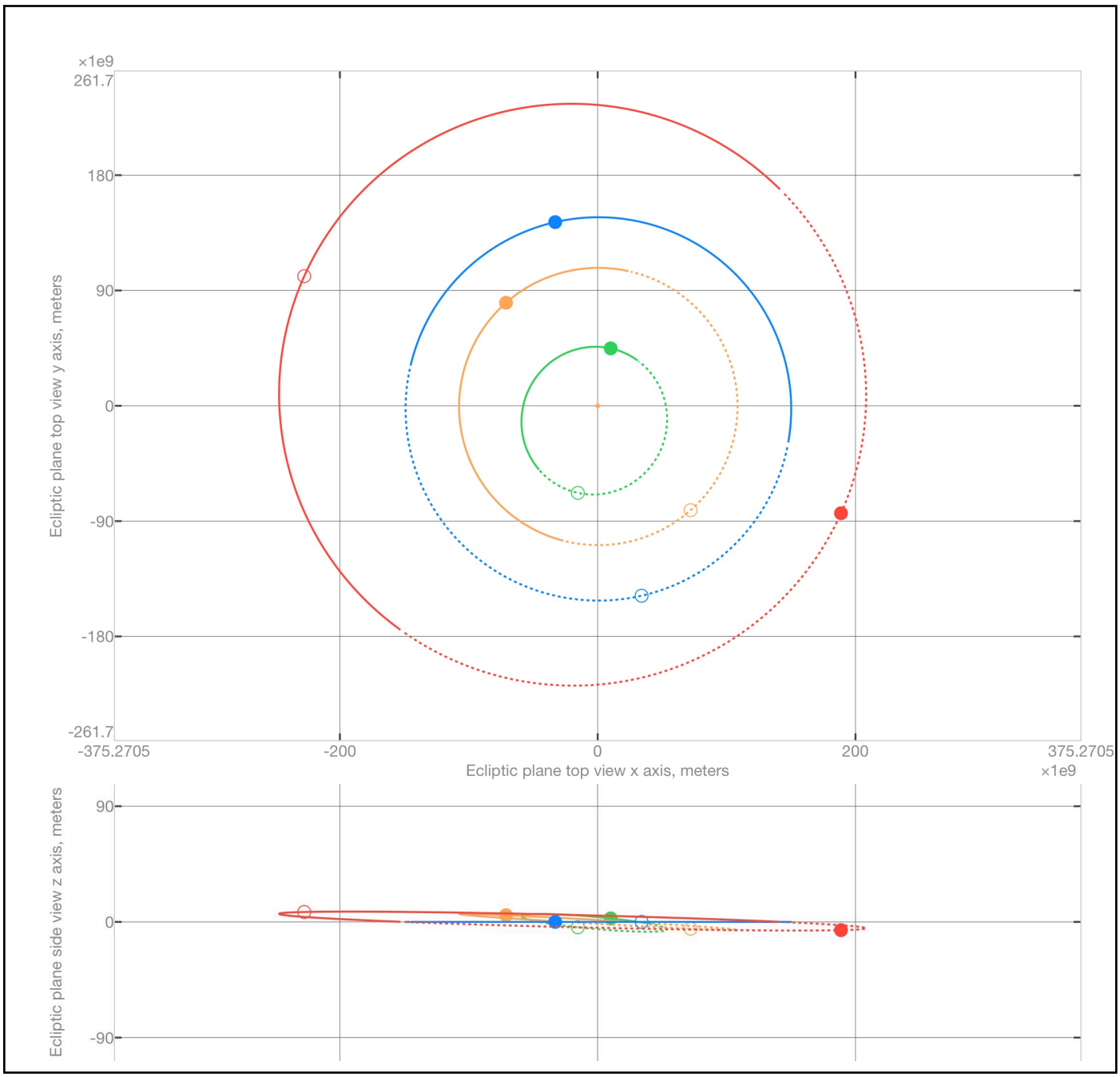


Figure 7 Orbits of Mercury, Venus, Earth, and Mars
Upper plot illustrates orbits as viewed from above the ecliptic plane
Lower plot illustrates orbits as viewed parallel to the ecliptic plane,

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.

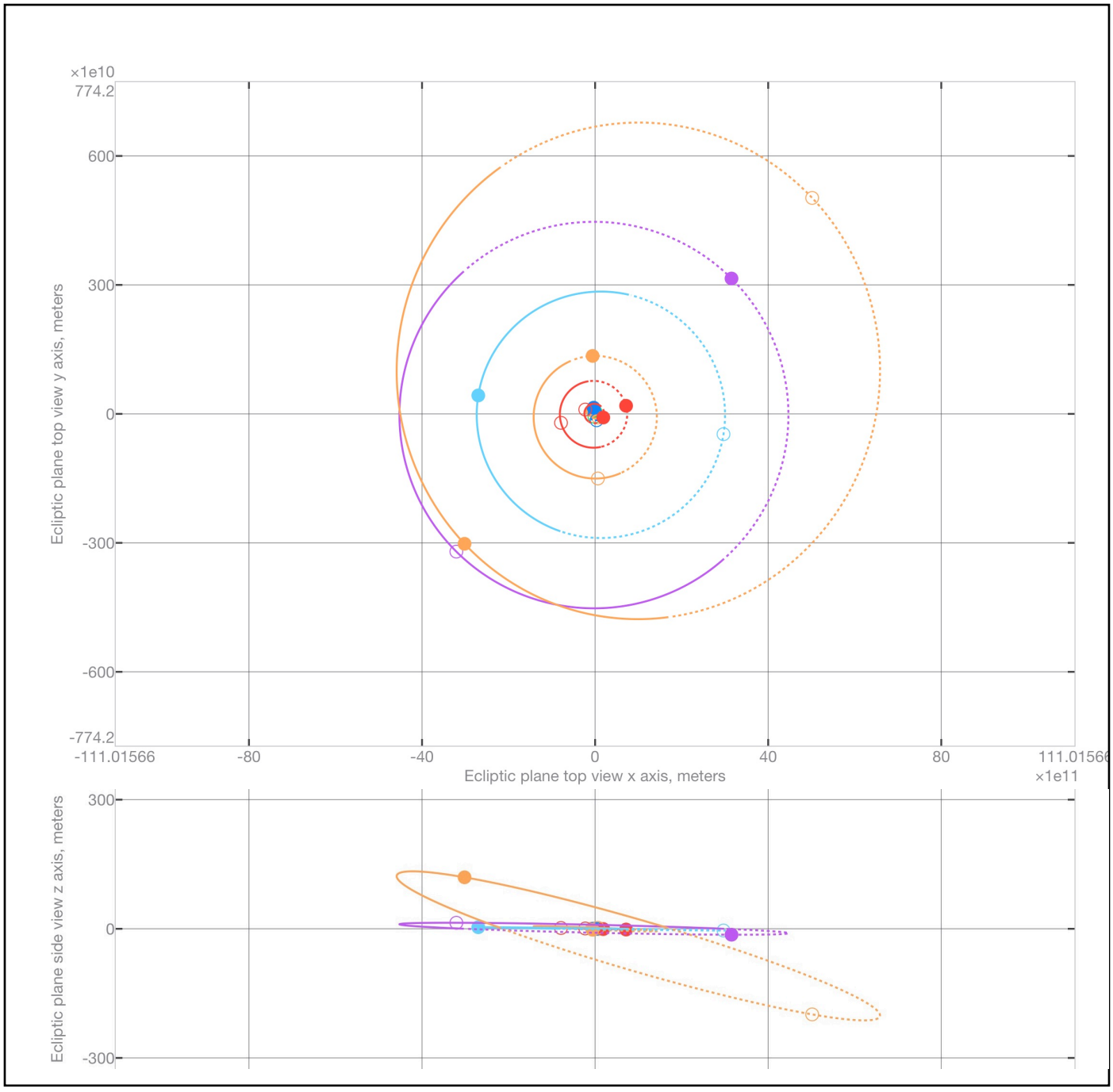


Figure 8 The outer orbits in this plot are Jupiter, Saturn, Uranus, Neptune, and Pluto
Upper plot illustrates orbits as viewed from above the ecliptic plane
Lower plot illustrates orbits as viewed parallel to the ecliptic plane

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.

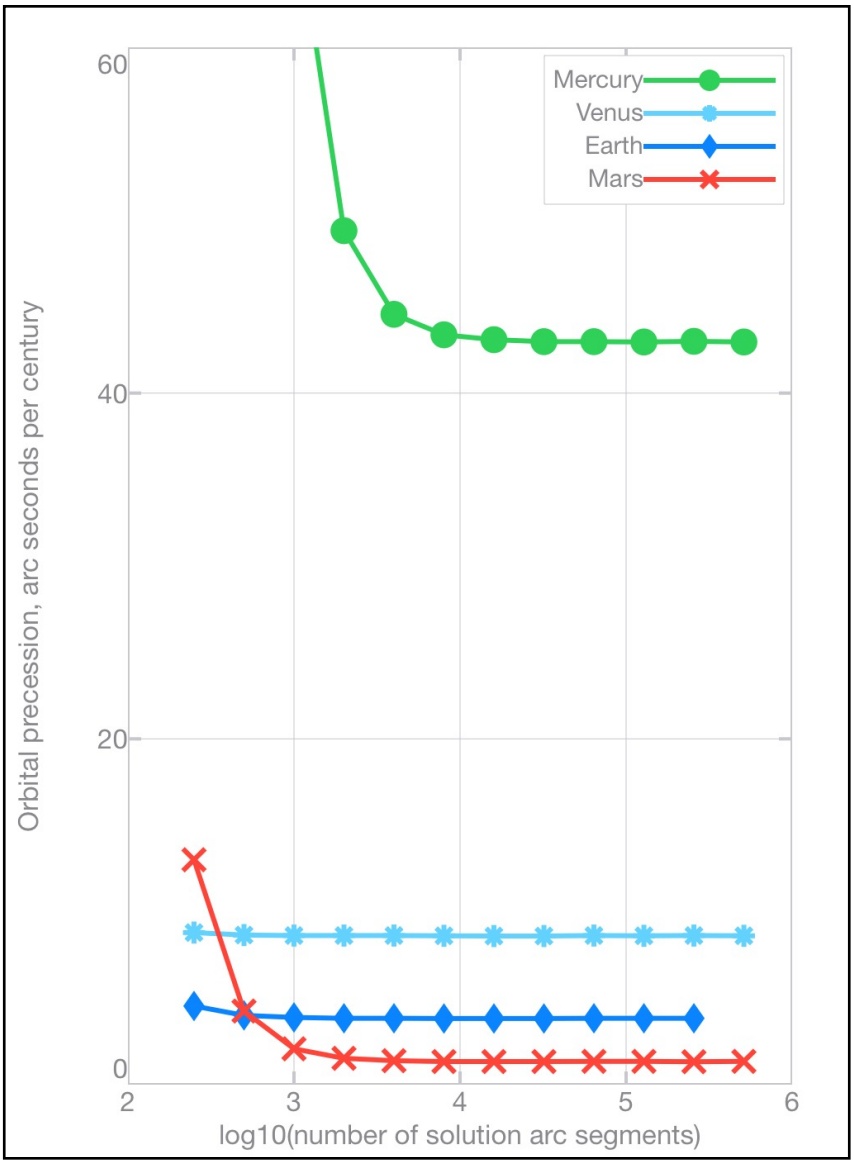


Figure 9 Solar system inner planet precession convergence.

Example_3_Solar_system.m Output

```
--- Inputs ---
Example_3_Solar_system

--- Outputs ---
This example calculates the orbital motion and precession of the Solar system planets.
Newtons universal gravitational constant, G = 6.67430e-11 m^3/(kg-s)
The standard speed of light, cs = 299792458.0 m/s
Earth mass = 5.97217e24 kg,  Earth Volumetric mean radius = 6371010.0 m
Reference speed of light  = 299792458.417 m/s at 1.4959790e11 meters from the Sun (the Earths Semi-
major axis)
Sun mass = 1.98840327824641e30 kg

------
This example calculates the orbital motion of Mercury
Initial conditions taken from https://nssdc.gsfc.nasa.gov/planetary/factsheet/mercuryfact.html
Position   x = 1.01154e10 m,   y = 4.47920e10 m,  z = 2.73046e9 m
Velocity  vx = -57281.4 m/s,  vy = 12552.9 m/s,  vz = 6283.00 m/s,  v = 58976.4 m/s
Beta    vx/c = -1.91070e-4, vy/c = 4.18721e-5, vz/c = 2.09578e-5,   v/c = 1.96724e-4

--- Calculating Mercury orbit ---
Total rotation angle for calculation = 360.0 degrees
Calculating solution with 360.0 points and 1.000 arc segments per point
>>> Calculating path with 360.0 equal arc angle segments
>>> Solution complete for path_option 3

--- Mercury Orbital Solution Results -----
Orbital Period = 87.9689 days, delta vs Kepler = -10.91 seconds
Circumference  = 359975485603 meters
Perihelion     = 46001099056.4 meters, delta vs Kepler = 0.000 meters
Aphelion       = 69816574870.0 meters, delta vs Kepler = -2.422e5 meters
Eccentricity   = 2.0562903e-1, delta vs Kepler = -1.661e-6

--- Calculating the orbital precession of Mercury caused by the gravitational field of the Sun ---
Calculating solution with 32.00 points and 1000 arc segments per point
>>> Calculating orbital precession with 3.200e4 equal arc angle segments
Solving, iteration = 1.000
Solving, iteration = 2.000
Solving, iteration = 3.000
>>> Solution complete for path_option 1

--- Mercury Orbital Precession Solution Results -----
Rotation      = 360.000028771530 degrees
Rotation time = 87.9690082218023 days
Precession    = 2.875e-5 degrees per orbit
Precession    = 42.9753940857042 arcsecond per century

--- For comparison only -----
--- General Relativity predicted value = 42.9814 arcsecond per century


------
This example calculates the orbital motion of Venus
Initial conditions taken from https://nssdc.gsfc.nasa.gov/planetary/factsheet/venusfact.html
Position   x = -7.11119e10 m,   y = 8.04182e10 m,  z = 5.20374e9 m
Velocity  vx = -26359.0 m/s,  vy = -23386.5 m/s,  vz = 1201.92 m/s,  v = 35258.6 m/s
Beta    vx/c = -8.79243e-5, vy/c = -7.80089e-5, vz/c = 4.00919e-6,   v/c = 1.17610e-4

--- Calculating Venus orbit ---
Total rotation angle for calculation = 360.0 degrees
Calculating solution with 360.0 points and 1.000 arc segments per point
>>> Calculating path with 360.0 equal arc angle segments
>>> Solution complete for path_option 3

--- Venus Orbital Solution Results -----
Orbital Period = 224.701 days, delta vs Kepler = -4.139e-2 seconds
Circumference  = 679888254015 meters
Perihelion     = 107475894242 meters, delta vs Kepler = -4.120e-4 meters
Aphelion       = 108941740155 meters, delta vs Kepler = -504.0 meters
Eccentricity   = 6.7732277e-3, delta vs Kepler = -2.313e-9

--- Calculating the orbital precession of Venus caused by the gravitational field of the Sun ---
Calculating solution with 32.00 points and 100.0 arc segments per point
>>> Calculating orbital precession with 3200 equal arc angle segments
Solving, iteration = 1.000
```

```
Solving, iteration = 2.000
Solving, iteration = 3.000
>>> Solution complete for path_option 1

--- Venus Orbital Precession Solution Results -----
Rotation      = 360.000014746344 degrees
Rotation time = 224.701009049524 days
Precession    = 1.466e-5 degrees per orbit
Precession    = 8.58121970954172 arcsecond per century

--- For comparison only -----
--- General Relativity predicted value = 8.62472 arcsecond per century

------
This example calculates the orbital motion of Earth
Initial conditions taken from https://nssdc.gsfc.nasa.gov/planetary/factsheet/earthfact.html
Position   x = -3.29577e10 m,   y = 1.43358e11 m,  z = 117079 m
Velocity  vx = -29516.6 m/s,  vy = -6785.80 m/s,  vz = -1.08376e-2 m/s,  v = 30286.6 m/s
Beta    vx/c = -9.84569e-5, vy/c = -2.26350e-5, vz/c = -3.61503e-11,   v/c = 1.01025e-4

--- Calculating Earth orbit ---
Total rotation angle for calculation = 360.0 degrees
Calculating solution with 360.0 points and 1.000 arc segments per point
>>> Calculating path with 360.0 equal arc angle segments
>>> Solution complete for path_option 3

--- Earth Orbital Solution Results -----
Orbital Period = 365.256 days, delta vs Kepler = -5.182e-1 seconds
Circumference  = 939883566722 meters
Perihelion     = 147097751064 meters, delta vs Kepler = 0.000 meters
Aphelion       = 152097363074 meters, delta vs Kepler = -4241 meters
Eccentricity   = 1.6710206e-2, delta vs Kepler = -1.394e-8

--- Calculating the orbital precession of Earth caused by the gravitational field of the Sun ---
Calculating solution with 32.00 points and 100.0 arc segments per point
>>> Calculating orbital precession with 3200 equal arc angle segments
Solving, iteration = 1.000
Solving, iteration = 2.000
Solving, iteration = 3.000
>>> Solution complete for path_option 1

--- Earth Orbital Precession Solution Results -----
Rotation      = 360.000010676697 degrees
Rotation time = 365.256371473234 days
Precession    = 1.063e-5 degrees per orbit
Precession    = 3.82658253435947 arcsecond per century

--- For comparison only -----
--- General Relativity predicted value = 3.83877 arcsecond per century

------
This example calculates the orbital motion of Mars
Initial conditions taken from https://nssdc.gsfc.nasa.gov/planetary/factsheet/Marsfact.html
Position   x = 1.88764e11 m,   y = -8.38498e10 m,  z = -6.39985e9 m
Velocity  vx = 10763.9 m/s,  vy = 24213.3 m/s,  vz = 242.507 m/s,  v = 26499.1 m/s
Beta    vx/c = 3.59044e-5, vy/c = 8.07668e-5, vz/c = 8.08915e-7,   v/c = 8.83915e-5

--- Calculating Mars orbit ---
Total rotation angle for calculation = 360.0 degrees
Calculating solution with 360.0 points and 1.000 arc segments per point
>>> Calculating path with 360.0 equal arc angle segments
>>> Solution complete for path_option 3

--- Mars Orbital Solution Results -----
Orbital Period = 686.980 days, delta vs Kepler = -19.67 seconds
Circumference  = 1.42906472800e12 meters
Perihelion     = 206648293121 meters, delta vs Kepler = -3.052e-5 meters
Aphelion       = 249233049407 meters, delta vs Kepler = -2.009e5 meters
Eccentricity   = 9.3411931e-2, delta vs Kepler = -3.995e-7


--- Calculating the orbital precession of Mars caused by the gravitational field of the Sun ---
Calculating solution with 32.00 points and 100.0 arc segments per point
>>> Calculating orbital precession with 3200 equal arc angle segments
Solving, iteration = 1.000
Solving, iteration = 2.000
Solving, iteration = 3.000
>>> Solution complete for path_option 1
```

```
--- Mars Orbital Precession Solution Results -----
Rotation      = 360.000007429745 degrees
Rotation time = 686.979934634346 days
Precession    = 7.344e-6 degrees per orbit
Precession    = 1.40577185423101 arcsecond per century

--- For comparison only -----
--- General Relativity predicted value = 1.35093 arcsecond per century

------
This example calculates the orbital motion of Jupiter
Initial conditions taken from https://nssdc.gsfc.nasa.gov/planetary/factsheet/jupiterfact.html
Position   x = 7.15826e11 m,   y = 1.88527e11 m,  z = -1.68215e10 m
Velocity  vx = -3490.73 m/s,  vy = 13256.1 m/s,  vz = 22.8574 m/s,  v = 13708.1 m/s
Beta    vx/c = -1.16438e-5, vy/c = 4.42177e-5, vz/c = 7.62440e-8,   v/c = 4.57252e-5

--- Calculating Jupiter orbit ---
Total rotation angle for calculation = 360.0 degrees
Calculating solution with 360.0 points and 1.000 arc segments per point
>>> Calculating path with 360.0 equal arc angle segments
>>> Solution complete for path_option 3

--- Jupiter Orbital Solution Results -----
Orbital Period = 4332.59 days, delta vs Kepler = -39.64 seconds
Circumference  = 4.88595800208e12 meters
Perihelion     = 740426736405 meters, delta vs Kepler = 0.000 meters
Aphelion       = 815733283336 meters, delta vs Kepler = -1.848e5 meters
Eccentricity   = 4.8392547e-2, delta vs Kepler = -1.130e-7

--- Calculating the orbital precession of Jupiter caused by the gravitational field of the Sun ---
Calculating solution with 32.00 points and 100.0 arc segments per point
>>> Calculating orbital precession with 3200 equal arc angle segments
Solving, iteration = 1.000
Solving, iteration = 2.000
Solving, iteration = 3.000
>>> Solution complete for path_option 1

--- Jupiter Orbital Precession Solution Results -----
Rotation      = 360.000002155590 degrees
Rotation time = 4332.58894177582 days
Precession    = 2.259e-6 degrees per orbit
Precession    = 6.85558578133894e-2 arcsecond per century

--- For comparison only -----
--- General Relativity predicted value = 6.23504e-2 arcsecond per century

------
This example calculates the orbital motion of Saturn
Initial conditions taken from https://nssdc.gsfc.nasa.gov/planetary/factsheet/saturnfact.html
Position   x = -5.76882e10 m,   y = 1.34819e12 m,  z = -2.12351e10 m
Velocity  vx = -10164.0 m/s,  vy = -428.435 m/s,  vz = 411.245 m/s,  v = 10181.4 m/s
Beta    vx/c = -3.39035e-5, vy/c = -1.42911e-6, vz/c = 1.37177e-6,   v/c = 3.39613e-5

--- Calculating Saturn orbit ---
Total rotation angle for calculation = 360.0 degrees
Calculating solution with 360.0 points and 1.000 arc segments per point
>>> Calculating path with 360.0 equal arc angle segments
>>> Solution complete for path_option 3

--- Saturn Orbital Solution Results -----
Orbital Period = 10759.2 days, delta vs Kepler = -120.1 seconds
Circumference  = 8.95861861792e12 meters
Perihelion     = 1.34958993923e12 meters, delta vs Kepler = 0.000 meters
Aphelion       = 1.50411962315e12 meters, delta vs Kepler = -4.241e5 meters
Eccentricity   = 5.4150459e-2, delta vs Kepler = -1.406e-7

--- Calculating the orbital precession of Saturn caused by the gravitational field of the Sun ---
Calculating solution with 32.00 points and 100.0 arc segments per point
>>> Calculating orbital precession with 3200 equal arc angle segments
Solving, iteration = 1.000
Solving, iteration = 2.000
Solving, iteration = 3.000
>>> Solution complete for path_option 1

--- Saturn Orbital Precession Solution Results -----
Rotation      = 360.000001247445 degrees
Rotation time = 10759.2198113414 days
```

```
Precession    = 1.479e-6 degrees per orbit
Precession    = 1.80726935019506e-2 arcsecond per century

--- For comparison only -----
--- General Relativity predicted value = 1.36996e-2 arcsecond per century

------
This example calculates the orbital motion of Uranus
Initial conditions taken from https://nssdc.gsfc.nasa.gov/planetary/factsheet/uranusfact.html
Position   x = -2.70003e12 m,   y = 4.29342e11 m,  z = 3.64838e10 m
Velocity  vx = -1119.74 m/s,  vy = -7040.83 m/s,  vz = -11.2330 m/s,  v = 7129.32 m/s
Beta    vx/c = -3.73505e-6, vy/c = -2.34857e-5, vz/c = -3.74694e-8,   v/c = 2.37808e-5

--- Calculating Uranus orbit ---
Total rotation angle for calculation = 360.0 degrees
Calculating solution with 360.0 points and 1.000 arc segments per point
>>> Calculating path with 360.0 equal arc angle segments
>>> Solution complete for path_option 3

--- Uranus Orbital Solution Results -----
Orbital Period = 30685.4 days, delta vs Kepler = -275.2 seconds
Circumference  = 1.80198838658e13 meters
Perihelion     = 2.73420017242e12 meters, delta vs Kepler = 0.000 meters
Aphelion       = 3.00489975673e12 meters, delta vs Kepler = -6.473e5 meters
Eccentricity   = 4.7167603e-2, delta vs Kepler = -1.075e-7

--- Calculating the orbital precession of Uranus caused by the gravitational field of the Sun ---
Calculating solution with 32.00 points and 100.0 arc segments per point
>>> Calculating orbital precession with 3200 equal arc angle segments
Solving, iteration = 1.000
Solving, iteration = 2.000
Solving, iteration = 3.000
>>> Solution complete for path_option 1

--- Uranus Orbital Precession Solution Results -----
Rotation      = 360.000000653148 degrees
Rotation time = 30685.3993979279 days
Precession    = 0.000 degrees per orbit
Precession    = 0.00000000000000 arcsecond per century

--- For comparison only -----
--- General Relativity predicted value = 2.38679e-3 arcsecond per century

------
This example calculates the orbital motion of Neptune
Initial conditions taken from https://nssdc.gsfc.nasa.gov/planetary/factsheet/neptunefact.html
Position   x = 3.15217e12 m,   y = 3.14919e12 m,  z = -1.37406e11 m
Velocity  vx = -3872.61 m/s,  vy = 3876.70 m/s,  vz = 9.58979 m/s,  v = 5479.60 m/s
Beta    vx/c = -1.29176e-5, vy/c = 1.29313e-5, vz/c = 3.19881e-8,   v/c = 1.82780e-5

--- Calculating Neptune orbit ---
Total rotation angle for calculation = 360.0 degrees
Calculating solution with 360.0 points and 1.000 arc segments per point
>>> Calculating path with 360.0 equal arc angle segments
>>> Solution complete for path_option 3

--- Neptune Orbital Solution Results -----
Orbital Period = 60189.0 days, delta vs Kepler = -121.9 seconds
Circumference  = 2.82515706280e13 meters
Perihelion     = 4.45785396231e12 meters, delta vs Kepler = -1.758e-2 meters
Aphelion       = 4.53506597041e12 meters, delta vs Kepler = -3.365e4 meters
Eccentricity   = 8.5858663e-3, delta vs Kepler = -3.710e-9

--- Calculating the orbital precession of Neptune caused by the gravitational field of the Sun ---
Calculating solution with 32.00 points and 100.0 arc segments per point
>>> Calculating orbital precession with 3200 equal arc angle segments

Solving, iteration = 1.000
Solving, iteration = 2.000

>>> Solution complete for path_option 1


--- Neptune Orbital Precession Solution Results -----
Rotation      = 360.000000367721 degrees
Rotation time = 60188.9988169844 days
Precession    = 0.000 degrees per orbit
Precession    = 0.00000000000000 arcsecond per century
```

```
--- For comparison only -----
--- General Relativity predicted value = 7.74885e-4 arcsecond per century

------
This example calculates the orbital motion of Pluto
Initial conditions taken from https://nssdc.gsfc.nasa.gov/planetary/factsheet/Plutofact.html
Position   x = -3.01763e12 m,   y = -3.02201e12 m,  z = 1.19636e12 m
Velocity  vx = 4148.92 m/s,  vy = -4430.32 m/s,  vz = -726.018 m/s,  v = 6112.97 m/s
Beta    vx/c = 1.38393e-5, vy/c = -1.47780e-5, vz/c = -2.42173e-6,   v/c = 2.03907e-5

--- Calculating Pluto orbit ---
Total rotation angle for calculation = 360.0 degrees
Calculating solution with 360.0 points and 1.000 arc segments per point
>>> Calculating path with 360.0 equal arc angle segments
>>> Solution complete for path_option 3

--- Pluto Orbital Solution Results -----
Orbital Period = 90559.8 days, delta vs Kepler = -1.648e4 seconds
Circumference  = 3.65153235931e13 meters
Perihelion     = 4.43507948298e12 meters, delta vs Kepler = 0.000 meters
Aphelion       = 7.37299107514e12 meters, delta vs Kepler = -3.569e7 meters
Eccentricity   = 2.4880539e-1, delta vs Kepler = -2.271e-6

--- Calculating the orbital precession of Pluto caused by the gravitational field of the Sun ---
Calculating solution with 32.00 points and 100.0 arc segments per point
>>> Calculating orbital precession with 3200 equal arc angle segments
Solving, iteration = 1.000
Solving, iteration = 2.000
>>> Solution complete for path_option 1

--- Pluto Orbital Precession Solution Results -----
Rotation      = 360.000002618067 degrees
Rotation time = 90559.9962239253 days
Precession    = 2.561e-6 degrees per orbit
Precession    = 3.71901470241254e-3 arcsecond per century

--- For comparison only -----
--- General Relativity predicted value = 4.18080e-4 arcsecond per century
```

Example 4: Orbital motion of Star S2 in the galactic center

Script file: Example_4_S2_galactic_center.m

This script file calculates the orbital motion, redshift, and precession of the Star S2 in the center of the Milky Way Galaxy. S2 orbits the galactic core in about 16 years with a maximum velocity of almost 2.7% that of light. The orbital plane is tilted such that the radial velocity relative to Earth alternates between positive and negative. This alternating velocity produces a corresponding periodic variation in the redshift of light emitted by the star S2. The motion of S2 and the gravitational field impart an additional influence on the redshift of the light from S2 viewed by Earth astronomers. The calculated variation agrees closely with the data [9].

Figure 10 presents the calculated orbit and the apparent radial velocity deviation caused by the influence of gravity and motion on the natural frequency of matter. Figure 11 illustrates the measured or "apparent" radial velocity shift (from spectroscopy data) compared with the orbital solution.

These results are in close agreement with Reference [9] Figures 2 and 3. The calculated orbital precession is 11.7 arc seconds per orbit in close agreement with the measured value of 12 [10].

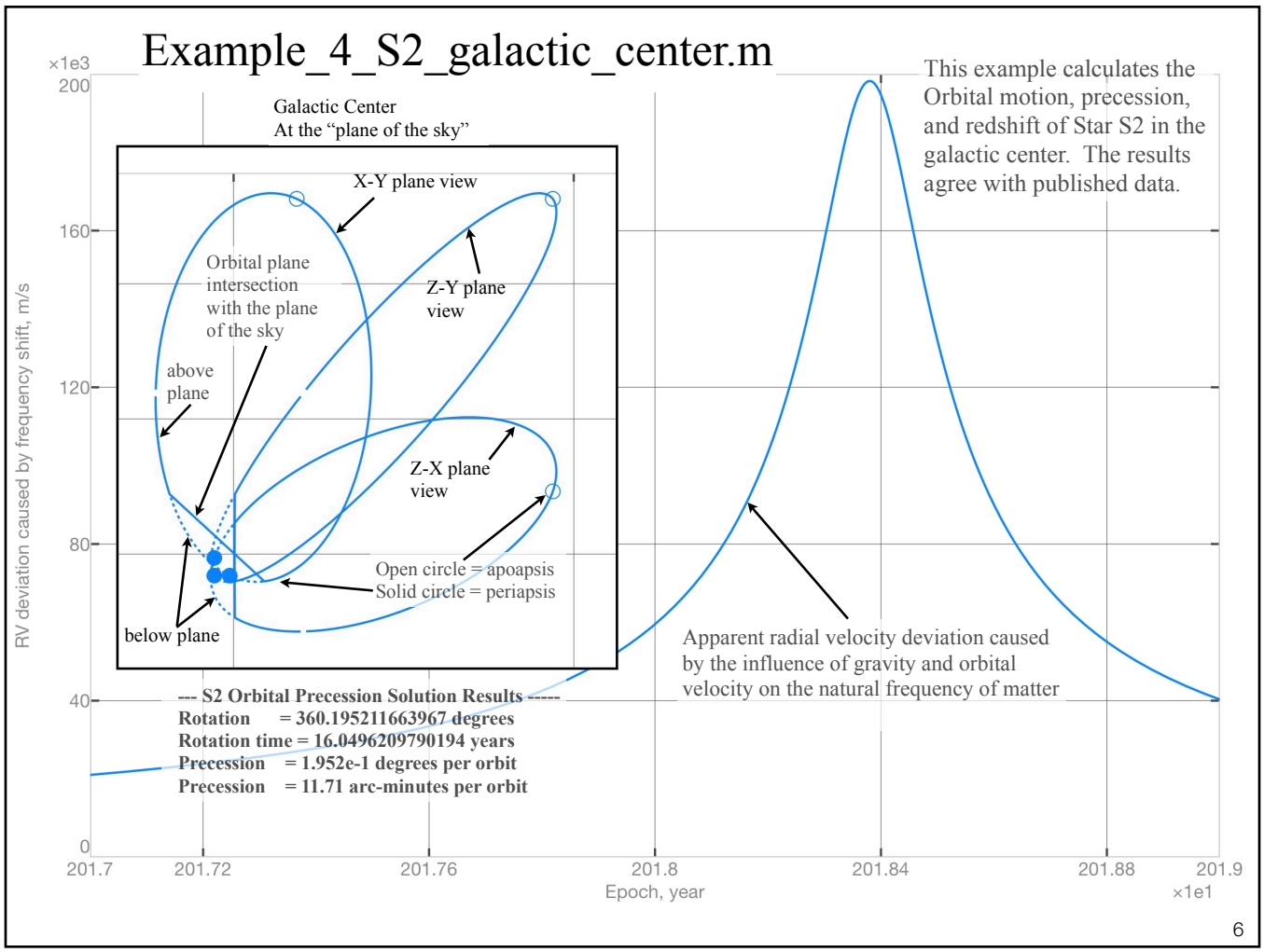


Figure 10 Orbital plot of the Galactic center star S2 and the apparent radial velocity deviation caused by gravity and motion-induced frequency shift.

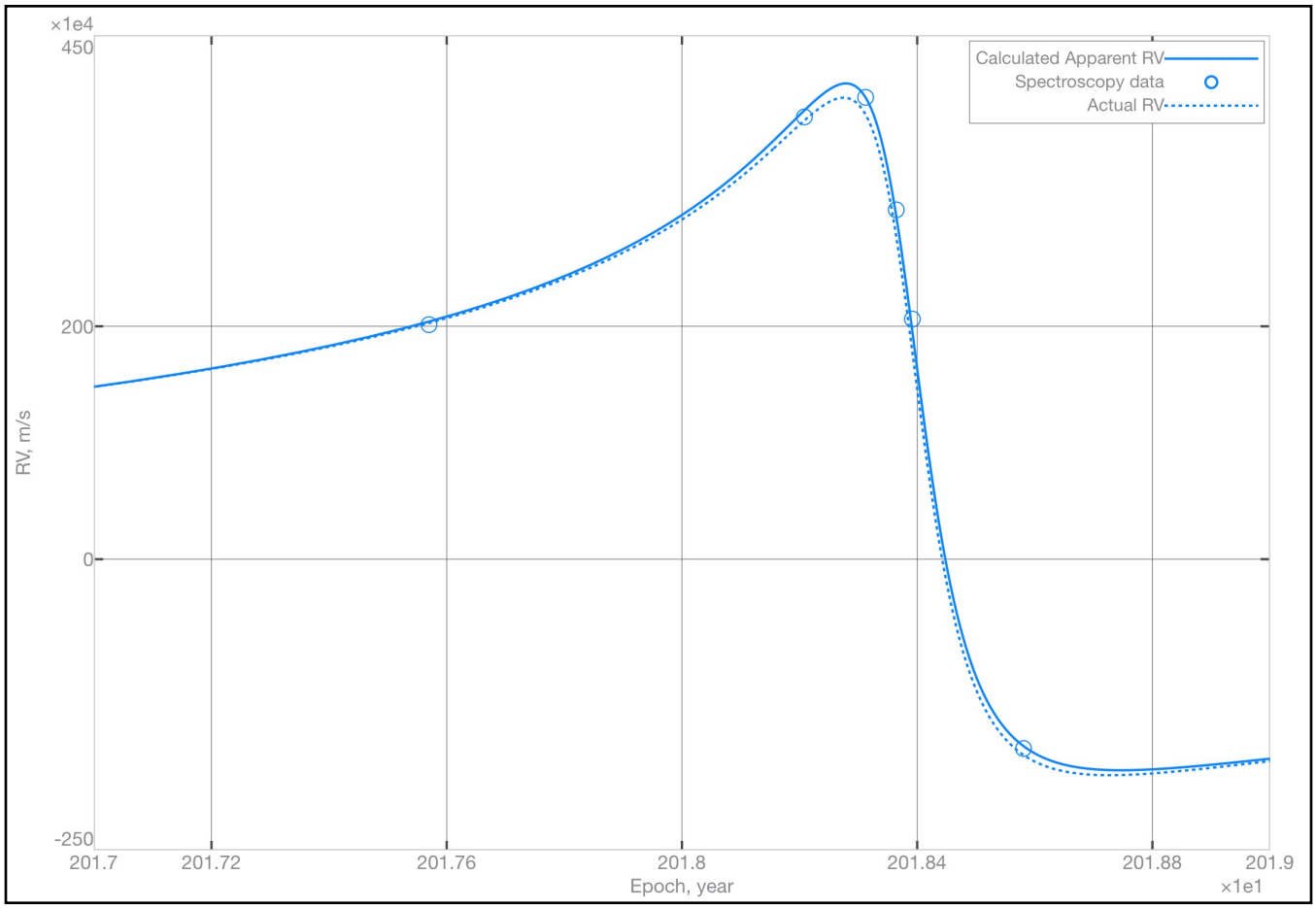


Figure 11 Apparent (Newtonian) radial velocity compared with the theoretical orbital model (see Figure 10) that includes frequency shift caused by motion and gravity.

The script input file, included in the MatLab Script File section of this document, contains comments documenting the calculation.

Note: The galactic center is thought to be a black hole, by many, based on general relativity (GR). However, the GR redshift equation is a 1st order approximation of a natural logarithmic solution. Therefore, based on TMG, the GR solution may be artificial. It may be that the curvature of motion caused by the "strong" gravitational field of the galactic center reduces its emissivity below the signature level of surrounding dust and gas. See Example 8 for a simulation compared with data.

Example_4_S2_galactic_center.m Output

```
--- Inputs ---
Example_4_S2_galactic_center

--- Outputs ---
This example calculates the orbital motion and precession of the star S2 near the galactic core
Newtons universal gravitational constant, G = 6.67430e-11 m^3/(kg-s)
The standard speed of light, cs = 299792458.0 m/s
Reference speed of light  = 299792464.336 m/s at 2.5234671e20 meters from the galactic center
Galactic center mass = 7.90390303102947e36 kg
Position   x = -2.22262e12 m,   y = -1.27476e13 m,  z = -1.13573e13 m
Velocity  vx = -6.55826e6 m/s,  vy = 3.12147e6 m/s,  vz = -2.22014e6 m/s,  v = 7.59496e6 m/s
Beta    vx/c = -2.18909e-2, vy/c = 1.04192e-2, vz/c = -7.41063e-3,   v/c = 2.53513e-2

--- Calculating S2 orbital precession ---
Total rotation angle for calculation = 360.0 degrees
Calculating solution with 360.0 points and 10.00 arc segments per point
>>> Calculating orbital precession with 3600 equal arc angle segments
Solving, iteration = 1.000
Solving, iteration = 2.000
Solving, iteration = 3.000
Solving, iteration = 4.000

>>> Solution complete for path_option 1

--- S2 Orbital Precession Solution Results -----
Rotation      = 360.195211663967 degrees
Rotation time = 16.0496209790194 years
Precession    = 1.952e-1 degrees per orbit
Precession    = 11.71 arc-minutes per orbit

Recalculating results for a simple 360-degree rotation starting at the Aphelion for plotting and
comparing the gravitational redshift with data

>>> Calculating path with 3600 equal arc angle segments
>>> Solution complete for path_option 3
--- S2 Orbital Solution Results -----
Orbital Period = 16.0420 years, delta vs Kepler = 63.38 seconds
Circumference  = 7.16914376266e14 meters
Perihelion     = 1.72172298319e13 meters, delta vs Kepler = 4.164e7 meters
Aphelion       = 2.84309716201e14 meters, delta vs Kepler = -3.424e7 meters
Eccentricity   = 8.8579973e-1, delta vs Kepler = -2.734e-7

Plot is generated.
…
Plot is generated.
```

Example 5 Strong gravity field precession

Script file: Example_5_strong_gravity_field_precession.m

This example illustrates orbital precession in a strong gravitational field. The calculation starts at the minimum orbital radius or periapsis. One orbit is defined as 360^o of path rotation. The orbital precession is defined as the additional rotation required to return back to the minimum orbital radius or periapsis. By this definition, the orbital precession can exceed 360^o.

Figure 12 illustrates orbital precession. In this chart $\phi = -\frac{2Gm}{c_s^2 r}$. The right-hand illustration is plotted as a function of $-\phi$ resulting in positive numerical values for the x-axis. For this example, fraction_escape is defined as:

$$\text{At the orbital periapsis; } \beta = \beta_{circular} + (fraction_{escape}) * (\beta_{escape} - \beta_{circular})$$

The input file can be modified to calculate each solution illustrated in Figure 12.

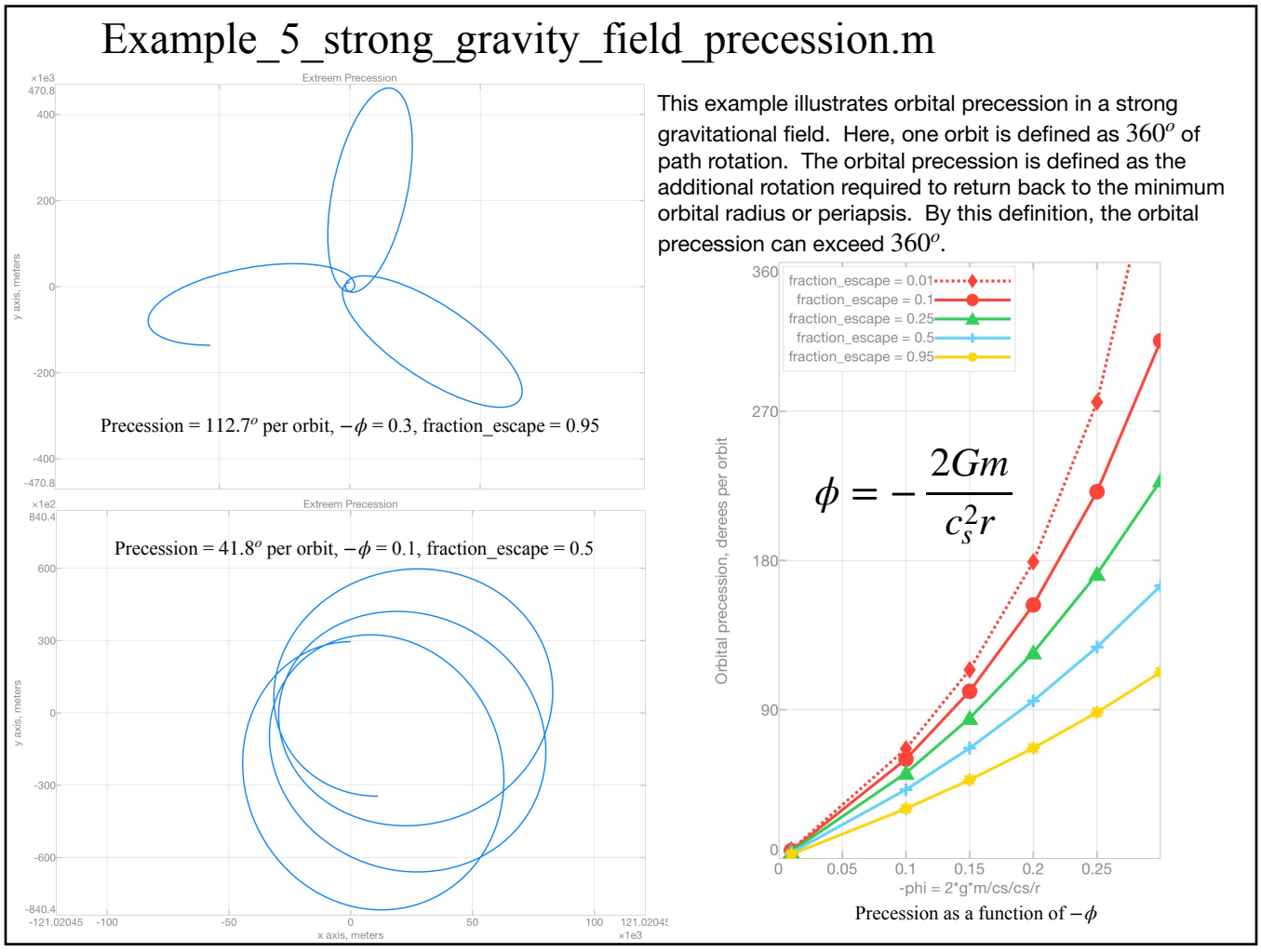


Figure 12 Orbital precession in a strong gravitational field. Rotation is counter clockwise.

Example_5_strong_gravity_field_precession.m Output

Only output for the upper left-hand portion of Figure 12 is provided. However, the user can modify the input file to calculate the rest of the solutions presented in Figure 12.

```
--- Inputs ---
Example_5_strong_gravity_field_precession

--- Outputs ---
Newtons G = 6.67430e-11 m^3/kg-s,  Standard speed of light = 299792458.0 m/s
Governing body = 1.0 solar mass = 1.988e30 kg
Reference speed of light  = 222091715.309 m/s at 9844.1343 meters from the governing body

--- At the orbital periapsis ---
phi = -3.000e-1, gravitational redshift = 1.61834e-1
v/c equal to 4.20084e-1 results in a circular orbit.  v/c = 5.09099e-1 equals escape velocity.
v/c for this orbit = 5.04648e-1 at the periapsis
fraction between circular orbit and escape velocity = 9.5e-1

--- Initial conditions at the orbital periapsis ---
Position   x = 0.00000 m,   y = 9844.13 m,  z = 0.00000 m
Beta, vx/c = -5.04648e-1, vy/c = 0.00000, vz/c = 0.00000

--- Calculating the orbital precession caused by the gravitational field of the governing body ---
Calculating solution with 360.0 points and 10.00 arc segments per point

>>> Calculating path with 3600 equal arc angle segments
>>> Solution complete for path_option 3

--- Orbital Solution Results -----
Rotation      = 1259.99999999995 degrees
Rotation time = 2.14484700828567e-6 days
Plot is generated.



--- Inputs ---
Example_5_strong_gravity_field_precession

--- Outputs ---
Newtons G = 6.67430e-11 m^3/kg-s,  Standard speed of light = 299792458.0 m/s
Governing body = 1.0 solar mass = 1.988e30 kg
Reference speed of light  = 271263433.643 m/s at 29532.403 meters from the governing body

--- At the orbital periapsis ---
phi = -1.000e-1, gravitational redshift = 5.12711e-2
v/c equal to 2.29416e-1 results in a circular orbit.  v/c = 3.08484e-1 equals escape velocity.
v/c for this orbit = 2.68950e-1 at the periapsis
fraction between circular orbit and escape velocity = 5.0e-1

--- Initial conditions at the orbital periapsis ---
Position   x = 0.00000 m,   y = 29532.4 m,  z = 0.00000 m
Beta, vx/c = -2.68950e-1, vy/c = 0.00000, vz/c = 0.00000

--- Calculating the orbital precession caused by the gravitational field of the governing body ---
Calculating solution with 360.0 points and 10.00 arc segments per point

>>> Calculating path with 3600 equal arc angle segments
>>> Solution complete for path_option 3

--- Orbital Solution Results -----
Rotation      = 1259.99999999995 degrees
Rotation time = 2.97091003606632e-7 days
Plot is generated.
```

Example 6 Sunlight deflection and the Shapiro Time Delay

Script file: Example_6_Sunlight_deflection_Shapiro_time_delay.m

Light follows a curved path near a gravitational body. This equates to about 1.75 arc-seconds for light that just grazes the solar limb on its way to earth [11].

It also takes light longer to pass through space near a governing body [12], [13], [14] as confirmed by test results documented in the following extract from the PHYSICAL REVIEW LETTERS [13].

> FOURTH TEST OF GENERAL RELATIVITY: PRELIMINARY RESULTS
>
> I. I. Shapiro, G. H. Pettergill, M. E. Ash, M. L.Stone, W. B. Smith, B. P. Ingalls, and R. A. Brockelman, Phys. Rev. Lett. 20, 1265 (1968).
>
> ... 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. For a radar pulse transmitted from the earth and reflected by another planet, the increase in the round-trip time delay, attributable to the predicted gravitational influence of the sun on the propagation, would be ≈200 μsec if the path of the pulse were to graze the solar limb.
>
> An intensive program was therefore undertaken early in 1965 to build a new transmitter and receiver system to provide the Lincoln Laboratory Haystack radar with the capability to measure to within 10 μsec the time delays of pulses traveling between the earth and Mercury or Venus when either planet was on the other side of the sun from the earth-the superior-conjunction alignment. The improved radar was put into operation shortly before the last such conjunction of Venus, which occurred on 9 November 1966. Time-delay measurements were made then and during the subsequent superior conjunctions of Mercury on 18 January, 11 May, and 24 August 1967. The most reliable of these data agree, on average, with the excess-delay predictions of general relativity to well within the experimental uncertainly of ±20%. ...

This example script file calculates the time delay based on TMG [1]. For this example, the Shapiro time delay is calculated as:

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

The time delay is a function of path length and how close the light passes to the Sun. Table 1 provides sample results for light (passing between the Earth to the superior conjunction of the inner solar system planets) that just grazes the solar limb. The user can calculate separate paths for comparison if desired.

The results presented in Table 1 represent converged solutions as illustrated in Figure 13. The time delay plotted with respect to path time and compared with the variation in light speed, deflection, and radial distances from the Sun is illustrated in Figure 14.

These results demonstrate that more than 60% of the time delay occurs within a light path distance of 30 seconds from the Sun. And the calculated increase in the light path distance between Earth and Venus is less than three meters compared with the Euclidean value.

Table 2 Predicted time delay results are in close agreement with data.

[1]Shapiro Time Delay caused by the gravitational field of the Sun						
Planet	**One way Distance, meters**			**Time Seconds**	**[2]Shapiro Delay Seconds**	**[3]Shapiro Delay Seconds**
	Euclidian distance	***Path length***	***Delta Path - Eucl.***	***Path Time One way***	***Path - Euclidean Round trip***	***Path - Euclidean Round trip***
Mercury	207507062254.803	207507062256.288	1.485	692.16915	**1.929e-4**	**2.203e-4**
Venus	257806812665.783	257806812663.540	2.243	859.95106	**1.986e-4**	**2.326e-4**
Mars	377534524553.764	377534524550.528	3.236	1259.3197	**1.975e-4**	**2.473e-4**

[1]Shapiro Delay for a round trip radar reflection from Earth to the planet at superior conjunction. One-way distance set equal to the sum of the Semi-major axis of the Earth and the planet of interest.

[2]Shapiro Delay = Time delay with respect to light propagation away from the Earth at the Earth's orbital radius from the Sun where $c_r = 299792458.418$ m/s

[3]Shapiro Delay = Time delay with respect to light propagation away from the influence of the Sun where $c_r = 299792464.336$ m/s

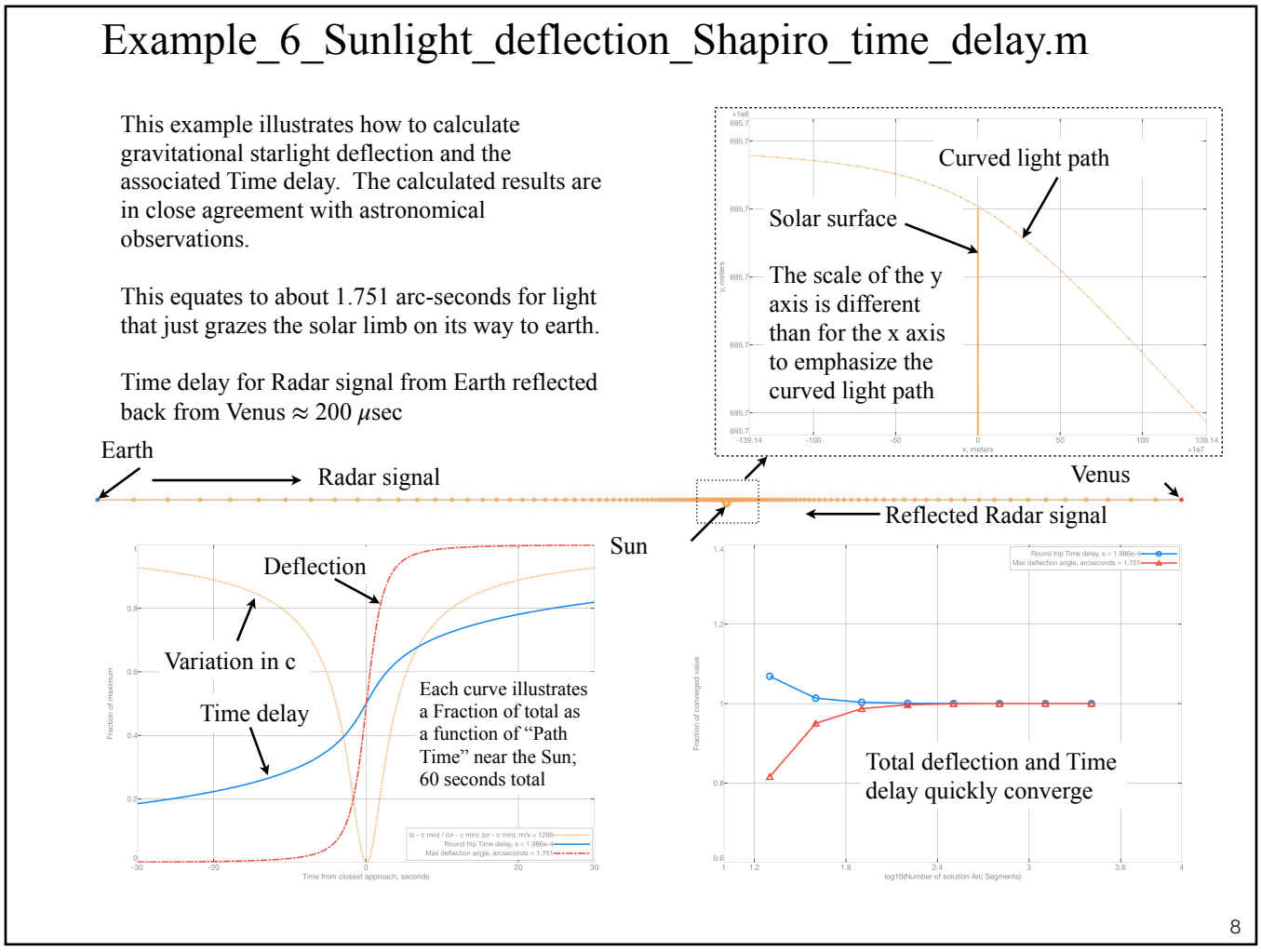


Figure 13 Light from Earth passing near the Sun and reflected back by Venus

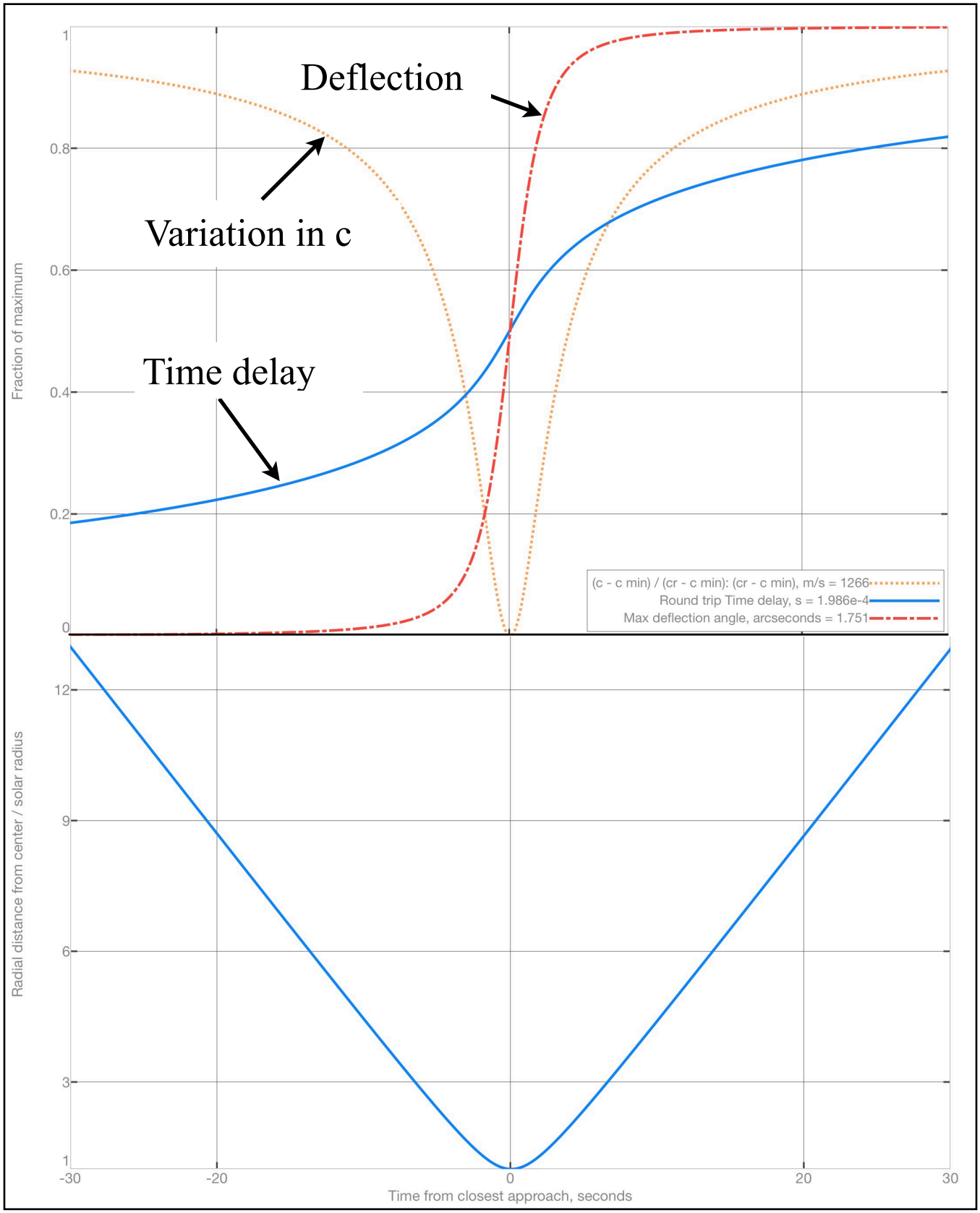


Figure 14 Light grazing the solar limb on its way to Venus from Earth.

Example_6_Sunlight_deflection_Shapiro_time_delay.m Output

Only output for light emanating from Earth and reflected by Venus is provided. The user can modify the script to calculate the rest of the results presented in Table 2.

```
--- Inputs ---
Example_6_Sunlight_deflection_Shapiro_time_delay

--- Outputs ---
Calculating the deflection of light passing near the Sun
Total path length for Earth to Venus superior conjunction= 2.57807e11 meters
Total path length for calculation = 2.57807e11 meters

Constants:
Newtons G = 6.67420e-11 m^3/(kg-s),  Standard speed of light = 299792458.0 m/s
Earth mass = 5.97230e24 kg,  Earth radius = 6367450.0 m
Reference speed of light  = 299792458.418 m/s at 1.4960000e11 meters from the Sun which is Earths
average orbital radius
Mass of the Sun   = 1.9884330706e30 kg
Radius of the Sun = 695700000.00 m

Initial Position x = -149597870700.000 y = 695703050.000000 z = 0.00000000000000 meters
Initial velocity (b = v/c),  bx = 1.000 by = 0.000 bz = 0.000

Calculating solution with = 2000 steps

>>> Calculating path with 2000 arc segments, arc angle segments = f(r)

Solving, iteration = 1.000
Solving, iteration = 2.000
...
Solving, iteration = 11.00

>>> Solution complete for path_option 2
Final Position x = 108208875155.124 y = 694784384.400379 z = 0.00000000000000 meters
Path   time = 9.953e-3 days = 859.950837291423 seconds
Path length = 257806745859.004 meters
Euclidian distance between the beginning and end = 257806745856.761 meters
(Path length - Euclidian distance) = 2.24295043945312 meters

Shapiro time delay = 1.98624929907965e-4 seconds with respect to electromagnetic wave propagation at
the earths orbital radius from the Sun

Speed of light away from the influence of the Sun = 299792464.335799 m/s

Shapiro time delay = 2.32577351653163e-4 seconds with respect electromagnetic wave propagation away
from the influence of the Sun

Angle between the beginning and ending directions = 4.864e-4 degrees = 1.751 arcseconds

Trajectory radius: Min = 695983500.580729, Max = 149599488370.474 meters
```

Example 7 The gravitational bending of light and cosmic rays passing near a neutron star

Script file: Example_7_Neutron_star_light_deflection.m

The curved path of matter passing through a gravitational field is a function of velocity relative to the speed of light (see TMG equation 7-12). This presents the opportunity to investigate the gravitational field of a Neutron star by observing both light and cosmic rays passing near a neutron star on the way to Earth.

Figure 15 illustrates light and cosmic rays with v/c = 0.7 passing near the neutron Star RX J0720.4-3125 [15] calculated in this example. The author is not aware of any data to compare with these results. However, this illustrates one method for evaluating the properties of a neutron star.

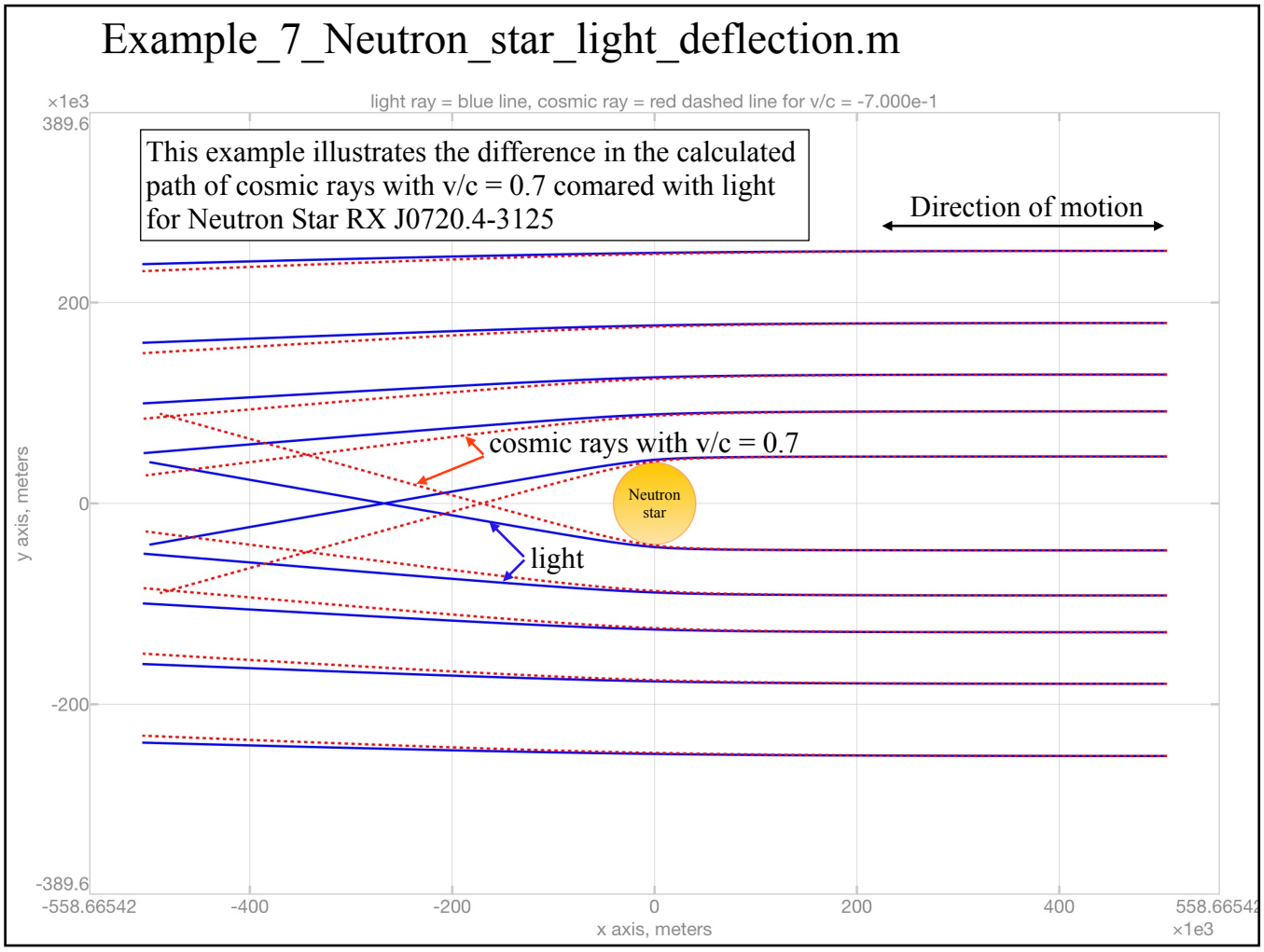


Figure 15 Light and cosmic rays passing near Neutron Star RX J0720.4-3125

Example_7_Neutron_star_light_deflection.m Output

```
--- Inputs ---
Example_7_Neutron_star_light_deflection

--- Outputs ---
Calculate the deflection of light passing near a Neutron star

Constants:
G = 6.67420e-11 m^3/(kg-s),  Standard speed of light = 299792458.0 m/s
Earth mass = 5.97230e24 kg,  Earth radius = 6367450.0 m
Reference speed of light  = 299792458.418 m/s at 1.4960000e11 meters from the neutron star
Mass of governing body = 1.2300000000 solar masses
Neutron star radius  = 38958.5370 meters
Neutron star density = 9.87460e15 kg/m^3

Total path length for calculation = 1.01292e6 meters
Initial velocity (b = v/c),  bx = -1.000 by = 0.000 bz = 0.000
Calculate the path of motion for light with the TMG solver from multiple starting points.
Initial Position x = 506460.980447891 y = 46750.2443490361 z = 0.00000000000000 meters
Calculate solution for initial radial offset of phi = -7.770e-2 and zg = 3.961e-2

>>> Calculating path with 2000 equal arc length segments
>>> Solution complete for path_option 4
Initial Position x = 506460.980447891 y = 91630.4789241107 z = 0.00000000000000 meters
Calculate solution for initial radial offset of phi = -3.964e-2 and zg = 2.002e-2

>>> Calculating path with 2000 equal arc length segments
>>> Solution complete for path_option 4
Initial Position x = 506460.980447891 y = 128282.670493755 z = 0.00000000000000 meters
Calculate solution for initial radial offset of phi = -2.832e-2 and zg = 1.426e-2

>>> Calculating path with 2000 equal arc length segments
>>> Solution complete for path_option 4
Initial Position x = 506460.980447891 y = 179595.738691257 z = 0.00000000000000 meters
Calculate solution for initial radial offset of phi = -2.023e-2 and zg = 1.016e-2

>>> Calculating path with 2000 equal arc length segments
>>> Solution complete for path_option 4
Initial Position x = 506460.980447891 y = 251434.034167760 z = 0.00000000000000 meters
Calculate solution for initial radial offset of phi = -1.445e-2 and zg = 7.250e-3

>>> Calculating path with 2000 equal arc length segments
>>> Solution complete for path_option 4
Plot is generated. ...

Calculate the deflection of a cosmic ray passing near a Neutron star, v/c = -7.00000000000000e-1
Calculate solution for initial radial offset of phi = -7.770e-2 and zg = 3.961e-2

>>> Calculating path with 2000 equal arc length segments
>>> Solution complete for path_option 4
Calculate for initial radial offset of phi = -3.964e-2 and zg = 2.002e-2

>>> Calculating path with 2000 equal arc length segments
>>> Solution complete for path_option 4
Calculate solution for initial radial offset of phi = -2.832e-2 and zg = 1.426e-2

>>> Calculating path with 2000 equal arc length segments
>>> Solution complete for path_option 4
Calculate solution for initial radial offset of phi = -2.023e-2 and zg = 1.016e-2

>>> Calculating path with 2000 equal arc length segments
>>> Solution complete for path_option 4
Calculate solution for initial radial offset of phi = -1.445e-2 and zg = 7.250e-3

>>> Calculating path with 2000 equal arc length segments
>>> Solution complete for path_option 4

Plot is generated. ...
```

Example 8 The emissivity, shadow, and curved path of light escaping from or passing near an apparent black hole or quasar. This is done for Sgr A* at the galactic center and for the central gravitational body of M87*

Script file: Example_8_Apparent_Black_Hole_Shadow_Diameter_and_Emissivity.m

This example calculates the shadow diameter and emissivity as a function of radius for two galactic center apparent black holes. The first is Sgr A* in the Milky Way. The second is M87*.

The radius at which light will orbit a gravitational body is calculated based on TMG equation 7-12:

$$r_c = \frac{c_s^2 r^2}{(1 + \frac{1}{\beta^2})MG(\hat{r} \cdot \hat{r}_c)}$$

If we set $r_c = r$, $\beta = 1$, and the direction of motion normal to the radial direction, then:

$$r_c = \frac{2MG}{c_s^2} = r_s$$

This is the radius at which (according to TMG) light will orbit a governing body. It is also commonly referred to as the General Relativity Schwarzschild radius. Within this dimension, the General Relativity 1st order approximation breaks down and cannot be used to describe a gravitational field (see Reference [16] section 3.2 Quasar Redshift). The TMG approach provides a similar but possibly more accurate natural logarithmic solution that is free of this limitation.

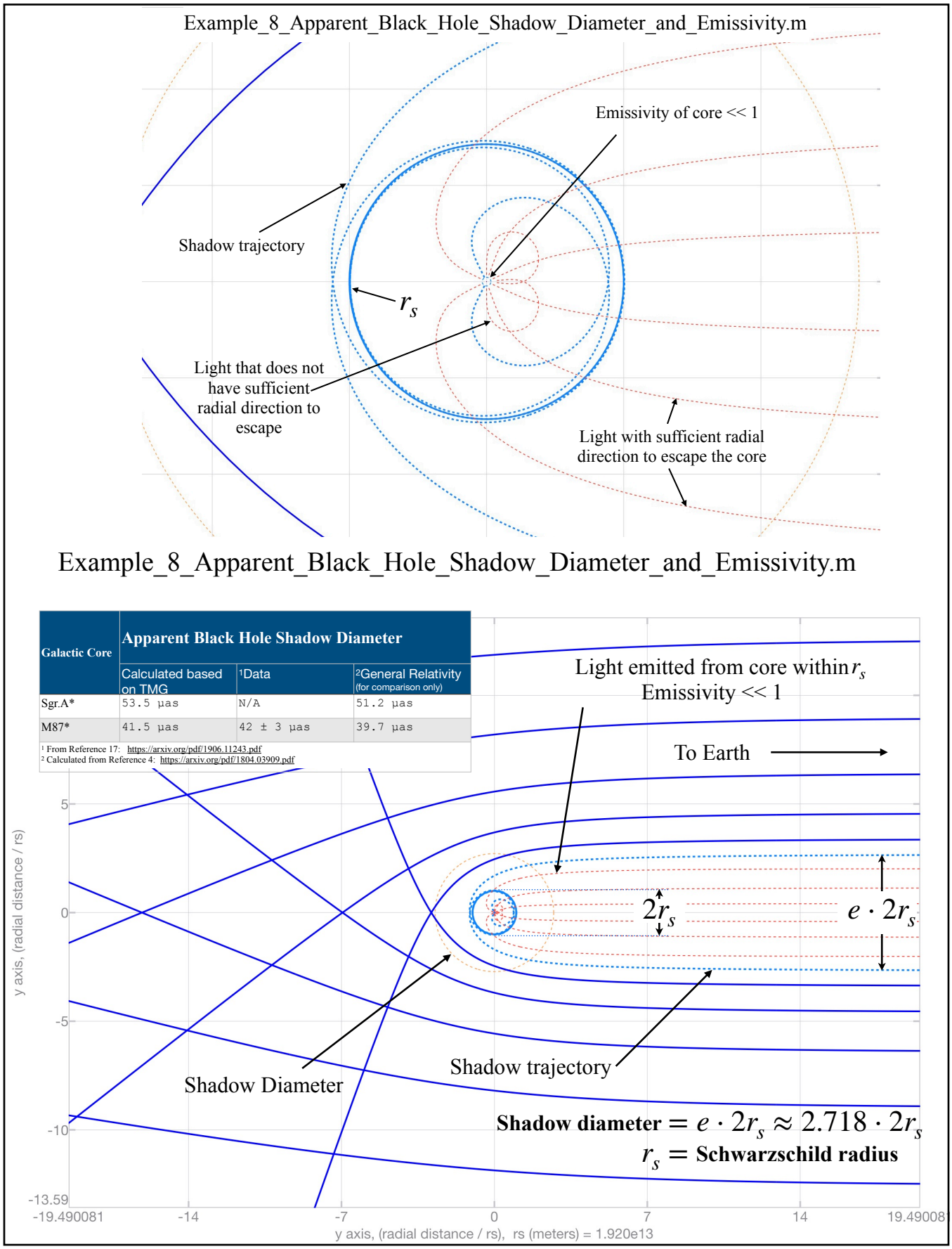
Inside the Schwarzschild radius, the curvature of motion is so great that light cannot orbit the body. Hence, light that does not have sufficient outward radial direction, curves inward and is captured (see top and bottom portions of Figure 16). This reduces the emissivity or the amount of light escaping from within the Schwarzschild radius (see Figure 17) causing the shadow observed by astronomers.

Table 3 and Figure 16 illustrate that the apparent black hole shadow diameter, calculated with this example script file, is in very close agreement with data.

Table 3 The TMG apparent black hole shadow diameter closely matches data

Galactic Core	Apparent Black Hole Shadow Diameter		
	Calculated based on TMG	[1]Data	[2]General Relativity (for comparison only)
Sgr.A*	53.5 μas	N/A	51.2 μas
M87*	41.5 μas	42 ± 3 μas	39.7 μas

[1] From Reference 17: https://arxiv.org/pdf/1906.11243.pdf
[2] Calculated from Reference 4: https://arxiv.org/pdf/1804.03909.pdf



Galactic Core	Apparent Black Hole Shadow Diameter		
	Calculated based on TMG	[1]Data	[2]General Relativity (for comparison only)
Sgr.A*	53.5 μas	N/A	51.2 μas
M87*	41.5 μas	42 ± 3 μas	39.7 μas

[1] From Reference 17: https://arxiv.org/pdf/1906.11243.pdf
[2] Calculated from Reference 4: https://arxiv.org/pdf/1804.03909.pdf

Figure 16. Apparent black hole shadow for M87*. The shadow for Sgr A* is similar and can be calculate and plotted with the example script file.

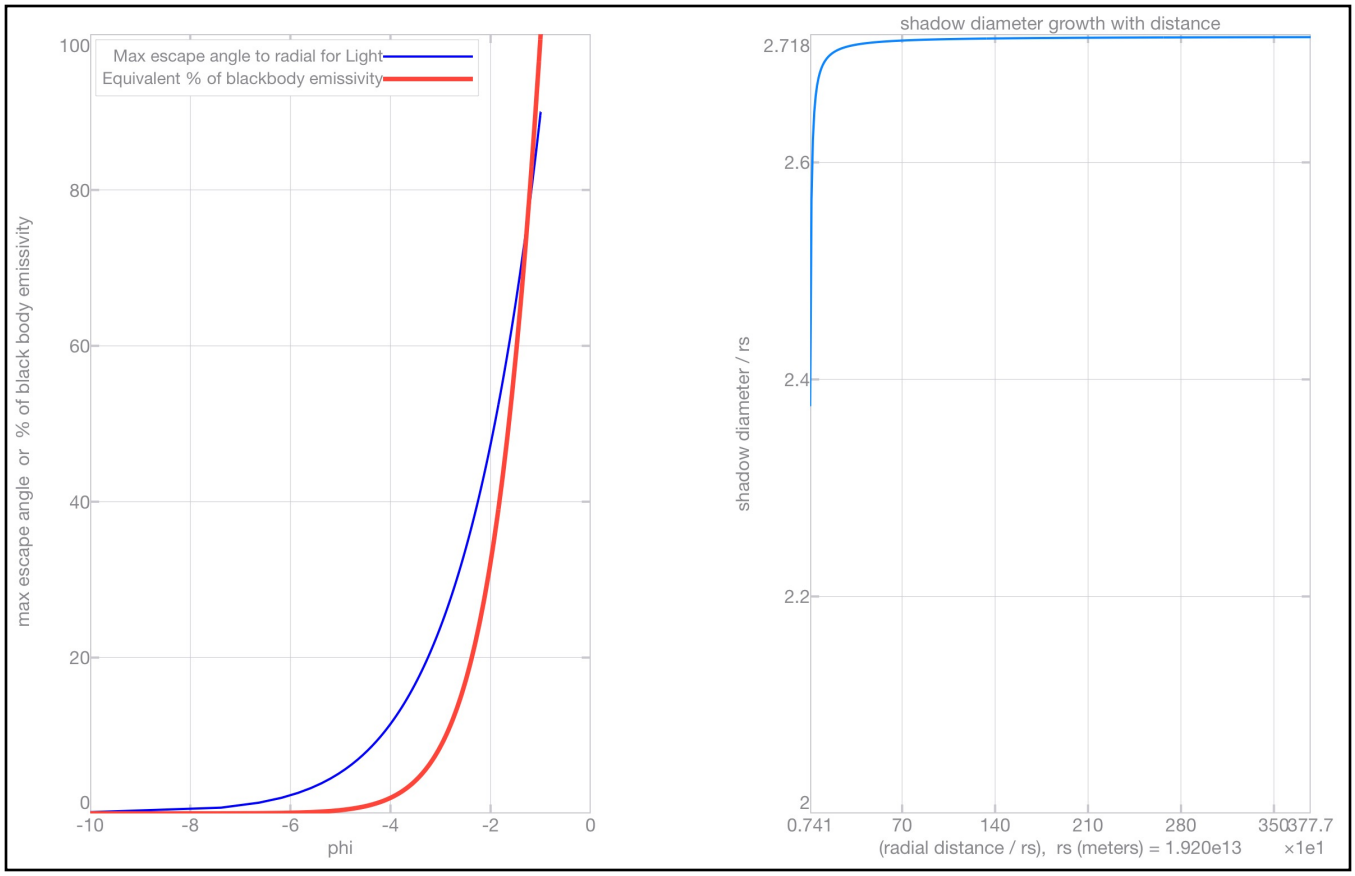


Figure 17. Left plot: Minimum light escape angle and emissivity as a function of ϕ
Right plot: Growth of apparent black hole shadow diameter with distance from center
Note: $\phi = -\frac{2Gm}{c_s^2 r}$. For $\phi = -1$, $r = r_s$, where r_s = the Schwarzschild radius.

Figure 16 illustrates the results calculated with this script file. The dashed blue line represents the shadow trajectory. This is the trajectory associated with light escaping at an angle of just less than 90 degrees to radial, at r_s, from the core. The solid dark blue lines illustrate light trajectories outside of the shadow. The red dashed lines illustrate light escaping with the exception of two depicting emitted light that curves back into the core. The high curvature of motion can cause the emissivity to approach zero for a compact core as illustrated in Figure 17 left plot. This reduces the central emissivity producing the shadow observed by astronomers for M87* (see Reference [17]). See Reference [16] for more information regarding the reduced emissivity of compact gravitational bodies.

Based on these calculations, if the core of M87 has a density similar to that of a neutron star, its size would be about 1 millionth of its apparent black hole shadow diameter. This coupled with its low emissivity would make it extremely difficult to detect. However, in theory, it is detectable. The core could also be larger than the neutron star density value because the physical characteristics of matter for ϕ less than -1 is still unknown. Another possibility is that rotation coupled with the near radial requirement for light to escape could facilitate the formation of the radial jets emanating from M87*. If so, these jets could originate within a radius smaller than r_s.

The right-hand plot of Figure 17 illustrates that the shadow diameter calculated with this script file converges and approaches $e \cdot 2r_s$ far away from the galactic center.

Example_8_Apparent_Black_Hole_Shadow_Diameter_and_Emissivity.m Output

To save space, some of the output is omitted. Running the script will create the rest of the output.

```
--- Inputs ---
Example_8_Apparent_Black_Hole_Shadow_Diameter_and_Emissivity

--- Outputs ---
Newtons universal gravitational constant, G = 6.67430e-11 m^3/(kg-s)
The standard speed of light, cs = 299792458.0 m/s
This example calculates the emissivity as a function of radius and the apparent black hole shadow
diameter of Sgr A*

Sgr A* galactic center mass = 3.98e6 Suns
Sgr A* galactic center distance = 7.959 Mpc
Reference speed of light  = 299792464.336 m/s at 2.4558908e20 meters from the galactic center
Initial Position x = 0.00000000000000 y = 11739130131.9995 z = 0.00000000000000 meters
Calculate light path from the photon sphere spiraling into the core

>>> Calculating path with 4960 arc segments, arc angle segments = f(r)
sr = 1e-3 ×
 3.00000000000000

>>> Solution complete for path_option 2
Warning in Lines 161 - 162 of Example_8_Apparent_Black_Hole_Shadow_Diameter_and_Emissivity.m. When
calling plot(x,y,...), imaginary parts of plot data are ignored.
Plot is generated.
…
Legend is shown.
Execution paused for 20.000000 seconds.
Calculate escape path

>>> Calculating path with 3000 arc segments, arc angle segments = f(r)
sr = 1e-3 ×
 6.00000000000000
>>> Solution complete for path_option 2
apparent Black Hole shadow diameter = 2.715 * 2 * rs,  rs is the Schwarzschild radius for which phi =
-1
apparent Black Hole shadow diameter = 6.375e10 meters
apparent Black Hole shadow diameter = 5.354e-5 arcseconds
FOR COMPARISON ONLY - GENERAL RELATIVE apparent Black Hole shadow diameter = 5.123e-5 arcseconds
Plot is generated.
Execution paused for 20.000000 seconds.
Plot is generated.
…
Plot is generated.

Core radius / rs  = 4.907e-4,  This is calculated assuming the density is similar to that of a
neutron star
Core radius / shadow radius = 1.807e-4,  This is calculated assuming the density is similar to that
of a neutron star
Plot is generated.

Calculate the path of motion for light from multiple starting points to illustrate the shadow.
Initial Position x = 414833433556.495 y = 39602668316.9943 z = 0.00000000000000 meters
Calculate solution for initial radial offset of phi = -2.964e-1 and zg = 1.598e-1
>>> Calculating path with 400.0 equal arc length segments
>>> Solution complete for path_option 4
Initial Position x = 414833433556.495 y = 53609243721.1470 z = 0.00000000000000 meters
Calculate solution for initial radial offset of phi = -2.190e-1 and zg = 1.157e-1
…
Plot is unheld.
```

```
--- Inputs ---
```
Example_8_Apparent_Black_Hole_Shadow_Diameter_and_Emissivity

```
--- Outputs ---
Newtons universal gravitational constant, G = 6.67430e-11 m^3/(kg-s)
The standard speed of light, cs = 299792458.0 m/s
```
This example calculates the emissivity as a function of radius and the apparent black hole shadow diameter of M87*
```
M87* galactic center mass = 6.50e9 Suns
M87* galactic center distance = 16.80 Mpc
Reference speed of light  = 299792464.336 m/s at 5.1839383e23 meters from the galactic center
Initial Position x = 0.00000000000000 y = 19196061851068.4 z = 0.00000000000000 meters
Calculate light path from the photon sphere spiraling into the core

>>> Calculating path with 4960 arc segments, arc angle segments = f(r)
sr = 1e-3 ×
 3.00000000000000

>>> Solution complete for path_option 2
Plot is generated.
…
Legend is shown.
Execution paused for 20.000000 seconds.
Calculate escape path

>>> Calculating path with 3000 arc segments, arc angle segments = f(r)
sr = 1e-3 ×
 6.00000000000000
>>> Solution complete for path_option 2
apparent Black Hole shadow diameter = 2.715 * 2 * rs,  rs is the Schwarzschild radius for which phi =
-1
apparent Black Hole shadow diameter = 1.042e14 meters
apparent Black Hole shadow diameter = 4.148e-5 arcseconds
FOR COMPARISON ONLY - GENERAL RELATIVE apparent Black Hole shadow diameter = 3.969e-5 arcseconds
Plot is generated.
Execution paused for 20.000000 seconds.
Plot is generated.
…
Plot is generated.

Core radius / rs  = 3.535e-6,  This is calculated assuming the density is similar to that of a
neutron star
Core radius / shadow radius = 1.302e-6,  This is calculated assuming the density is similar to that
of a neutron star
Plot is generated.

Calculate the path of motion for light from multiple starting points to illustrate the shadow.
Initial Position x = 678343979400557 y = 64759080266783.2 z = 0.00000000000000 meters
Calculate solution for initial radial offset of phi = -2.964e-1 and zg = 1.598e-1
>>> Calculating path with 400.0 equal arc length segments
>>> Solution complete for path_option 4
Initial Position x = 678343979400557 y = 87662914260995.1 z = 0.00000000000000 meters
Calculate solution for initial radial offset of phi = -2.190e-1 and zg = 1.157e-1
…
Plot is unheld.
```

References

[1] Anderson, M. G. Time, Matter, and Gravity, copyright 2004 https://vixra.org/abs/2102.0123 https://vixra.org/pdf/2102.0123v1.pdf

[2] Mashhoon, B. Wave propagation in a gravitational field, Phys. Lett. A 122, number 6, 7 page 299-304 (1987)

[3] Robertson, S. L., Comments on a Concept from General Relativity Prespacetime Journal j June 2019 j Volume 10 j Issue 5 j pp. 621-626

[4] Luminet, Jean-Pierre Seeing Black Holes :from the Computer to the Telescope. April 2018 https://arxiv.org/pdf/1804.03909.pdf

[5] Robertson S. L. Where Einstein Got It Wrong https://vixra.org/abs/1808.0642 https://vixra.org/pdf/1808.0642v1.pdf

[6] Krogh, K. Gravitation Without Curved Space-time https://arxiv.org/abs/astro-ph/9910325v23 https://arxiv.org/pdf/astro-ph/9910325v23.pdf

[7] Ashby, N. Relativity in the Global Positioning System. January 2003 -01-28. https://link.springer.com/article/10.12942/lrr-2003-1

[8] 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

[9] Relativistic redshift of the star S0-2 orbiting the Galactic center supermassive black hole https://arxiv.org/abs/1907.10731

[10] Detection of the Schwarzschild precession in the orbit of the star S2 near the Galactic centre massive black hole https://arxiv.org/abs/2004.07187

[11] Blanco, V. M. and McCuskey, S. W. (1961) Basic Physics Of The Solar System page 218 https://archive.org/details/basicphysicsofso0000blan/page/218/mode/2up

[12] Irwin I. Shapiro Phys. Rev. Lett. 13, 789 – Published 28 December 1964 Fourth Test of General Relativity https://doi.org/10.1103/PhysRevLett.13.789

[13] Irwin I. Shapiro, Gordon H. Pettengill, Michael E. Ash, Melvin L. Stone, William B. Smith, Richard P. Ingalls, and Richard A. Brockelman Phys. Rev. Lett. 20, 1265 – Published 27 May 1968; Erratum Phys. Rev. Lett. 21, 266 (1968). FOURTH TEST OF GENERAL RELATIVITY: PRELIMINARY RESULTS https://doi.org/10.1103/PhysRevLett.20.1265

[14] Irwin I. Shapiro, Michael E. Ash, Richard P. Ingalls, William B. Smith, Donald B. Campbell, Rolf B. Dyce, Raymond F. Jurgens, and Gordon H. Pettengill Phys. Rev. Lett. 26, 1132 – Published 3 May 1971. Fourth Test of General Relativity: New Radar Result
https://doi.org/10.1103/PhysRevLett.26.1132

[15] Shaopeng Tang, et al., 2020-01-08 The Masses of Isolated Neutron Stars Inferred from the Gravitational Redshift Measurements
https://arxiv.org/pdf/1911.08107.pdf

[16] Anderson, M. G. 2007-10-19 A Method for Predicting Quasar Luminosity Consistent With the NASA/IPAC Extragalactic Database
https://www.rxiv.org/abs/2102.0165
https://www.rxiv.org/pdf/2102.0165v1.pdf

[17] First M87 Event Horizon Telescope Results. VI. The Shadow and Mass of the Central Black Hole
https://arxiv.org/pdf/1906.11243.pdf

MatLab Script Files

TMG_solver.m

```
%  Matlab scrip: TMG_solver
%-------------------------------------------------------------------------------------------
%
%  Author:  Morris G. Anderson
%
%  This MatLab script can be used to calculate the path of matter in a gravitational
%  field such as the motion of a ball, the precession of an orbit, and the deflection
%  of light passing near the Sun.
%
%  Method initially developed in FORTRAN and 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
%
%  Modified from FORTRAN to Matlab script in 2021
%
%  Program units
%
%  length = meters
%  time   = seconds
%  mass   = kg
%
%  *** The following must be defined before calling this script ***
%
%  g  = 0.66743e-10 m^3/(kg s), Newton's gravitational constant
%
%  cs = 299792458.0 m/s,  the standard speed of light
%
%  The solver option:
%
%   path_option = 1; % Calculate the precession of an orbit.  The solution is calculated for one
%                       revolution if the max angle (atr) is input as 360.  The solution is calculated
for
%                       two revolutions if the max angle (atr) is input as 2 * 360 and so forth.
%                       All arc segments have the same rotation angle.
%                       The total rotation angle is part of the orbital precession solution.
%
%                 atr = The total rotation angle for calculation, initially an input for the
"TMG_solver",
%                       is overwritten as an output of the "TMG_path" solution.  units = radians
%
%
%   path_option = 2; % Calculate the bending of light and the Shapiro time delay.
%                       Arc segment length is a function of radius from the governing body center.
%
%                       if st > 1, st = The intended maximum path length for the calculation
%
%                       if st < 1, sr = st. Arc segment length = sr * r(n,4) for each calculation
step.
%
%
%   path_option = 3; % Calculate a simple trajectory or orbit with equal arc angle segments
%
%                 atr = The total rotation angle for calculation, initially an input for the
"TMG_solver",
%                       is overwritten as an output of the "TMG_path" solution.  units = radians
%
%
%   path_option = 4; % Calculate a simple trajector with equal arc length segments
%
%                  st = The intended maximum path length for calculation
%
%
%  m   = mass of the governing body.  units = kg
%  rr  = radius from center of m corresponding to cr.  units = m
```

```
%  cr  = the reference speed of light at a distance of rr from m.  units = m/s
%
%  x   = initial x coordinate relative to the governing body m.  units = m
%  y   = initial y coordinate relative to the governing body m.  units = m
%  z   = initial z coordinate relative to the governing body m.  units = m
%
%  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
%
%  np     = number of points stored in memory for the solution which can be plotted
%  nac    = number of arc segments integrated to calculate each point that is stored in memory
%
%-------------------------------------------------------------------------------------------
%  Definitions
%
%  TMG = the book "Time Matter and Gravity"
%
%  ai     = angle of rotation for an individual arc segment
%  atr    = total rotation angle for calculation - initially an input for "TMG_solver", it is
%           then over written as an output of "TMG_path"
%  b      = motion vector. 1 = i, 2 = j, 3 = k, 4 = magnitude of v / c
%  c      = speed of light
%  cr     = the reference speed of light at a distance of rr from m
%  cs     = standard speed of light (see TMG Definition 10)
%  dot_br = the dot product between the motion unit vector and the position vector at point n+1.
%           If equal to zero, the vectors are perpendicular to each other.
%  f2x    = An output of TMG_path.m.  It is used to calculate the time-averaged frequency
%           shift (see TMG eq. 5-29)
%  g      = Newton's gravitational constant
%  k      = rotation vector
%  m      = mass of the gravitational body
%  np     = number of points stored in memory for the solution which can be plotted
%  nac    = number of arc segments integrated to calculate each point that is stored in memory
%  ni     = total number of arc segments calculated to obtain the solution.  ni = np * nac
%  r      = position vector with respect to the governing body m.
%           1 = i, 2 = j, 3 = k, 4 = magnitude of r, 5 = x, 6 = y, 7 = z,
%           8 = rotation angle in degrees from start, 9 = time from start in seconds
%  rc     = curvature vector. 1 = i, 2 = j, 3 = k, 4 = magnitude of rc
%  rr     = radius from center of m corresponding to cr
%  sr     = (for path_option = 2), maximum path length / r(n,4) for each calculation step
%           (for path_option = 4), maximum path length for each calculation step
%  s      = calculated path length
%  st     = (for path_option = 2) if st > 1, st = The intended maximum path length for the
calculation
%           (for path_option = 2) if st < 1, sr = st. Arc segment length = sr * r(n,4) for each
calculation step
%  t      = path time
%  tol    = convergence tolerance
%  v      = velocity
%  vt     = temporary vector for calculations
%
%-------------------------------------------------------------------------------------------
%  Preallocate arrays to reduce runtime

b  = zeros(np,4);
c  = zeros(np,1);
r  = zeros(np,9);
rc = zeros(np,4);

%  Initialize values
ai = 0.0;
ni = (np * nac);
sr = 0.0;

%-------------------------------------------------------------------------------------------
%  Calculate initial values for starting point
%-------------------------------------------------------------------------------------------

r(1,5) = x;
r(1,6) = y;
r(1,7) = z;
```

```
r(1,4) = sqrt(  x*x +  y*y + z*z  );
r(1,1) = x  /  r(1,4);
r(1,2) = y  /  r(1,4);
r(1,3) = z  /  r(1,4);

c(1,1) = cr * exp( (2*m*g/cs/cs ) * ( ( r(1,4)-rr ) / (rr*r(1,4))) );

%   calculate beta

if beta_option == 1
    b(1,4) = sqrt(  bx*bx + by*by + bz*bz );
    b(1,1) = bx/b(1,4);
    b(1,2) = by/b(1,4);
    b(1,3) = bz/b(1,4);
else
        v  = sqrt(  vx*vx + vy*vy + vz*vz );
    b(1,4) = v  /  c(1);
    b(1,1) = (  vx  /  c(1)  )  /  b(1,4);
    b(1,2) = (  vy  /  c(1)  )  /  b(1,4);
    b(1,3) = (  vz  /  c(1)  )  /  b(1,4);
end

% ------------------------------------------------------------------------------------------
%   path_option = 1, Calculate the precession of an orbit.  The solution is calculated for one
%                    revolution if the max angle (atr) is input as 360.  The solution is calculated
for
%                    two revolutions if the max angle (atr) is input as 2 * 360 and so forth.
%                    All arc segments have the same rotation angle.
%                    The total rotation angle is part of the orbital precession solution.
%
%               atr = The total rotation angle for calculation, initially an input for the
"TMG_solver",
%                    is overwritten as an output of the "TMG_path" solution.  units = radians
% ------------------------------------------------------------------------------------------

if path_option == 1

    tol    = 1.0e-12;

%  sr  = maximum path length / r(n,4) per calculation step.  If = 0.0 there is no constraint

    %sr = 0;

    ai = atr/ni;

    disp(['>>> Calculating orbital precession with ' num2str(np*nac) ' equal arc angle segments'])

    nmax   = 10;

    x0 = ai;

    %   Calculate the 1st rotation guess with the TMG_path.m script which returns dot_br
    %   which is the dot product between the motion unit vector and the position vector
    %   at point n+1.  If equal to zero, the vectors are perpendicular to each other.

    [atr, dot_br, f2x, s, t, b, c, r, rc] = TMG_path(path_option, ai, cs, g, m, nac, np, sr, b, c, r,
tol);

    f0 = dot_br;

    ai = ai * 1.01;

    x1 = ai;

    %   Calculate the 2nd rotation guess.

    [atr, dot_br, f2x, s, t, b, c, r, rc] = TMG_path(path_option, ai, cs, g, m, nac, np, sr, b, c, r,
tol);

    f1 = dot_br;

    %  Apply the Newton-Raphson method to solve for the angle of rotation required to bring the
    %  planet back to its nearest location to the governing body.  This happens when its direction
    %  of motion is perpendicular to its position vector relative to the governing body.

    for jj=1:nmax
```

```
        disp(['Solving, iteration = ' num2str(jj) ])

        x = x1 - f1 * ( x1 - x0 ) / ( f1 - f0 );

        if ( abs(( x - x1 )/x1) < tol )
            break
        else
            x0 = x1;
            x1 = x;
            f0 = f1;

            ai = x1;

            [atr, dot_br, f2x, s, t, b, c, r, rc] = TMG_path(path_option, ai, cs, g, m, nac, np, sr,
b, c, r, tol);

            f1 = dot_br;
        end

    end

    if jj > nmax-1

        disp('solution failed to converge for option 1')
    end

    disp('>>> Solution complete for path_option 1')

end

% ------------------------------------------------------------------------------------------
%   path_option = 2; % Calculate the bending of light and the Shapiro time delay.
%                      Arc segment length is a function of radius from the governing body center.
%
%                      if st > 1, st = The intended maximum path length for the calculation
%
%                      if st < 1, sr = st. Arc segment length = sr * r(n,4) for each calculation
step.
% ------------------------------------------------------------------------------------------

if( path_option == 2 )

    tol    = 1.0e-6;

    disp(['>>> Calculating path with ' num2str(np*nac) ' arc segments, arc angle segments = f(r)'])

    if st > 1


    nmax = 50;

    %  sr  = maximum path length / r(n,4) for each calculation step.

    ai = 2.0 * pi/ni;

    sr = (1/ni);

    x0 = sr;

    [atr, dot_br, f2x, s, t, b, c, r, rc] = TMG_path(path_option, ai, cs, g, m, nac, np, sr, b, c, r,
tol);

    f0 = 1 - st / s;

    sr = sqrt(1/ni);

    x1 = sr;

    [atr, dot_br, f2x, s, t, b, c, r, rc] = TMG_path(path_option, ai, cs, g, m, nac, np, sr, b, c, r,
tol);


    f1 = 1 - st / s;

    for jj=1:nmax;

        disp(['Solving, iteration = ' num2str(jj) ])
```

```
        x = x1 - f1 * ( x1 - x0 ) / ( f1 - f0 );

        if ( abs(( x - x1 )/x1) < tol)
            break
        else

            if (x < 0)
               %sr = 1/ni;
                x = x1/2;
            end

            x0 = x1;
            x1 = x;
            f0 = f1;

            sr = x1;


            [atr, dot_br, f2x, s, t, b, c, r, rc] = TMG_path(path_option, ai, cs, g, m, nac, np, sr,
b, c, r, tol);

            f1 = 1 - st / s;
        end

    end


    if jj > nmax-1

        disp('solution failed to converge for option 2')
    end

    else

        sr = st

        [atr, dot_br, f2x, s, t, b, c, r, rc] = TMG_path(path_option, ai, cs, g, m, nac, np, sr, b,
c, r, tol);
    end


    disp('>>> Solution complete for path_option 2')

end

% ------------------------------------------------------------------------------------------
%   path_option = 3, Calculate a simple trajectory with equal arc angle segments
%
%               atr = The total rotation angle for calculation, initially an input for the
"TMG_solver",
%                     is overwritten as an output of the "TMG_path" solution.  units = radians
% ------------------------------------------------------------------------------------------

if( path_option == 3)

    tol    = 1.0e-12;

    disp(['>>> Calculating path with ' num2str(np*nac) ' equal arc angle segments'])

    ai = atr / ni;

    [atr, dot_br, f2x, s, t, b, c, r, rc] = TMG_path(path_option, ai, cs, g, m, nac, np, sr, b, c, r,
tol);

    disp('>>> Solution complete for path_option 3')

end

% ------------------------------------------------------------------------------------------
%   path_option = 4, Calculate a simple trajectory with equal arc length segments
%
%                st = The intended maximum path length for calculation
% ------------------------------------------------------------------------------------------

if( path_option == 4)

    tol    = 1.0e-12;
```

```
    disp(['>>> Calculating path with ' num2str(np*nac) ' equal arc length segments'])

    %  sr  = maximum path length for each calculation step.

    sr = st/ni;

    [atr, dot_br, f2x, s, t, b, c, r, rc] = TMG_path(path_option, ai, cs, g, m, nac, np, sr, b, c, r,
tol);

    disp('>>> Solution complete for path_option 4')

end
```

TMG_path.m

```
function [atr, dot_br, f2x, s, t, b, c, r, rc] = TMG_path(path_option, ai, cs, g, m, nac, np, sr, b,
c, r, tol)
%-----------------------------------------------------------------------------------------
%
%  Author:  Morris G. Anderson
%
%  This function is called by the TMG_solver.m script to calculate the path of matter in a
%  gravitational field such as the motion of a ball, the precession of an orbit, and the
%  deflection of light passing near the Sun.
%
%  Method initially developed in FORTRAN and 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
%
%  Modified from FORTRAN to Matlab script in 2021
%
%  Program units
%
%  length = meters
%  time   = seconds
%  mass   = kg
%
%  *** The following must be defined before calling this script ***
%
%  g  = 0.66743e-10 m^3/(kg s), Newton's gravitational constant
%
%  cs = 299792458.0 m/s,  the standard speed of light
%
%  The solver option:
%
%   path_option = 1; % Calculate the precession of an orbit.  The solution is calculated for one
%                      revolution if the max angle (atr) is input as 360.  The solution is calculated
for
%                      two revolutions if the max angle (atr) is input as 2 * 360 and so forth.
%                      All arc segments have the same rotation angle.
%                      The total rotation angle is part of the orbital precession solution.
%
%                 atr = The total rotation angle for calculation, initially an input for the
"TMG_solver",
%                      is overwritten as an output of the "TMG_path" solution.  units = radians
%
%
%   path_option = 2; % Calculate the bending of light and the Shapiro time delay.
%                      Arc segment length is a function of radius from the governing body center.
%
%                      if st > 1, st = The intended maximum path length for the calculation
%
%                      if st < 1, sr = st. Arc segment length = sr * r(n,4) for each calculation
step.
%
%
%   path_option = 3; % Calculate a simple trajectory or orbit with equal arc angle segments
%
%                 atr = The total rotation angle for calculation, initially an input for the
"TMG_solver",
%                      is overwritten as an output of the "TMG_path" solution.  units = radians
%
%
%   path_option = 4; % Calculate a simple trajector with equal arc length segments
%
%                  st = The intended maximum path length for calculation
%
%
%  m   = mass of the governing body.  units = kg
%  rr  = radius from center of m corresponding to cr.  units = m
%  cr  = the reference speed of light at a distance of rr from m.  units = m/s
%
%  x   = initial x coordinate relative to the governing body m.  units = m
%  y   = initial y coordinate relative to the governing body m.  units = m
```

```
%  z   = initial z coordinate relative to the governing body m.  units = m
%
%  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
%
%  np     = number of points stored in memory for the solution which can be plotted
%  nac    = number of arc segments integrated to calculate each point that is stored in memory
%
%-------------------------------------------------------------------------------------------
%  Definitions
%
%  TMG = the book "Time Matter and Gravity"
%
%  ai     = angle of rotation for an individual arc segment
%  atr    = total rotation angle for calculation - initially an input for "TMG_solver", it is
%           then over written as an output of "TMG_path"
%  b      = motion vector. 1 = i, 2 = j, 3 = k, 4 = magnitude of v / c
%  c      = speed of light
%  cr     = the reference speed of light at a distance of rr from m
%  cs     = standard speed of light (see TMG Definition 10)
%  dot_br = the dot product between the motion unit vector and the position vector at point n+1.
%           If equal to zero, the vectors are perpendicular to each other.
%  f2x    = An output of TMG_path.m.  It is used to calculate the time-averaged frequency
%           shift (see TMG eq. 5-29)
%  g      = Newton's gravitational constant
%  k      = rotation vector
%  m      = mass of the gravitational body
%  np     = number of points stored in memory for the solution which can be plotted
%  nac    = number of arc segments integrated to calculate each point that is stored in memory
%  ni     = total number of arc segments calculated to obtain the solution.  ni = np * nac
%  r      = position vector with respect to the governing body m.
%           1 = i, 2 = j, 3 = k, 4 = magnitude of r, 5 = x, 6 = y, 7 = z,
%           8 = rotation angle in degrees from start, 9 = time from start in seconds
%  rc     = curvature vector. 1 = i, 2 = j, 3 = k, 4 = magnitude of rc
%  rr     = radius from center of m corresponding to cr
%  sr     = (for path_option = 2), maximum path length / r(n,4) for each calculation step
%           (for path_option = 4), maximum path length for each calculation step
%  s      = calculated path length
%  st     = (for path_option = 2) if st > 1, st = The intended maximum path length for the
calculation
%           (for path_option = 2) if st < 1, sr = st. Arc segment length = sr * r(n,4) for each
calculation step
%  t      = path time
%  tol    = convergence tolerance
%  v      = velocity
%  vt     = temporary vector for calculations
%
%-------------------------------------------------------------------------------------------

atr = 0;

f2xt = 0;

s = 0;
t = 0;
n = 1;

b1(1,:) = b(1,:);
c1(1)   = c(1);
r1(1,:) = r(1,:);

%  Calculate the 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 vectors.
%  For a single governing body, the gradient of c is radially outward from the governing body.
%  Therefore, it is in the same direction as the position vector r.

kr1  = cross( r1(1,1:3), b1(1,1:3) );

kr1 = kr1 / sqrt( kr1(1,1)*kr1(1,1) + kr1(1,2)*kr1(1,2)+ kr1(1,3)*kr1(1,3) );

%  Calculate the direction 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 with the rotation vector.

rc1(1,1:3) = cross( b1(1,1:3), kr1(1,1:3) );

%   See TMG appendix A Figure 21 for numerical approach.
%   Use the Newton-Raphson method to solve for the path radius of curvature
%   Calculate path radius of curvature at point 1, see TMG eq 7-12


%  Check on Rc to prevent divide by zero.  Depending on check, rc is set very
%  large to simulate straight-line motion

rdotrc1 = dot( r1(1,1:3), rc1(1,1:3) );

if rdotrc1 > 1e-50

    rc1(1,4) = ( r1(1,4)^2 * cs^2 ) / (  ( 1 + 1 / b1(1,4)^2 ) * m * g * rdotrc1 );
else
    rc1(1,4) = 1e50;
end

rc(1,:) = rc1(1,:);


for n = 1:np

    for arc_count = 1:nac

        %   1st estimate of rca, the average radius of curvature between point 1 and 2

        rca  = rc1(1,4);

        xrca0 = rca;

        %  Check on the limit of path length for iteration

        if path_option == 2
            if rca > r1(1,4)
                ai = (sr * r1(1,4) ) / rca;
            else
                ai = sr;
            end
        end

        if path_option == 4
            ai = sr / rca;
            else
        end

        %  Rotate unit vectors with Rodrigues' formula to calculate position and motion at point 2

        b2(1,1:3) =  b1(1,1:3) * cos(ai)  +  cross( kr1(1,1:3),  b1(1,1:3) ) * sin(ai)  +  kr1(1,1:3)
* dot( kr1(1,1:3), b1(1,1:3) ) * ( 1 - cos(ai) );

        b2(1,1:3) =  b2(1,1:3) / sqrt( b2(1,1)*b2(1,1) + b2(1,2)*b2(1,2) + b2(1,3)*b2(1,3) );

        %rc2(1,1:3) = rc1(1,1:3) * cos(ai)  +  cross( kr1(1,1:3), rc1(1,1:3) ) * sin(ai)  +
kr1(1,1:3) * dot( kr1(1,1:3), rc1(1,1:3) ) * ( 1 - cos(ai) );

        rc2(1,1:3) = cross( b2(1,1:3), kr1(1,1:3) );

        rc2(1,1:3) =  rc2(1,1:3) / sqrt( rc2(1,1)*rc2(1,1) + rc2(1,2)*rc2(1,2) + rc2(1,3)*rc2(1,3) );

        %r2(1,5:7) = r1(1,5:7) + rca * ( rc2(1,1:3)  - rc1(1,1:3) );
        %r2(1,4) = sqrt( r2(1,5)*r2(1,5) +  r2(1,6)*r2(1,6)  +  r2(1,7)*r2(1,7) );
        %r2(1,1:3) = r2(1,5:7) / r2(1,4);
        % Replaced with the following to reduce numerical errors associated with very large rc

        vt(1:3) = b1(1,1:3) + b2(1,1:3);

        vt(4)     = sqrt( vt(1)*vt(1) + vt(2)*vt(2) + vt(3)*vt(3) );

        vt(1:3)   = vt(1:3) / vt(4);

        vt(4)     = 2.0 * rca * sin(ai/2.0);


        r2(1,5:7) = r1(1,5:7) + vt(1:3) * vt(4);
```

```
r2(1,4)   = sqrt( r2(1,5)*r2(1,5) +  r2(1,6)*r2(1,6)  +  r2(1,7)*r2(1,7) );

r2(1,1:3) = r2(1,5:7) / r2(1,4);


%  ------------------------------------
%  Calculate conditions for point 2
%  ------------------------------------

%  Calculate the speed of light at point 2 (see TMG equations 4-10 through 4-14)

g1 = ( 2*m*g/cs/cs )  *  ( r2(1,4) - r(1,4) ) / ( r2(1,4) * r(1,4) );

%  Choose the best method to minimize numerical errors

if abs(g1) > 0.1

    c2(1,1) = c(1,1) * exp(g1);

    dc = c(1,1) - c2(1,1);

else

    g2 = g1 * g1;
    g3 = g2 * g1;
    g4 = g3 * g1;
    g5 = g4 * g1;
    g6 = g5 * g1;
    g7 = g6 * g1;
    g8 = g7 * g1;

    dc = c(1,1)*(-g1 -g2/2. -g3/6. -g4/24. -g5/120.0 -g6/720. -g7/5040. -g8/40320);

    c2(1,1) = c(1,1) - dc;

end

%  Calculate beta at point 2 (see TMG equations 4-18)

b2(1,4) = sqrt( (dc + b(1,4)*b(1,4) * c2(1,1)) / c(1,1) );

%   Calculate path radius of curvature at point 2, see TMG eq 7-12
%   Check on Rc to prevent divide by zero.  Depending on check, rc is set very
%   large to simulate straight-line motion

rdotrc2= dot( r2(1,1:3), rc2(1,1:3) );

if rdotrc2>1e-50
    rc2(1,4) = ( r2(1,4)^2 * cs^2 ) / (  ( 1 + 1 / b2(1,4)^2 ) * m * g * rdotrc2 );
else
    rc2(1,4) = 1e50;
end



frca0 = rca - ( (rc1(1,4) + rc2(1,4)  ) / 2);


if  abs( frca0 ) > tol

    rca = (  rc1(1,4) + rc2(1,4)  ) / 2;

    xrca1 = rca;

    if path_option == 2
        if rca > r1(1,4)
            ai = (sr * r1(1,4) ) / rca;
        else
            ai = sr;
        end
    end

    if path_option == 4
        ai = sr / rca;
    else
    end
```

```
            %  Rotate unit vectors with Rodrigues' formula to calculate position and motion at point
2

            b2(1,1:3) =  b1(1,1:3) * cos(ai)  +  cross( kr1(1,1:3),  b1(1,1:3) ) * sin(ai)  +
kr1(1,1:3) * dot( kr1(1,1:3), b1(1,1:3) ) * ( 1 - cos(ai) );

            b2(1,1:3) =  b2(1,1:3) / sqrt( b2(1,1)*b2(1,1) + b2(1,2)*b2(1,2) + b2(1,3)*b2(1,3) );

            %rc2(1,1:3) = rc1(1,1:3) * cos(ai)  +  cross( kr1(1,1:3), rc1(1,1:3) ) * sin(ai)  +
kr1(1,1:3) * dot( kr1(1,1:3), rc1(1,1:3) ) * ( 1 - cos(ai) );

            rc2(1,1:3) = cross( b2(1,1:3), kr1(1,1:3) );

            rc2(1,1:3) =  rc2(1,1:3) / sqrt( rc2(1,1)*rc2(1,1) + rc2(1,2)*rc2(1,2) +
rc2(1,3)*rc2(1,3) );

            %r2(1,5:7) = r1(1,5:7) + rca * ( rc2(1,1:3)  - rc1(1,1:3) );
            %r2(1,4) = sqrt( r2(1,5)*r2(1,5) +  r2(1,6)*r2(1,6)  +  r2(1,7)*r2(1,7) );
            %r2(1,1:3) = r2(1,5:7) / r2(1,4);
            % Replaced with the following to reduce numerical errors associated with very large rc

            vt(1:3) = b1(1,1:3) + b2(1,1:3);

            vt(4)     = sqrt( vt(1)*vt(1) + vt(2)*vt(2) + vt(3)*vt(3) );

            vt(1:3)   = vt(1:3) / vt(4);

            vt(4)     = 2.0 * rca * sin(ai/2.0);


            r2(1,5:7) = r1(1,5:7) + vt(1:3) * vt(4);

            r2(1,4)   = sqrt( r2(1,5)*r2(1,5) +  r2(1,6)*r2(1,6)  +  r2(1,7)*r2(1,7) );

            r2(1,1:3) = r2(1,5:7) / r2(1,4);


            %  ------------------------------------
            %  Calculate conditions for point 2
            %  ------------------------------------

            %  Calculate the speed of light at point 2 (see TMG equations 4-10 through 4-14)

            g1 = ( 2*m*g/cs/cs )  *  ( r2(1,4) - r(1,4) ) / ( r2(1,4) * r(1,4) );

            %  Choose the best method to minimize numerical errors

            if abs(g1) > 0.1

                c2(1,1) = c(1,1) * exp(g1);

                dc = c(1,1) - c2(1,1);

            else

                g2 = g1 * g1;
                g3 = g2 * g1;
                g4 = g3 * g1;
                g5 = g4 * g1;
                g6 = g5 * g1;
                g7 = g6 * g1;
                g8 = g7 * g1;

                dc = c(1,1)*(-g1 -g2/2. -g3/6. -g4/24. -g5/120.0 -g6/720. -g7/5040. -g8/40320);

                c2(1,1) = c(1,1) - dc;

            end

            %  Calculate beta at point 2 (see TMG equations 4-18)

            b2(1,4) = sqrt( (dc + b(1,4)*b(1,4) * c2(1,1)) / c(1,1) );


            %   Calculate path radius of curvature at point 2, see TMG eq 7-12
            %   Check on Rc to prevent divide by zero.  Depending on check, rc is set very
            %   large to simulate straight-line motion
```

```
            rdotrc2= dot( r2(1,1:3), rc2(1,1:3) );

            if rdotrc2>1e-50
                rc2(1,4) = ( r2(1,4)^2 * cs^2 ) / (  ( 1 + 1 / b2(1,4)^2 ) * m * g * rdotrc2 );
            else
                rc2(1,4) = 1e50;
            end


            frca1 = rca - (  (rc1(1,4) + rc2(1,4)  ) / 2.0 );


            for count = 1:20;

                if count > 19

                    disp(['solution failed to converge for rc at n = ' num2str(count) ' steps' ])
                    disp(' ')

                    break
                else
                end

                xrca = xrca1 - frca1 * ( xrca1 - xrca0 ) / ( frca1 - frca0 );

                if  abs(   ( xrca - xrca1 ) / xrca1   ) < tol


                    si = ai * rca;
                    s  = s + si;

                    dt = si / ( (b1(1,4)*c1(1,1) + b2(1,4)*c2(1,1)) / 2.0 );
                    t  = t + dt;

                    %  f2xt is used to calculate the time-averaged frequency shift, see TMG eq. 5-29

                    f2xt = f2xt + dt * sqrt( ( (c1(1,1) + c2(1,1))/2 ) * ( 1 - ((b1(1,4) + b2(1,4))/
2)^2 ) );

                    atr = atr + ai;

                    c1(1,1) = c2(1,1);

                    r2(1,8) = atr*180/pi;
                    r2(1,9) = t;

                    break
                else
                    xrca0 = xrca1;
                    xrca1 = xrca;
                    frca0 = frca1;
                    rca   = xrca1;
                    end

                if path_option == 2

                    if rca > r1(1,4)
                        ai = (sr * r1(1,4) ) / rca;
                    else
                        ai = sr;
                    end
                end

                if path_option == 4
                    ai = sr / rca;
                else
                end

                %  Rotate unit vectors with Rodrigues' formula to calculate position and motion at
point 2

                b2(1,1:3) =  b1(1,1:3) * cos(ai)  +  cross( kr1(1,1:3),  b1(1,1:3) ) * sin(ai)  +
kr1(1,1:3) * dot( kr1(1,1:3), b1(1,1:3) ) * ( 1 - cos(ai) );

                b2(1,1:3) =  b2(1,1:3) / sqrt( b2(1,1)*b2(1,1) + b2(1,2)*b2(1,2) + b2(1,3)*b2(1,3) );

                %rc2(1,1:3) = rc1(1,1:3) * cos(ai)  +  cross( kr1(1,1:3), rc1(1,1:3) ) * sin(ai)  +
kr1(1,1:3) * dot( kr1(1,1:3), rc1(1,1:3) ) * ( 1 - cos(ai) );
```

```
                rc2(1,1:3) = cross( b2(1,1:3), kr1(1,1:3) );

                rc2(1,1:3) =  rc2(1,1:3) / sqrt( rc2(1,1)*rc2(1,1) + rc2(1,2)*rc2(1,2) +
rc2(1,3)*rc2(1,3) );

                %r2(1,5:7) = r1(1,5:7) + rca * ( rc2(1,1:3)  - rc1(1,1:3) );
                %r2(1,4) = sqrt( r2(1,5)*r2(1,5) +  r2(1,6)*r2(1,6)  +  r2(1,7)*r2(1,7) );
                %r2(1,1:3) = r2(1,5:7) / r2(1,4);
                % Replaced with the following to reduce numerical errors associated with very large
rc

                vt(1:3) = b1(1,1:3) + b2(1,1:3);

                vt(4)     = sqrt( vt(1)*vt(1) + vt(2)*vt(2) + vt(3)*vt(3) );

                vt(1:3)   = vt(1:3) / vt(4);

                vt(4)     = 2.0 * rca * sin(ai/2.0);


                r2(1,5:7) = r1(1,5:7) + vt(1:3) * vt(4);

                r2(1,4)   = sqrt( r2(1,5)*r2(1,5) +  r2(1,6)*r2(1,6)  +  r2(1,7)*r2(1,7) );

                r2(1,1:3) = r2(1,5:7) / r2(1,4);

                %  ------------------------------------
                %  Calculate conditions for point 2
                %  ------------------------------------

                %  Calculate the speed of light at point 2 (see TMG equations 4-10 through 4-14)

                g1 = ( 2*m*g/cs/cs )  *  ( r2(1,4) - r(1,4) ) / ( r2(1,4) * r(1,4) );

                %  Choose the best method to minimize numerical errors

                if abs(g1) > 0.1

                    c2(1,1) = c(1,1) * exp(g1);

                    dc = c(1,1) - c2(1,1);

                else

                    g2 = g1 * g1;
                    g3 = g2 * g1;
                    g4 = g3 * g1;
                    g5 = g4 * g1;
                    g6 = g5 * g1;
                    g7 = g6 * g1;
                    g8 = g7 * g1;

                    dc = c(1,1)*(-g1 -g2/2. -g3/6. -g4/24. -g5/120.0 -g6/720. -g7/5040. -g8/40320);

                    c2(1,1) = c(1,1) - dc;

                end

                %  Calculate beta at point 2 (see TMG equations 4-18)

                b2(1,4) = sqrt( (dc + b(1,4)*b(1,4) * c2(1,1)) / c(1,1) );

                %   Calculate path radius of curvature at point 2, see TMG eq 7-12
                %   Check on Rc to prevent divide by zero.  Depending on check, rc is set very
                %   large to simulate straight-line motion

                rdotrc2= dot( r2(1,1:3), rc2(1,1:3) );

                if rdotrc2>1e-50
                    rc2(1,4) = ( r2(1,4)^2 * cs^2 ) / (  ( 1 + 1 / b2(1,4)^2 ) * m * g * rdotrc2 );
                else
                    rc2(1,4) = 1e50;
                end


                frca1 = rca - (  (rc1(1,4) + rc2(1,4)  ) / 2  );
```

```
            end

        else

            si = ai * rca;
            s  = s + si;

            dt = si / ( (b1(1,4)*c1(1,1) + b2(1,4)*c2(1,1)) / 2.0 );
            t  = t + dt;

            %  f2xt is used to calculate the time-averaged frequency shift, see TMG eq. 5-29

            f2xt = f2xt + dt * sqrt( ( (c1(1,1) + c2(1,1))/2 ) * ( 1 - ((b1(1,4) + b2(1,4))/2)^2 ) );

            atr = atr + ai;

            c1(1,1) = c2(1,1);

            r2(1,8) = atr*180/pi;
            r2(1,9) = t;

        end

        %  End of inner loop for rca

        b1  = b2;
        r1  = r2;
        rc1 = rc2;

    end

    b(n+1,:)  = b2(1,:);
    c(n+1,1)  = c2(1,1);
    r(n+1,:)  = r2(1,:);
    rc(n+1,:) = rc2(1,:);

end

b1  = b(1,:);
r1  = r(1,:);
rc1 = rc(1,:);
f2x = f2xt / t;

%   Calculate the dot product between the motion unit vector and the position vector at point n+1.
%   If equal to zero, the vectors are perpendicular to each other.

dot_br = b(n+1,1)*r(n+1,5) + b(n+1,2)*r(n+1,6) + b(n+1,3)*r(n+1,7);

end
```

Example_1_Simple_Trajectory.m

```
%   Example_1_Simple_Trajectory.m
%
%   MatLab script for calculating a low-speed trajectory and comparing it with Newtons method
%   based on a constant acceleration of gravity
%
%
%   Copyright 2021 Morris G. Anderson
%
%
%   This script will be documented in the following paper to be ePublished on https://vixra.org
%
%
%   Curvature of Motion in a Gravitational Field
%                       by
%               Morris G. Anderson
%
%
%   It will also be uploaded to https://www.mathworks.com/matlabcentral/fileexchange under the same
name.
%
%
%   Information source
%   1)  Time Matter and Gravity (TMG) by Morris G. Anderson provides a derivation of the
%       equations used in this calculation.
%       https://vixra.org/abs/2102.0123
%       https://vixra.org/pdf/2102.0123v1.pdf
%
%   2)  Physical constants
%       https://physics.nist.gov/cuu/pdf/wall_2018.pdf
%
%   3)  Astronomical data
%       https://ssd.jpl.nasa.gov/horizons.cgi
%
%
%   The path of motion is calculated with the TMG_path.m MatLab function.  The TMG_solver.m
%   script applies the TMG_path.m function to obtain solutions for:
%
%   path_option = 1; % Calculate the precession of an orbit.  The solution is calculated for one
%                       revolution if the max angle (atr) is input as 360.  The solution is calculated
for
%                       two revolutions if the max angle (atr) is input as 2 * 360 and so forth.
%                       All arc segments have the same rotation angle.
%                       The total rotation angle is part of the orbital precession solution.
%
%                atr = The total rotation angle for calculation, initially an input for the
"TMG_solver",
%                       is overwritten as an output of the "TMG_path" solution.  units = radians
%
%
%   path_option = 2; % Calculate the bending of light and the Shapiro time delay for a given path
length.
%                       Arc segment length is a function of radius from the governing body center.
%
%                st =  The intended maximum path length for the calculation
%
%
%   path_option = 3; % Calculate a simple trajectory or orbit with equal arc angle segments
%
%                atr = The total rotation angle for calculation, initially an input for the
"TMG_solver",
%                       is overwritten as an output of the "TMG_path" solution.  units = radians
%
%
%   path_option = 4; % Calculate a simple trajectory with equal arc length segments
%
%                 st = The intended maximum path length for calculation
%
%
%
%   Process for this example:
%
%     1)  Define the universal gravitational constant, g, and the standard speed of light, cs.
%         o  Define the mass of the Earth.
%         o  Define the radius of the Earth.
%
```

```
%     2)  Define the reference speed of light, cr, equal to cs and the reference radius,
%         rr, equal to re.  This is used for calculating the speed of light as a function of
%         position from the Earth.
%
%     3)  Define the mass of the governing body, m, to be equal to the mass of the Earth.
%
%     4)  Define the initial position and velocity for the calculation.
%
%     5)  Define total rotation, number of points, and arc segments per point for this calculation
%
%     6)  Define path_option = 3 for calculating a simple trajectory.
%
%     7)  Call TMG_solver.m which in turn calls TMG_path.m to calculate the path for this case.
%
%     8)  Post-process and plot the results.
%
%     9)  Calculate the trajectory using Newton's gravitational force law and compare results
%         with the TMG solution.

clear
clc

%  g, universal gravitational constant. Units = m^3/(kg-s)

g  = 0.66743e-10;

%   The standard speed of light, cs.  This is on the Earth at re.
%   see TMG section 2.4.1 The Speed of light at the standard location.

cs = 299792458.0;

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

%  The mass and surface radius of the Earth.  From NASA JPL HORIZONS Web-Interface
%  https://ssd.jpl.nasa.gov/horizons.cgi

m_Earth = 398600.435436e9/g;
r_Earth = 6371010.0;

disp(['Earth mass = ' num2str(m_Earth,6) ' kg,  Earth volumetric mean radius = ' num2str(r_Earth, 9)
' m'])

%   For this calculation, define cr = cs

cr = cs;

rr = r_Earth;

disp(['Reference speed of light  = ' num2str(cr,12) ' m/s at ' num2str(rr,8) ' meters from the Earths
center of mass'])

%   Define m, This is the mass of the governing body in kilograms.  For this example,
%   it is the mass of the Earth.

m = m_Earth;

%   Define the initial x, y, z, coordinates with respect to the center of mass of the
%   governing body, m, in meters.

x = 0;
y = r_Earth;
z = 0;

disp(['Initial Position x = ' num2str(x,15) ' y = ' num2str(y,15) ' z = ' num2str(z,15) ' meters' ])


%   Define vx, vy, vz, Initial velocity in m/s.  If beta_option = 1, these values are treated as
%   bx=vx/c, by=vy/c, bz=vz/c

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

beta_option = 0;
vx = v*cos(alpha*pi/180);
vy = v*sin(alpha*pi/180);
vz = 0;
```

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

%   Define total rotation, number of points, and arc segments per point for this calculation

atr = 2 * alpha + (90-alpha)*0.5;
np  = 15;
nac = 30;

disp(['Total rotation angle for calculation = ' num2str(atr,4) ' degrees' ])
disp(['Number of solution points saved in memory for plotting = ' num2str(np) ])
disp(['Number of arc segments calculated for each point       = ' num2str(nac) ])


%     Calculate the path of motion with the TMG solver.

path_option = 3; % Calculate a simple trajectory or orbit with equal arc angle segments

%  This can also be solved with option 4
%  path_option = 4; % Calculate a simple trajectory with equal arc length segments
%  st= 600; % The intended maximum path length for calculation

disp('*** Calculating a simple trajectory ***')

% convert total rotation angle to radians for calculation

atr = atr * pi / 180;

TMG_solver


%     Display the results

disp(['Final Position x = ' num2str(r((np+1),5),15) ' y = ' num2str(r((np+1),6),15) ' z = '
num2str(r((np+1),7),15) ' meters' ])

EucD = sqrt( (r((np+1),5) - r(1,5))^2 + (r((np+1),6) - r(1,6))^2 +(r((np+1),7) - r(1,7))^2);

disp(['Euclidian distance between beginning and end = ' num2str(EucD,15) ' meters'])

disp(['Path length = ' num2str((s),15) ' meters'])
disp(['Path   time = ' num2str(t,15)   ' seconds' ])

%  Calculate the Angle between the beginning and ending directions

ab = b(np+1,1)*b(1,1) + b(np+1,2)*b(1,2) + b(np+1,3)*b(1,3);

ab = acos(ab)*180/pi;

disp(['Angle between beginning and ending directions = ' num2str(ab) ' degrees'])

disp(['Trajectory radius: Min = ' num2str(min(r(:,4)),15) ', Max = ' num2str(max(r(:,4)),15) '
meters' ])

%   Calculate results using Newtons method based on a constant acceleration of gravity

ag= m*g/y/y;

dtn = t / (np*nac);

xn(1) = x;
yn(1) = y;
tn = 0;

for nn=2:(np*nac)+1;
    tn = tn + dtn;
    xn(nn) = xn(1) + vx * tn;
    yn(nn) = yn(1) + vy * tn - 0.5 * ag * tn * tn;
end


vtmg = b(np+1,4) * c(np+1,1);
vtn = sqrt(vx*vx + (vy - ag*t) * (vy - ag*t) );

disp(['Plot results relative to starting point' ])

plot(r(:,5)-x,(r(:,6)-y),'o','MarkerSize',7,'LineStyle','none','MarkerEdgeColor',[1, 0,
0],'MarkerFaceColor','r')
```

```
hold('on');

plot(xn(:)-x,(yn(:)-y),'LineWidth',3,'Color',[0, 0, 1])

hold('off');

legend('TMG  ', 'Newton  ')

title('Simple Trajectory')


%window_ar = 7.05/4.90; %x/y old ipad full
%window_ar = 3.00/4.75; %x/y old ipad half
window_ar = 6.70/10.5; %x/y new ipad half

pxmin = min(r(:,5)) - x;
pxmax = max(r(:,5)) - x;

pymin = min(r(:,6)) - y;
pymax = max(r(:,6)) - y;

py = (pymax - pymin)*1.05;

px = (pxmax - pxmin)*1.05;


if py > px / window_ar;
    ylim([pymin,pymin+py]);
    xlim([pxmin,pxmin+py*window_ar]);
else
    ylim([pymin,pymin+px/window_ar]);
    xlim([pxmin,pxmin+px]);
end

xlabel('x, meter')
ylabel('y, meter')
```

Example_2_Earth_satellites.m

```
%   Example_2_Earth_satellites.m
%
%   MatLab script for calculating satellite orbital conditions and clock rate
%   relative to the master clock at the U.S. Naval Observatory in Washington DC.
%
%   Copyright 2021 Morris G. Anderson
%
%
%   This script will be documented in the following paper to be ePublished on https://vixra.org
%
%
%   Curvature of Motion in a Gravitational Field
%                       by
%               Morris G. Anderson
%
%
%   It will also be uploaded to https://www.mathworks.com/matlabcentral/fileexchange under the same
name.
%
%
%   Information source
%   1)  Time Matter and Gravity (TMG) by Morris G. Anderson provides a derivation of the
%       equations used in this calculation.
%       https://vixra.org/abs/2102.0123
%       https://vixra.org/pdf/2102.0123v1.pdf
%
%   2)  Physical constants
%       https://physics.nist.gov/cuu/pdf/wall_2018.pdf
%
%   3)  Astronomical data
%       https://ssd.jpl.nasa.gov/horizons.cgi
%
%   4)  Satellite TLE data obtained from https://celestrak.com
%
%   5)  Fractional frequency shift data obtained and from
%       Relativity in the Global Positioning System by Neil Ashby
%       Published on 28 January 2003
%       https://link.springer.com/article/10.12942/lrr-2003-1
%
%
%   The path of motion is calculated with the TMG_path.m MatLab function.  The TMG_solver.m
%   script applies the TMG_path.m function to obtain solutions for:
%
%   path_option = 1; % Calculate the precession of an orbit.  The solution is calculated for one
%                       revolution if the max angle (atr) is input as 360.  The solution is calculated
for
%                       two revolutions if the max angle (atr) is input as 2 * 360 and so forth.
%                       All arc segments have the same rotation angle.
%                       The total rotation angle is part of the orbital precession solution.
%
%                atr = The total rotation angle for calculation, initially an input for the
"TMG_solver",
%                       is overwritten as an output of the "TMG_path" solution.  units = radians
%
%
%   path_option = 2; % Calculate the bending of light and the Shapiro time delay for a given path
length.
%                       Arc segment length is a function of radius from the governing body center.
%
%                st =  The intended maximum path length for the calculation
%
%
%   path_option = 3; % Calculate a simple trajectory or orbit with equal arc angle segments
%
%                atr = The total rotation angle for calculation, initially an input for the
"TMG_solver",
%                       is overwritten as an output of the "TMG_path" solution.  units = radians
%
%
%   path_option = 4; % Calculate a simple trajectory with equal arc length segments
%
%                 st = The intended maximum path length for calculation
%
%
%
%   Process for this example:
```

```
%
%     1)  Define the universal gravitational constant, g, and the standard speed of light, cs.
%         o  Define the mass of the Earth   (for calculating cr).
%         o  Define the radius of the Earth (for calculating cr).
%
%     2)  Define the reference speed of light, cr, equal to cs and the reference radius,
%         rr, equal to re.  This is used for calculating the speed of light as a function of
%         position from the Earth.
%
%     3)  Define the mass of the governing body, m, to be equal to the mass of the Earth.
%
%     For each satellite, call the associated input file to:
%
%     4)  Convert Two-line element set data to input values position and velocity
%
%     5)  Define path_option = 3 for calculating the satellite orbit.
%
%     6)  Define total rotation, number of points, and number of arc segments per point for this
calculation
%
%     7)  Call TMG_solver.m which in turn calls TMG_path.m to calculate the path.
%
%     8)  Post-process and plot the results.

clear
clc

%  g, universal gravitational constant. Units = m^3/(kg-s)

g  = 0.66743e-10;

%   The standard speed of light, cs.
%   see TMG section 2.4.1 The Speed of light at the standard location.

cs = 299792458.0;

disp(['Newtons universal gravitational constant, G = ' num2str(g,6) ' m^3/(kg-s)'])

disp(['The standard speed of light, cs = ' num2str(cs,10) ' m/s'])

%  The mass and surface radius of the Earth.  From NASA JPL HORIZONS Web-Interface

m_Earth = 398600.435436e9/g;
r_Earth = 6371010.0;

disp(['Earth mass = ' num2str(m_Earth,6) ' kg,  Earth volumetric mean radius = ' num2str(r_Earth, 9)
' m'])

%   For this calculation, define cr = cs

cr = cs;
rr = r_Earth;

disp(['Reference speed of light  = ' num2str(cr,12) ' m/s at ' num2str(rr,8) ' meters from the Earths
center of mass'])

%   Define m, This is the mass of the governing body in kilograms.  For this example,
%   it is the mass of the Earth.

m = m_Earth;

%   disp(['Mass of the governing body = ' num2str(m,11) ' kg' ])

%   Call script files to define the individual satellite initial x, y, z, coordinates with respect
%   to the center of mass of the governing body, m, in meters.




disp([''])
disp(['------'])
Example_2a_input_for_INTELSAT_39
[r_aboves, r_belows] = sort_for_plotting_orbit(np, r);
INTELSAT_39_r_above = r_aboves;
INTELSAT_39_r_below = r_belows;
```

```
disp([''])
disp(['------'])
Example_2b_input_for_MOLNIYA_1_36
[r_aboves, r_belows] = sort_for_plotting_orbit(np, r);
MOLNIYA_1_36_r_above = r_aboves;
MOLNIYA_1_36_r_below = r_belows;

disp([''])
disp(['------'])
Example_2c_input_for_NTS_2
[r_aboves, r_belows] = sort_for_plotting_orbit(np, r);
NTS_2_r_above = r_aboves;
NTS_2_r_below = r_belows;


disp([''])
disp(['------'])
Example_2d_input_for_ISS
[r_aboves, r_belows] = sort_for_plotting_orbit(np, r);
ISS_r_above = r_aboves;
ISS_r_below = r_belows;



% Top view
% plot surface of Earth

ae = linspace(0,2*pi,360);
xe = r_Earth * cos(ae);
ye = r_Earth * sin(ae);

plot(xe,ye,'Color','b','LineWidth',3)

hold('on')

plot(INTELSAT_39_r_above(:,5),INTELSAT_39_r_above(:,6),'LineWidth',2,'Color','r')
plot(INTELSAT_39_r_below(:,5),INTELSAT_39_r_below(:,6),'LineWidth',2,'Color','r', 'LineStyle', ':')


row =find(r_INTELSAT_39_path(:,4)==Perigee_INTELSAT_39);
plot(r_INTELSAT_39_path(row,5),r_INTELSAT_39_path(row,6),'LineStyle','none','Marker','o','MarkerSize'
,10,'MarkerEdgeColor','r','MarkerFaceColor','r')

row =find(r_INTELSAT_39_path(:,4)==Apogee_INTELSAT_39);
plot(r_INTELSAT_39_path(row,5),r_INTELSAT_39_path(row,6),'LineStyle','none','Marker','o','MarkerSize'
,10,'MarkerEdgeColor','r')




plot(MOLNIYA_1_36_r_above(:,5),MOLNIYA_1_36_r_above(:,6),'LineWidth',2,'Color','g')
plot(MOLNIYA_1_36_r_below(:,5),MOLNIYA_1_36_r_below(:,6),'LineWidth',2,'Color','g', 'LineStyle', ':')

row =find(r_MOLNIYA_1_36_path(:,4)==Perigee_MOLNIYA_1_36);
plot(r_MOLNIYA_1_36_path(row,5),r_MOLNIYA_1_36_path(row,6),'LineStyle','none','Marker','o','MarkerSiz
e',10,'MarkerEdgeColor','g','MarkerFaceColor','g')

row =find(r_MOLNIYA_1_36_path(:,4)==Apogee_MOLNIYA_1_36);
plot(r_MOLNIYA_1_36_path(row,5),r_MOLNIYA_1_36_path(row,6),'LineStyle','none','Marker','o','MarkerSiz
e',10,'MarkerEdgeColor','g')


plot(NTS_2_r_above(:,5),NTS_2_r_above(:,6),'LineWidth',2,'Color','c')
plot(NTS_2_r_below(:,5),NTS_2_r_below(:,6),'LineWidth',2,'Color','c', 'LineStyle', ':')

row =find(r_NTS_2_path(:,4)==Perigee_NTS_2);
plot(r_NTS_2_path(row,5),r_NTS_2_path(row,6),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerE
dgeColor','c','MarkerFaceColor','c')

row =find(r_NTS_2_path(:,4)==Apogee_NTS_2);
plot(r_NTS_2_path(row,5),r_NTS_2_path(row,6),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerE
dgeColor','c')


plot(ISS_r_above(:,5),ISS_r_above(:,6),'LineWidth',2,'Color',[1, 0.65, 0.35])
plot(ISS_r_below(:,5),ISS_r_below(:,6),'LineWidth',2,'Color',[1, 0.65, 0.35], 'LineStyle', ':')

row =find(r_ISS_path(:,4)==Perigee_ISS);
```

```
plot(r_ISS_path(row,5),r_ISS_path(row,6),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerEdgeC
olor',[1, 0.65, 0.35],'MarkerFaceColor',[1, 0.65, 0.35])

row =find(r_ISS_path(:,4)==Apogee_ISS);
plot(r_ISS_path(row,5),r_ISS_path(row,6),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerEdgeC
olor',[1, 0.65, 0.35])


hold('off')

title('Earth surface, blue; INTELSAT 39, red; MOLNIYA 1-36, green; NTS-2, cyan; ISS, orange')

xlabel('ECI x axis, meters')
ylabel('ECI y axis, meters')

plot_limit(1) = 1.05 * max(r_INTELSAT_39_path(:,4));
plot_limit(2) = 1.05 * max(r_MOLNIYA_1_36_path(:,4));

xlim([-1.44*max(plot_limit),1.44*max(plot_limit)])
ylim([-max(plot_limit),max(plot_limit)])

pause(20)




%side view


plot(xe,ye,'Color','b','LineWidth',3)

hold('on')

plot(INTELSAT_39_r_above(:,5),INTELSAT_39_r_above(:,7),'LineWidth',2,'Color','r')
plot(INTELSAT_39_r_below(:,5),INTELSAT_39_r_below(:,7),'LineWidth',2,'Color','r', 'LineStyle', ':')


row =find(r_INTELSAT_39_path(:,4)==Perigee_INTELSAT_39);
plot(r_INTELSAT_39_path(row,5),r_INTELSAT_39_path(row,7),'LineStyle','none','Marker','o','MarkerSize'
,10,'MarkerEdgeColor','r','MarkerFaceColor','r')

row =find(r_INTELSAT_39_path(:,4)==Apogee_INTELSAT_39);
plot(r_INTELSAT_39_path(row,5),r_INTELSAT_39_path(row,7),'LineStyle','none','Marker','o','MarkerSize'
,10,'MarkerEdgeColor','r')




plot(MOLNIYA_1_36_r_above(:,5),MOLNIYA_1_36_r_above(:,7),'LineWidth',2,'Color','g')
plot(MOLNIYA_1_36_r_below(:,5),MOLNIYA_1_36_r_below(:,7),'LineWidth',2,'Color','g', 'LineStyle', ':')

row =find(r_MOLNIYA_1_36_path(:,4)==Perigee_MOLNIYA_1_36);
plot(r_MOLNIYA_1_36_path(row,5),r_MOLNIYA_1_36_path(row,7),'LineStyle','none','Marker','o','MarkerSiz
e',10,'MarkerEdgeColor','g','MarkerFaceColor','g')

row =find(r_MOLNIYA_1_36_path(:,4)==Apogee_MOLNIYA_1_36);
plot(r_MOLNIYA_1_36_path(row,5),r_MOLNIYA_1_36_path(row,7),'LineStyle','none','Marker','o','MarkerSiz
e',10,'MarkerEdgeColor','g')


plot(NTS_2_r_above(:,5),NTS_2_r_above(:,7),'LineWidth',2,'Color','c')
plot(NTS_2_r_below(:,5),NTS_2_r_below(:,7),'LineWidth',2,'Color','c', 'LineStyle', ':')

row =find(r_NTS_2_path(:,4)==Perigee_NTS_2);
plot(r_NTS_2_path(row,5),r_NTS_2_path(row,7),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerE
dgeColor','c','MarkerFaceColor','c')

row =find(r_NTS_2_path(:,4)==Apogee_NTS_2);
plot(r_NTS_2_path(row,5),r_NTS_2_path(row,7),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerE
dgeColor','c')


plot(ISS_r_above(:,5),ISS_r_above(:,7),'LineWidth',2,'Color',[1, 0.65, 0.35])
plot(ISS_r_below(:,5),ISS_r_below(:,7),'LineWidth',2,'Color',[1, 0.65, 0.35], 'LineStyle', ':')

row =find(r_ISS_path(:,4)==Perigee_ISS);
```

```
plot(r_ISS_path(row,5),r_ISS_path(row,7),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerEdgeC
olor',[1, 0.65, 0.35],'MarkerFaceColor',[1, 0.65, 0.35])

row =find(r_ISS_path(:,4)==Apogee_ISS);
plot(r_ISS_path(row,5),r_ISS_path(row,7),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerEdgeC
olor',[1, 0.65, 0.35])


hold('off')

%legend('Earth surface','INTELSAT 39 above Ecliptic','below Ecliptic','Perigee','Apogee','MOLNIYA
1-36 above Ecliptic','below Ecliptic','Perigee','Apogee','NTS-2 above Ecliptic','below
Ecliptic','Perigee','Apogee')

title('Earth surface, blue; INTELSAT 39, red; MOLNIYA 1-36, green; NTS-2, cyan; ISS, orange')

xlabel('ECI x axis, meters')
ylabel('ECI z axis, meters')

plot_limit(1) = 1.05 * max(r_INTELSAT_39_path(:,4));
plot_limit(2) = 1.05 * max(r_MOLNIYA_1_36_path(:,4));

xlim([-1.44*max(plot_limit),1.44*max(plot_limit)])
ylim([-max(plot_limit),max(plot_limit)])

pause(20)

%  calculating satellite fractional frequency as a function of orbital radius relative
%  to the United States Naval Observatory master clock (see TMG eq. 5-29)

r_cir(1) = r_Earth;

b_cir(1) = sqrt( 1 / ( (cs*cs*r_cir(1) / m / g) -1 ) );

g1 = ( 2*m*g/cs/cs )  *  ( r_cir(1) - r_Earth ) / ( r_cir(1) * r_Earth );

c_cir(1) = cs* exp(g1);

ffs_cir(1) = 1 - ( sqrt( cs*(1 - b_usno * b_usno) ) / sqrt( c_cir(1)*(1 - b_cir(1) * b_cir(1) ) ) );


for nn = 2:100;

    r_cir(nn) = 1.03* r_cir(nn-1);

    b_cir(nn) = sqrt( 1 / ( (cs*cs*r_cir(nn) / m / g) -1 ) );

    g1 = ( 2*m*g/cs/cs )  *  ( r_cir(nn) - r_Earth ) / ( r_cir(nn) * r_Earth );

    c_cir(nn) = cs* exp(g1);

    ffs_cir(nn) = 1 - ( sqrt( cs*(1 - b_usno * b_usno) ) / sqrt( c_cir(nn)*(1 - b_cir(nn) * b_cir(nn)
) ) );

end

plot(r_cir(:),ffs_cir(:),'LineWidth',3,'Color','r')

hold('on')

plot(r_INTELSAT_39_semi,ffs_INTELSAT_39,'LineStyle','none','LineWidth',2,'Marker','o','MarkerSize',15
,'MarkerEdgeColor','r')

plot(r_MOLNIYA_1_36_semi,ffs_MOLNIYA_1_36,'LineStyle','none','LineWidth',2,'Marker','o','MarkerSize',
15,'MarkerEdgeColor','g')

plot(r_NTS_2_semi,ffs_NTS_2,'LineStyle','none','LineWidth',2,'Marker','^','MarkerSize',15,'MarkerEdge
Color','c', 'MarkerFaceColor','c')

plot(r_ISS_semi,ffs_ISS,'LineStyle','none','LineWidth',2,'Marker','*','MarkerSize',20,'MarkerEdgeColo
r',[1, 0.65, 0.35],'MarkerFaceColor',[1, 0.65, 0.35])

plot(r_NTS_2_semi,442.5e-12,'LineStyle','none','LineWidth',2,'Marker','x','MarkerSize',20,'MarkerEdge
Color','m','MarkerFaceColor','m')


hold('off')
```

```
title('Calculated ffs, red; INTELSAT 39, red; MOLNIYA 1-36, green; NTS-2, cyan; ISS, orange')

xlabel('Orbital semi-major axis, meters')
ylabel('fractional frequency shift')

legend('Calculated ffs', 'INTELSAT-39  ', 'MOLNIYA 1-36  ', 'NTS-2', 'ISS', 'Measured for NTS-2')

xlim([0,100000000]);
ylim([-300e-12, 700e-12]);
```

Example_2a_input_for_INTELSAT_39.m

```
%  Example_2a_input_for_INTELSAT_39.m

%  This script is called by Example_2_Earth_satellites.m

disp(['This example calculates the orbital motion of the INTELSAT_39 satellite'])

disp(['Initial conditions taken from https://celestrak.com NORAD Two-Line Element Sets 2020-08-07'])

%  NORAD Two-Line Element Set (TLE)
%  INTELSAT 39 (IS-39)
%  1 44476U 19049B   20220.46161271  .00000024  00000-0  00000-0 0  9998
%  2 44476   0.0088  71.4300 0001064 274.4541 198.6737  1.00268711  3714
%   SAT ID  Orb_Inc  Orb_LA  Ecc     Orb_LP            Orb_Period

Orb_Inc_TLE     = 0.0088;    % degrees, Orbital Inclination
Orb_LA_TLE      = 71.4300;   % degrees, Orbital Longitude of Ascending Node
Orb_Ecc_TLE     = 0.0001064; % Orbital eccentricity
Orb_LP_TLE      = 274.4541;  % degrees, Orbital Longitude of Perigee
Orb_Period_TLE  = 86400 / 1.00268711; %seconds, Orbital Period

%  Calculate the semi-major axis with Kepler's third law to be consistent with the NASA TLE data.

r_semi_TLE  = ( Orb_Period_TLE^2 * m* g / 4 / pi / pi )^(1/3);

Perigee_TLE = r_semi_TLE * (1-Orb_Ecc_TLE); % meters, applied as initial condition
Apogee_TLE  = r_semi_TLE * (1+Orb_Ecc_TLE); % meters, for comparison with solution

% Convert TLE angles to radians

 Orb_Inc_TLE = Orb_Inc_TLE * pi/180.0;
 Orb_LA_TLE  = Orb_LA_TLE  * pi/180.0;
 Orb_LP_TLE  = Orb_LP_TLE  * pi/180.0;

%   Define initial unrotated position and motion unit vectors for the orbital Perihelion
%   For this example it is done for 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) =  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 Perigee starting point.  Use Rodrigues' rotation formula.

theta = Orb_LA_TLE;

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_TLE - Orb_LA_TLE;

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_TLE;

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;


%  Define the initial x, y, z,  position coordinates with respect to the center of mass of the
%  governing body, m, in meters.  For this example, it is the closest orbital distance between
%  the satellite and center of the Earth.

x = rhat(4,1) * Perigee_TLE;
y = rhat(4,2) * Perigee_TLE;
z = rhat(4,3) * Perigee_TLE;

r(1,4) = Perigee_TLE;

disp(['Position   x = ' num2str(x,6) ' m,   y = ' num2str(y,6) ' m,  z = ' num2str(z,6) ' m' ])

%   Define vx, vy, vz, Initial velocity in m/s.  If beta_option = 1, these values are treated as
%   bx=vx/c, by=vy/c, bz=vz/c
%
%   For this example it is the maximum orbital velocity (this is at the Perigee).
%
%   Start with the circular orbital velocity corresponding to the semi-major axis.

%   Calculate beta with TMG equation 7-12

b_circular = sqrt( 1 / ( (cs*cs*r_semi_TLE / m / g) -1 ) );

%   Calculate c with TMG equation 4-10

c_circular = cr * exp( (2*m*g/cs/cs ) * ( ( r_semi_TLE-rr ) / (rr*r_semi_TLE)) );

c_Perigee  = cr * exp( (2*m*g/cs/cs ) * ( ( Perigee_TLE-rr ) / (rr*Perigee_TLE)) );

%   Calculate dc with TMG equation 4-14 to reduce numerical errors

g1 = ( 2*m*g/cs/cs )  *  ( ( Perigee_TLE - r_semi_TLE ) / (r_semi_TLE * Perigee_TLE));

g2 = g1 * g1;
g3 = g2 * g1;
g4 = g3 * g1;
g5 = g4 * g1;
g6 = g5 * g1;
g7 = g6 * g1;
g8 = g7 * g1;

dc = c_circular*(-g1 -g2/2. -g3/6. -g4/24. -g5/120.0 -g6/720. -g7/5040. -g8/40320);

%   Calculate the initial beta with TMG equations 4-18
```

```
b_Perigee = sqrt( (dc + b_circular*b_circular * c_Perigee) / c_circular );

beta_option = 1;
bx = b_Perigee * bhat(4,1);
by = b_Perigee * bhat(4,2);
bz = b_Perigee * bhat(4,3);

disp(['Velocity  vx = ' num2str(bx*c_Perigee,6) ' m/s,  vy = ' num2str(by*c_Perigee,6) ' m/s,  vz = '
num2str(bz*c_Perigee,6) ' m/s'])

disp(['Beta    vx/c = ' num2str(bx,6) ', vy/c = ' num2str(by,6) ', vz/c = ' num2str(bz,6) ])


%   Calculate the Orbit
path_option = 3; % Calculate a simple trajectory or orbit with equal arc angle segments

disp(' ')
disp('--- Calculating INTELSAT_39 orbit ---')

%   Define total rotation, number of points, and arc segments per point for this calculation

atr = 360 * 1.0;
np  = 180;
nac = 10;

disp(['Total rotation angle for calculation = ' num2str(atr,4) ' degrees' ])

disp(['Calculating solution with ' num2str(np) ' points, and ' num2str(nac) ' arc segments per
point'])


% Convert total rotation angle to radians for calculation

atr = atr * pi / 180;

TMG_solver

r_INTELSAT_39_path = r;

%   Post-process and display results

disp('--- INTELSAT_39 Orbital Solution Results -----')

error_t = t - Orb_Period_TLE;

disp(['Orbital Period = ' num2str( t/60,6) ' minutes, delta vs TLE = ' num2str(error_t,4) '
seconds' ])

disp(['Circumference  = ' num2str( s,12) ' meters' ])

Perigee_INTELSAT_39 = min(r(:,4));
 Apogee_INTELSAT_39 = max(r(:,4));


error_Perigee  =  Perigee_INTELSAT_39 -  Perigee_TLE;
error_Apogee   =   Apogee_INTELSAT_39 -   Apogee_TLE;

disp(['Perigee        = ' num2str(min(r(:,4)),12) ' meters, delta vs TLE = ' num2str(error_Perigee,4)
' meters' ])

disp(['Apogee         = ' num2str(max(r(:,4)),12) ' meters, delta vs TLE = ' num2str(error_Apogee,4)
' meters' ])

eccentricity_INTELSAT_39 = ( max(r(:,4)) - min(r(:,4)) ) / ( max(r(:,4)) + min(r(:,4)) );

error_eccentricity = eccentricity_INTELSAT_39 - Orb_Ecc_TLE;

disp(['Eccentricity   = ' num2str( eccentricity_INTELSAT_39,8) ', delta vs TLE = '
num2str(error_eccentricity,4)])

%   calculate the clock rate shift relative to the United States Naval Observatory master clock

r_usno = r_Earth * cos(38.9217 * pi / 180);
v_usno = 2*r_usno*pi/86400;
b_usno = v_usno / cs;

%  f2x is an output of TMG_path.m.  It is used to calculate the time-averaged frequency
%  shift (see TMG eq. 5-29)
```

```
ffs_INTELSAT_39 = 1 - (sqrt( cs*(1 - b_usno * b_usno) ) / f2x );

disp(['df/f           = ' num2str(ffs_INTELSAT_39,4) ' fractional frequency shift relative to the
USNO master clock'])

% for plotting results
r_INTELSAT_39_semi = ( Perigee_INTELSAT_39 + Apogee_INTELSAT_39 ) / 2;
```

Example_2b_input_for_MOLNIYA_1_36.m

```
%  Example_2b_input_for_MOLNIYA_1_36.m

%  This script is called by Example_2_Earth_satellites.m

disp(['This example calculates the orbital motion of the MOLNIYA_1_36 satellite'])

disp(['Initial conditions taken from https://celestrak.com NORAD Two-Line Element Sets 2020-08-08'])

%  NORAD Two-Line Element Set (TLE)
%  MOLNIYA 1-36
%  1 09880U 77021A   20221.43151221 -.00000781  00000-0  12291-3 0  9993
%  2 09880  62.5563  12.2976 7475291 285.3676   9.4914  2.00344631215809
%   SAT ID  Orb_Inc  Orb_LA  Ecc     Orb_LP            Orb_Period

Orb_Inc_TLE     = 62.5563;    % degrees, Orbital Inclination
Orb_LA_TLE      = 12.2976;   % degrees, Orbital Longitude of Ascending Node
Orb_Ecc_TLE     = 0.7475291; % Orbital eccentricity
Orb_LP_TLE      = 285.3676;  % degrees, Orbital Longitude of Perigee
Orb_Period_TLE  = 86400.0 / 2.00344631; %seconds, Orbital Period

%  Calculate the semi-major axis with Kepler's third law to be consistent with the NASA TLE data.

r_semi_TLE  = ( Orb_Period_TLE^2 * m* g / 4 / pi / pi )^(1/3);

Perigee_TLE = r_semi_TLE * (1-Orb_Ecc_TLE); % meters, applied as initial condition
Apogee_TLE  = r_semi_TLE * (1+Orb_Ecc_TLE); % meters, for comparison with solution

% Convert TLE angles to radians

 Orb_Inc_TLE = Orb_Inc_TLE * pi/180.0;
 Orb_LA_TLE  = Orb_LA_TLE  * pi/180.0;
 Orb_LP_TLE  = Orb_LP_TLE  * pi/180.0;

%   Define initial unrotated position and motion unit vectors for the orbital Perihelion
%   For this example it is done for 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) =  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 Perigee starting point.  Use Rodrigues' rotation formula.

theta = Orb_LA_TLE;

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_TLE - Orb_LA_TLE;

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_TLE;

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;


%  Define the initial x, y, z,  position coordinates with respect to the center of mass of the
%  governing body, m, in meters.  For this example, it is the closest orbital distance between
%  the satellite and center of the Earth.

x = rhat(4,1) * Perigee_TLE;
y = rhat(4,2) * Perigee_TLE;
z = rhat(4,3) * Perigee_TLE;

r(1,4) = Perigee_TLE;

disp(['Position   x = ' num2str(x,6) ' m,   y = ' num2str(y,6) ' m,  z = ' num2str(z,6) ' m' ])

%   Define vx, vy, vz, Initial velocity in m/s.  If beta_option = 1, these values are treated as
%   bx=vx/c, by=vy/c, bz=vz/c
%
%   For this example it is the maximum orbital velocity (this is at the Perigee).
%
%   Start with the circular orbital velocity corresponding to the semi-major axis.

%   Calculate beta with TMG equation 7-12

b_circular = sqrt( 1 / ( (cs*cs*r_semi_TLE / m / g) -1 ) );

%   Calculate c with TMG equation 4-10

c_circular = cr * exp( (2*m*g/cs/cs ) * ( ( r_semi_TLE-rr ) / (rr*r_semi_TLE)) );

c_Perigee  = cr * exp( (2*m*g/cs/cs ) * ( ( Perigee_TLE-rr ) / (rr*Perigee_TLE)) );

%   Calculate dc with TMG equation 4-14 to reduce numerical errors

g1 = ( 2*m*g/cs/cs )  *  ( ( Perigee_TLE - r_semi_TLE ) / (r_semi_TLE * Perigee_TLE));

g2 = g1 * g1;
g3 = g2 * g1;
g4 = g3 * g1;
g5 = g4 * g1;
g6 = g5 * g1;
g7 = g6 * g1;
g8 = g7 * g1;

dc = c_circular*(-g1 -g2/2. -g3/6. -g4/24. -g5/120.0 -g6/720. -g7/5040. -g8/40320);

%   Calculate the initial beta with TMG equations 4-18
```

```
b_Perigee = sqrt( (dc + b_circular*b_circular * c_Perigee) / c_circular );

beta_option = 1;
bx = b_Perigee * bhat(4,1);
by = b_Perigee * bhat(4,2);
bz = b_Perigee * bhat(4,3);

disp(['Velocity  vx = ' num2str(bx*c_Perigee,6) ' m/s,  vy = ' num2str(by*c_Perigee,6) ' m/s,  vz = '
num2str(bz*c_Perigee,6) ' m/s'])

disp(['Beta    vx/c = ' num2str(bx,6) ', vy/c = ' num2str(by,6) ', vz/c = ' num2str(bz,6) ])


%   Calculate the Orbit
path_option = 3; % Calculate a simple trajectory or orbit with equal arc angle segments

disp(' ')
disp('--- Calculating MOLNIYA_1_36 orbit ---')

%   Define total rotation, number of points, and arc segments per point for this calculation

atr = 360 * 1.0;
np  = 180;
nac = 10;

disp(['Total rotation angle for calculation = ' num2str(atr,4) ' degrees' ])

disp(['Calculating solution with ' num2str(np) ' points, and ' num2str(nac) ' arc segments per
point'])


% Convert total rotation angle to radians for calculation

atr = atr * pi / 180;

TMG_solver

r_MOLNIYA_1_36_path = r;

%   Post-process and display results

disp('--- MOLNIYA_1_36 Orbital Solution Results -----')

error_t = t - Orb_Period_TLE;

disp(['Orbital Period = ' num2str( t/60,6) ' minutes, delta vs TLE = ' num2str(error_t,4) '
seconds' ])

disp(['Circumference  = ' num2str( s,12) ' meters' ])

Perigee_MOLNIYA_1_36 = min(r(:,4));
 Apogee_MOLNIYA_1_36 = max(r(:,4));


error_Perigee  =  Perigee_MOLNIYA_1_36 -  Perigee_TLE;
error_Apogee   =   Apogee_MOLNIYA_1_36 -   Apogee_TLE;

disp(['Perigee        = ' num2str(min(r(:,4)),12) ' meters, delta vs TLE = ' num2str(error_Perigee,4)
' meters' ])

disp(['Apogee         = ' num2str(max(r(:,4)),12) ' meters, delta vs TLE = ' num2str(error_Apogee,4)
' meters' ])

eccentricity_MOLNIYA_1_36 = ( max(r(:,4)) - min(r(:,4)) ) / ( max(r(:,4)) + min(r(:,4)) );

error_eccentricity = eccentricity_MOLNIYA_1_36 - Orb_Ecc_TLE;

disp(['Eccentricity   = ' num2str( eccentricity_MOLNIYA_1_36,8) ', delta vs TLE = '
num2str(error_eccentricity,4)])

%   calculate the clock rate shift relative to the United States Naval Observatory master clock

r_usno = r_Earth * cos(38.9217 * pi / 180);
v_usno = 2*r_usno*pi/86400;
b_usno = v_usno / cs;

%  f2x is an output of TMG_path.m.  It is used to calculate the time-averaged frequency
%  shift (see TMG eq. 5-29)
```

```
ffs_MOLNIYA_1_36 = 1 - (sqrt( cs*(1 - b_usno * b_usno) ) / f2x );

disp(['df/f           = ' num2str(ffs_MOLNIYA_1_36,4) ' fractional frequency shift relative to the
USNO master clock'])

% for plotting results
r_MOLNIYA_1_36_semi = ( Perigee_MOLNIYA_1_36 + Apogee_MOLNIYA_1_36 ) / 2;

%dr = r(1,4) - r(np+1,4)
```

Example_2c_input_for_NTS_2.m

```
%  Example_2c_input_for_NTS_2.m

%  This script is called by Example_2_Earth_satellites.m

disp(['This example calculates the orbital motion of the NTS_2 satellite'])

disp(['Initial conditions taken from https://celestrak.com NORAD Two-Line Element Sets 2020-08-10'])

%  NORAD Two-Line Element Set (TLE)
%  NTS 2
%  1 10091U 77053A   20223.48586522 -.00000072  00000-0  00000-0 0  9992
%  2 10091  64.4363 348.0239 0064823 230.3096 128.9200  2.00449092315847
%   SAT ID  Orb_Inc  Orb_LA  Ecc     Orb_LP            Orb_Period

Orb_Inc_TLE     = 64.4363;    % degrees, Orbital Inclination
Orb_LA_TLE      = 348.0239;   % degrees, Orbital Longitude of Ascending Node
Orb_Ecc_TLE     = 0.0064823; % Orbital eccentricity
Orb_LP_TLE      = 285.3676;  % degrees, Orbital Longitude of Perigee
Orb_Period_TLE  = 86400.0 / 2.00449092; %seconds, Orbital Period

%  Calculate the semi-major axis with Kepler's third law to be consistent with the NASA TLE data.

r_semi_TLE  = ( Orb_Period_TLE^2 * m* g / 4 / pi / pi )^(1/3);

Perigee_TLE = r_semi_TLE * (1-Orb_Ecc_TLE); % meters, applied as initial condition
Apogee_TLE  = r_semi_TLE * (1+Orb_Ecc_TLE); % meters, for comparison with solution

% Convert TLE angles to radians

 Orb_Inc_TLE = Orb_Inc_TLE * pi/180.0;
 Orb_LA_TLE  = Orb_LA_TLE  * pi/180.0;
 Orb_LP_TLE  = Orb_LP_TLE  * pi/180.0;

%   Define initial unrotated position and motion unit vectors for the orbital Perihelion
%   For this example it is done for 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) =  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 Perigee starting point.  Use Rodrigues' rotation formula.

theta = Orb_LA_TLE;

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_TLE - Orb_LA_TLE;

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_TLE;

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;


%  Define the initial x, y, z,  position coordinates with respect to the center of mass of the
%  governing body, m, in meters.  For this example, it is the closest orbital distance between
%  the satellite and center of the Earth.

x = rhat(4,1) * Perigee_TLE;
y = rhat(4,2) * Perigee_TLE;
z = rhat(4,3) * Perigee_TLE;

r(1,4) = Perigee_TLE;

disp(['Position   x = ' num2str(x,6) ' m,   y = ' num2str(y,6) ' m,  z = ' num2str(z,6) ' m' ])

%   Define vx, vy, vz, Initial velocity in m/s.  If beta_option = 1, these values are treated as
%   bx=vx/c, by=vy/c, bz=vz/c
%
%   For this example it is the maximum orbital velocity (this is at the Perigee).
%
%   Start with the circular orbital velocity corresponding to the semi-major axis.

%   Calculate beta with TMG equation 7-12

b_circular = sqrt( 1 / ( (cs*cs*r_semi_TLE / m / g) -1 ) );

%   Calculate c with TMG equation 4-10

c_circular = cr * exp( (2*m*g/cs/cs ) * ( ( r_semi_TLE-rr ) / (rr*r_semi_TLE)) );

c_Perigee  = cr * exp( (2*m*g/cs/cs ) * ( ( Perigee_TLE-rr ) / (rr*Perigee_TLE)) );

%   Calculate dc with TMG equation 4-14 to reduce numerical errors

g1 = ( 2*m*g/cs/cs )  *  ( ( Perigee_TLE - r_semi_TLE ) / (r_semi_TLE * Perigee_TLE));

g2 = g1 * g1;
g3 = g2 * g1;
g4 = g3 * g1;
g5 = g4 * g1;
g6 = g5 * g1;
g7 = g6 * g1;
g8 = g7 * g1;

dc = c_circular*(-g1 -g2/2. -g3/6. -g4/24. -g5/120.0 -g6/720. -g7/5040. -g8/40320);

%   Calculate the initial beta with TMG equations 4-18
```

```
b_Perigee = sqrt( (dc + b_circular*b_circular * c_Perigee) / c_circular );

beta_option = 1;
bx = b_Perigee * bhat(4,1);
by = b_Perigee * bhat(4,2);
bz = b_Perigee * bhat(4,3);

disp(['Velocity  vx = ' num2str(bx*c_Perigee,6) ' m/s,  vy = ' num2str(by*c_Perigee,6) ' m/s,  vz = '
num2str(bz*c_Perigee,6) ' m/s'])

disp(['Beta    vx/c = ' num2str(bx,6) ', vy/c = ' num2str(by,6) ', vz/c = ' num2str(bz,6) ])


%   Calculate the Orbit
path_option = 3; % Calculate a simple trajectory or orbit with equal arc angle segments

disp(' ')
disp('--- Calculating NTS_2 orbit ---')

%   Define total rotation, number of points, and arc segments per point for this calculation

atr = 360 * 1.0;
np  = 180;
nac = 10;

disp(['Total rotation angle for calculation = ' num2str(atr,4) ' degrees' ])

disp(['Calculating solution with ' num2str(np) ' points, and ' num2str(nac) ' arc segments per
point'])


% Convert total rotation angle to radians for calculation

atr = atr * pi / 180;

TMG_solver

r_NTS_2_path = r;

%   Post-process and display results

disp('--- NTS_2 Orbital Solution Results -----')

error_t = t - Orb_Period_TLE;

disp(['Orbital Period = ' num2str( t/60,6) ' minutes, delta vs TLE = ' num2str(error_t,4) '
seconds' ])

disp(['Circumference  = ' num2str( s,12) ' meters' ])

Perigee_NTS_2 = min(r(:,4));
 Apogee_NTS_2 = max(r(:,4));


error_Perigee  =  Perigee_NTS_2 -  Perigee_TLE;
error_Apogee   =   Apogee_NTS_2 -   Apogee_TLE;

disp(['Perigee        = ' num2str(min(r(:,4)),12) ' meters, delta vs TLE = ' num2str(error_Perigee,4)
' meters' ])

disp(['Apogee         = ' num2str(max(r(:,4)),12) ' meters, delta vs TLE = ' num2str(error_Apogee,4)
' meters' ])

eccentricity_NTS_2 = ( max(r(:,4)) - min(r(:,4)) ) / ( max(r(:,4)) + min(r(:,4)) );

error_eccentricity = eccentricity_NTS_2 - Orb_Ecc_TLE;

disp(['Eccentricity   = ' num2str( eccentricity_NTS_2,8) ', delta vs TLE = '
num2str(error_eccentricity,4)])

%   calculate the clock rate shift relative to the United States Naval Observatory master clock

r_usno = r_Earth * cos(38.9217 * pi / 180);
v_usno = 2*r_usno*pi/86400;
b_usno = v_usno / cs;

%  f2x is an output of TMG_path.m.  It is used to calculate the time-averaged frequency
%  shift (see TMG eq. 5-29)
```

```
ffs_NTS_2 = 1 - (sqrt( cs*(1 - b_usno * b_usno) ) / f2x );

disp(['df/f           = ' num2str(ffs_NTS_2,4) ' fractional frequency shift relative to the USNO
master clock'])

% for plotting results
r_NTS_2_semi = ( Perigee_NTS_2 + Apogee_NTS_2 ) / 2;
```

Example_2d_input_for_ISS.m

```
%  Example_2d_input_for_ISS.m

%  This script is called by Example_2_Earth_satellites.m

disp(['This example calculates the orbital motion of the ISS satellite'])

disp(['Initial conditions taken from https://celestrak.com NORAD Two-Line Element Sets 2020-08-10'])

%  NORAD Two-Line Element Set (TLE)
%  ISS (ZARYA)
%  1 25544U 98067A   20225.87978009 -.00000449  00000-0  00000+0 0  9996
%  2 25544  51.6461  67.8873 0001569  33.9347 160.9997 15.49157177240806
%   SAT ID  Orb_Inc  Orb_LA  Ecc     Orb_LP            Orb_Period

Orb_Inc_TLE     = 51.6461;    % degrees, Orbital Inclination
Orb_LA_TLE      = 67.8873;   % degrees, Orbital Longitude of Ascending Node
Orb_Ecc_TLE     = 0.0001569; % Orbital eccentricity
Orb_LP_TLE      = 33.9347;  % degrees, Orbital Longitude of Perigee
Orb_Period_TLE  = 86400.0 / 15.49157177; %seconds, Orbital Period

%  Calculate the semi-major axis with Kepler's third law to be consistent with the NASA TLE data.

r_semi_TLE  = ( Orb_Period_TLE^2 * m* g / 4 / pi / pi )^(1/3);

Perigee_TLE = r_semi_TLE * (1-Orb_Ecc_TLE); % meters, applied as initial condition
Apogee_TLE  = r_semi_TLE * (1+Orb_Ecc_TLE); % meters, for comparison with solution

% Convert TLE angles to radians

 Orb_Inc_TLE = Orb_Inc_TLE * pi/180.0;
 Orb_LA_TLE  = Orb_LA_TLE  * pi/180.0;
 Orb_LP_TLE  = Orb_LP_TLE  * pi/180.0;

%   Define initial unrotated position and motion unit vectors for the orbital Perihelion
%   For this example it is done for 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) =  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 Perigee starting point.  Use Rodrigues' rotation formula.

theta = Orb_LA_TLE;

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_TLE - Orb_LA_TLE;

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_TLE;

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;


%  Define the initial x, y, z,  position coordinates with respect to the center of mass of the
%  governing body, m, in meters.  For this example, it is the closest orbital distance between
%  the satellite and center of the Earth.

x = rhat(4,1) * Perigee_TLE;
y = rhat(4,2) * Perigee_TLE;
z = rhat(4,3) * Perigee_TLE;

r(1,4) = Perigee_TLE;

disp(['Position   x = ' num2str(x,6) ' m,   y = ' num2str(y,6) ' m,  z = ' num2str(z,6) ' m' ])

%   Define vx, vy, vz, Initial velocity in m/s.  If beta_option = 1, these values are treated as
%   bx=vx/c, by=vy/c, bz=vz/c
%
%   For this example it is the maximum orbital velocity (this is at the Perigee).
%
%   Start with the circular orbital velocity corresponding to the semi-major axis.

%   Calculate beta with TMG equation 7-12

b_circular = sqrt( 1 / ( (cs*cs*r_semi_TLE / m / g) -1 ) );

%   Calculate c with TMG equation 4-10

c_circular = cr * exp( (2*m*g/cs/cs ) * ( ( r_semi_TLE-rr ) / (rr*r_semi_TLE)) );

c_Perigee  = cr * exp( (2*m*g/cs/cs ) * ( ( Perigee_TLE-rr ) / (rr*Perigee_TLE)) );

%   Calculate dc with TMG equation 4-14 to reduce numerical errors

g1 = ( 2*m*g/cs/cs )  *  ( ( Perigee_TLE - r_semi_TLE ) / (r_semi_TLE * Perigee_TLE));

g2 = g1 * g1;
g3 = g2 * g1;
g4 = g3 * g1;
g5 = g4 * g1;
g6 = g5 * g1;
g7 = g6 * g1;
g8 = g7 * g1;

dc = c_circular*(-g1 -g2/2. -g3/6. -g4/24. -g5/120.0 -g6/720. -g7/5040. -g8/40320);

%   Calculate the initial beta with TMG equations 4-18

b_Perigee = sqrt( (dc + b_circular*b_circular * c_Perigee) / c_circular );
```

```
beta_option = 1;
bx = b_Perigee * bhat(4,1);
by = b_Perigee * bhat(4,2);
bz = b_Perigee * bhat(4,3);

disp(['Velocity  vx = ' num2str(bx*c_Perigee,6) ' m/s,  vy = ' num2str(by*c_Perigee,6) ' m/s,  vz = '
num2str(bz*c_Perigee,6) ' m/s'])

disp(['Beta    vx/c = ' num2str(bx,6) ', vy/c = ' num2str(by,6) ', vz/c = ' num2str(bz,6) ])


%   Calculate the Orbit
path_option = 3; % Calculate a simple trajectory or orbit with equal arc angle segments

disp(' ')
disp('--- Calculating ISS orbit ---')

%   Define total rotation, number of points, and arc segments per point for this calculation

atr = 360 * 1.0;
np  = 180;
nac = 10;

disp(['Total rotation angle for calculation = ' num2str(atr,4) ' degrees' ])

disp(['Calculating solution with ' num2str(np) ' points, and ' num2str(nac) ' arc segments per
point'])


% Convert total rotation angle to radians for calculation

atr = atr * pi / 180;

TMG_solver

r_ISS_path = r;

%   Post-process and display results

disp('--- ISS Orbital Solution Results -----')

error_t = t - Orb_Period_TLE;

disp(['Orbital Period = ' num2str( t/60,6) ' minutes, delta vs TLE = ' num2str(error_t,4) '
seconds' ])

disp(['Circumference  = ' num2str( s,12) ' meters' ])

Perigee_ISS = min(r(:,4));
 Apogee_ISS = max(r(:,4));


error_Perigee  =  Perigee_ISS -  Perigee_TLE;
error_Apogee   =   Apogee_ISS -   Apogee_TLE;

disp(['Perigee        = ' num2str(min(r(:,4)),12) ' meters, delta vs TLE = ' num2str(error_Perigee,4)
' meters' ])

disp(['Apogee         = ' num2str(max(r(:,4)),12) ' meters, delta vs TLE = ' num2str(error_Apogee,4)
' meters' ])

eccentricity_ISS = ( max(r(:,4)) - min(r(:,4)) ) / ( max(r(:,4)) + min(r(:,4)) );

error_eccentricity = eccentricity_ISS - Orb_Ecc_TLE;

disp(['Eccentricity   = ' num2str( eccentricity_ISS,8) ', delta vs TLE = '
num2str(error_eccentricity,4)])

%   calculate the clock rate shift relative to the United States Naval Observatory master clock

r_usno = r_Earth * cos(38.9217 * pi / 180);
v_usno = 2*r_usno*pi/86400;
b_usno = v_usno / cs;

%  f2x is an output of TMG_path.m.  It is used to calculate the time-averaged frequency
%  shift (see TMG eq. 5-29)
```

```
ffs_ISS = 1 - (sqrt( cs*(1 - b_usno * b_usno) ) / f2x );

disp(['df/f           = ' num2str(ffs_ISS,4) ' fractional frequency shift relative to the USNO master
clock'])

% for plotting results
r_ISS_semi = ( Perigee_ISS + Apogee_ISS ) / 2;
```

Example_3_Solar_system.m

```
%   Example_3_Solar_system.m
%
%   MatLab script for calculating the orbits of planets in the solar system.
%
%   Copyright 2021 Morris G. Anderson
%
%
%   This script will be documented in the following paper to be ePublished on https://vixra.org
%
%
%   Curvature of Motion in a Gravitational Field
%                       by
%               Morris G. Anderson
%
%
%   It will also be uploaded to https://www.mathworks.com/matlabcentral/fileexchange under the same
name.
%
%
%   Information source
%   1)  Time Matter and Gravity (TMG) by Morris G. Anderson provides a derivation of the
%       equations used in this calculation.
%       https://vixra.org/abs/2102.0123
%       https://vixra.org/pdf/2102.0123v1.pdf
%
%   2)  Physical constants
%       https://physics.nist.gov/cuu/pdf/wall_2018.pdf
%
%   3)  Astronomical data
%       https://ssd.jpl.nasa.gov/horizons.cgi
%       https://nssdc.gsfc.nasa.gov/planetary/factsheet
%
%
%
%   The path of motion is calculated with the TMG_path.m MatLab function.  The TMG_solver.m
%   script applies the TMG_path.m function to obtain solutions for:
%
%   path_option = 1; % Calculate the precession of an orbit.  The solution is calculated for one
%                       revolution if the max angle (atr) is input as 360.  The solution is calculated
for
%                       two revolutions if the max angle (atr) is input as 2 * 360 and so forth.
%                       All arc segments have the same rotation angle.
%                       The total rotation angle is part of the orbital precession solution.
%
%                atr = The total rotation angle for calculation, initially an input for the
"TMG_solver",
%                       is overwritten as an output of the "TMG_path" solution.  units = radians
%
%
%   path_option = 2; % Calculate the bending of light and the Shapiro time delay for a given path
length.
%                       Arc segment length is a function of radius from the governing body center.
%
%                st =  The intended maximum path length for the calculation
%
%
%   path_option = 3; % Calculate a simple trajectory or orbit with equal arc angle segments
%
%                atr = The total rotation angle for calculation, initially an input for the
"TMG_solver",
%                       is overwritten as an output of the "TMG_path" solution.  units = radians
%
%
%   path_option = 4; % Calculate a simple trajectory with equal arc length segments
%
%                 st = The intended maximum path length for calculation
%
%
%
%   Process for this example:
%
%     1)  Define the universal gravitational constant, g, and the standard speed of light, cs.
%         o  Define the mass of the Earth   (for calculating cr).
%         o  Define the radius of the Earth (for calculating cr).
%
%     2)  Calculate cr which is the speed of light away from the Earth at rr (the Earth's
```

```
%         average distance from the Sun).
%
%     3)  Define the mass of the governing body, m, to be equal to the mass of the Sun.
%
%     For each planet, call the associated input file to:
%
%     4)  Convert NASA orbital elements to input values for position and velocity
%
%     5)  Define path_option = 3 for calculating the planetary orbit.
%
%     6)  Define total rotation, number of points, and number of arc segments per point for this
calculation
%
%     7)  Call TMG_solver.m which in turn calls TMG_path.m to calculate the path.
%
%     8)  Define path_option = 1 for calculating the planet orbital precession
%
%     9)  Define total rotation, number of points, and number of arc segments per point for this
calculation
%
%     10)  Call TMG_solver.m which in turn calls TMG_path.m to calculate the path.
%
%     11)  Post-process and plot the results.
clear
clc

disp(['This example calculates the orbital motion and precession of the Solar system planets.'])

%  g, universal gravitational constant. Units = m^3/(kg-s),  See Reference 2

g  = 0.66743e-10;

%   The standard speed of light, cs.
%   From Reference 1 (see TMG section 2.4.1  The Speed of light at the standard location)

cs = 299792458.0;

disp(['Newtons universal gravitational constant, G = ' num2str(g,6) ' m^3/(kg-s)'])

disp(['The standard speed of light, cs = ' num2str(cs,10) ' m/s'])

%  The mass and surface radius of the Earth.  From NASA JPL HORIZONS Web-Interface

m_Earth = 398600.435436e9/g;
r_Earth = 6371010.0;

disp(['Earth mass = ' num2str(m_Earth,6) ' kg,  Earth Volumetric mean radius = ' num2str(r_Earth,8) '
m'])

%   For this calculation, cr is the speed of light away from the Earth at rr (the Earth's
%   average distance from the Sun).  Calculate cr with TMG eq 4-11 as:

cr = cs * exp(2.0 * m_Earth * g / cs / cs / r_Earth);

%   Define rr as the Earth's Semi-major axis from Reference 4.

rr = 149597898000.0;

disp(['Reference speed of light  = ' num2str(cr,12) ' m/s at ' num2str(rr,8) ' meters from the Sun
(the Earths Semi-major axis)'])

%     Define m, This is the mass of the governing body in kilograms
%     For this example it is the mass of the Sun, see Reference 3

m_Sun =  1.32712e20/g; % from https://nssdc.gsfc.nasa.gov/planetary/factsheet/sunfact.html

m = m_Sun;

disp(['Sun mass = ' num2str((m),15) ' kg' ])

%   Call script files to calculate orbits


disp([''])
disp(['------'])
Example_3a_input_for_Mercury
r  =  r_Mercury_path;
np = np_Mercury_path;
```

```
[r_aboves, r_belows] = sort_for_plotting_orbit(np, r);
Mercury_r_above = r_aboves;
Mercury_r_below = r_belows;

disp([''])
disp(['------'])
Example_3b_input_for_Venus
r  =  r_Venus_path;
np = np_Venus_path;
[r_aboves, r_belows] = sort_for_plotting_orbit(np, r);
Venus_r_above = r_aboves;
Venus_r_below = r_belows;

disp([''])
disp(['------'])
Example_3c_input_for_Earth
r  =  r_Earth_path;
np = np_Earth_path;
[r_aboves, r_belows] = sort_for_plotting_orbit(np, r);
Earth_r_above = r_aboves;
Earth_r_below = r_belows;

disp([''])
disp(['------'])
Example_3d_input_for_Mars
r  =  r_Mars_path;
np = np_Mars_path;
[r_aboves, r_belows] = sort_for_plotting_orbit(np, r);
Mars_r_above = r_aboves;
Mars_r_below = r_belows;

disp([''])
disp(['------'])
Example_3e_input_for_Jupiter
r  =  r_Jupiter_path;
np = np_Jupiter_path;
[r_aboves, r_belows] = sort_for_plotting_orbit(np, r);
Jupiter_r_above = r_aboves;
Jupiter_r_below = r_belows;

disp([''])
disp(['------'])
Example_3f_input_for_Saturn
r  =  r_Saturn_path;
np = np_Saturn_path;
[r_aboves, r_belows] = sort_for_plotting_orbit(np, r);
Saturn_r_above = r_aboves;
Saturn_r_below = r_belows;

disp([''])
disp(['------'])
Example_3g_input_for_Uranus
r  =  r_Uranus_path;
np = np_Uranus_path;
[r_aboves, r_belows] = sort_for_plotting_orbit(np, r);
Uranus_r_above = r_aboves;
Uranus_r_below = r_belows;

disp([''])
disp(['------'])
Example_3h_input_for_Neptune
r  =  r_Neptune_path;
np = np_Neptune_path;
[r_aboves, r_belows] = sort_for_plotting_orbit(np, r);
Neptune_r_above = r_aboves;
Neptune_r_below = r_belows;

disp([''])
disp(['------'])
Example_3i_input_for_Pluto
r  =  r_Pluto_path;
np = np_Pluto_path;
[r_aboves, r_belows] = sort_for_plotting_orbit(np, r);
Pluto_r_above = r_aboves;
Pluto_r_below = r_belows;


% Top view
```

```
% plot surface of Sun
r_Sun = 695700000.0;

as= linspace(0,2*pi,360);
xs = r_Sun * cos(as);
ys = r_Sun * sin(as);

plot(xs,ys,'Color',[1, 0.65, 0.35],'LineWidth',3)

hold('on')

plot(Mercury_r_above(:,5),Mercury_r_above(:,6),'LineWidth',2,'Color','g')
plot(Mercury_r_below(:,5),Mercury_r_below(:,6),'LineWidth',2,'Color','g', 'LineStyle', ':')

row =find(r_Mercury_path(:,4)==Perihelion_Mercury);
plot(r_Mercury_path(row,5),r_Mercury_path(row,6),'LineStyle','none','Marker','o','MarkerSize',10,'Mar
kerEdgeColor','g','MarkerFaceColor','g')

row =find(r_Mercury_path(:,4)==Aphelion_Mercury);
plot(r_Mercury_path(row,5),r_Mercury_path(row,6),'LineStyle','none','Marker','o','MarkerSize',10,'Mar
kerEdgeColor','g')



plot(Venus_r_above(:,5),Venus_r_above(:,6),'LineWidth',2,'Color',[1, 0.65, 0.35])
plot(Venus_r_below(:,5),Venus_r_below(:,6),'LineWidth',2,'Color',[1, 0.65, 0.35], 'LineStyle', ':')

row =find(r_Venus_path(:,4)==Perihelion_Venus);
plot(r_Venus_path(row,5),r_Venus_path(row,6),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerE
dgeColor',[1, 0.65, 0.35],'MarkerFaceColor',[1, 0.65, 0.35])

row =find(r_Venus_path(:,4)==Aphelion_Venus);
plot(r_Venus_path(row,5),r_Venus_path(row,6),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerE
dgeColor',[1, 0.65, 0.35])



plot(Earth_r_above(:,5),Earth_r_above(:,6),'LineWidth',2,'Color','b')
plot(Earth_r_below(:,5),Earth_r_below(:,6),'LineWidth',2,'Color','b', 'LineStyle', ':')

row =find(r_Earth_path(:,4)==Perihelion_Earth);
plot(r_Earth_path(row,5),r_Earth_path(row,6),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerE
dgeColor','b','MarkerFaceColor','b')

row =find(r_Earth_path(:,4)==Aphelion_Earth);
plot(r_Earth_path(row,5),r_Earth_path(row,6),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerE
dgeColor','b')



plot(Mars_r_above(:,5),Mars_r_above(:,6),'LineWidth',2,'Color','r')
plot(Mars_r_below(:,5),Mars_r_below(:,6),'LineWidth',2,'Color','r', 'LineStyle', ':')

row =find(r_Mars_path(:,4)==Perihelion_Mars);
plot(r_Mars_path(row,5),r_Mars_path(row,6),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerEdg
eColor','r','MarkerFaceColor','r')

row =find(r_Mars_path(:,4)==Aphelion_Mars);
plot(r_Mars_path(row,5),r_Mars_path(row,6),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerEdg
eColor','r')

plot_limit = 1.05 * max(r_Mars_path(:,4));
xlim([-1.434*plot_limit,1.434*plot_limit])
ylim([-plot_limit,plot_limit])
xlabel('Ecliptic plane top view x axis, meters')
ylabel('Ecliptic plane top view y axis, meters')



pause(20)



plot(Jupiter_r_above(:,5),Jupiter_r_above(:,6),'LineWidth',2,'Color','r')
plot(Jupiter_r_below(:,5),Jupiter_r_below(:,6),'LineWidth',2,'Color','r', 'LineStyle', ':')


row =find(r_Jupiter_path(:,4)==Perihelion_Jupiter);
```

```
plot(r_Jupiter_path(row,5),r_Jupiter_path(row,6),'LineStyle','none','Marker','o','MarkerSize',10,'Mar
kerEdgeColor','r','MarkerFaceColor','r')

row =find(r_Jupiter_path(:,4)==Aphelion_Jupiter);
plot(r_Jupiter_path(row,5),r_Jupiter_path(row,6),'LineStyle','none','Marker','o','MarkerSize',10,'Mar
kerEdgeColor','r')



plot(Saturn_r_above(:,5),Saturn_r_above(:,6),'LineWidth',2,'Color',[1, 0.65, 0.35])
plot(Saturn_r_below(:,5),Saturn_r_below(:,6),'LineWidth',2,'Color',[1, 0.65, 0.35], 'LineStyle', ':')

row =find(r_Saturn_path(:,4)==Perihelion_Saturn);
plot(r_Saturn_path(row,5),r_Saturn_path(row,6),'LineStyle','none','Marker','o','MarkerSize',10,'Marke
rEdgeColor',[1, 0.65, 0.35],'MarkerFaceColor',[1, 0.65, 0.35])

row =find(r_Saturn_path(:,4)==Aphelion_Saturn);
plot(r_Saturn_path(row,5),r_Saturn_path(row,6),'LineStyle','none','Marker','o','MarkerSize',10,'Marke
rEdgeColor',[1, 0.65, 0.35])



plot(Uranus_r_above(:,5),Uranus_r_above(:,6),'LineWidth',2,'Color','c')
plot(Uranus_r_below(:,5),Uranus_r_below(:,6),'LineWidth',2,'Color','c', 'LineStyle', ':')

row =find(r_Uranus_path(:,4)==Perihelion_Uranus);
plot(r_Uranus_path(row,5),r_Uranus_path(row,6),'LineStyle','none','Marker','o','MarkerSize',10,'Marke
rEdgeColor','c','MarkerFaceColor','c')

row =find(r_Uranus_path(:,4)==Aphelion_Uranus);
plot(r_Uranus_path(row,5),r_Uranus_path(row,6),'LineStyle','none','Marker','o','MarkerSize',10,'Marke
rEdgeColor','c')



plot(Neptune_r_above(:,5),Neptune_r_above(:,6),'LineWidth',2,'Color','m')
plot(Neptune_r_below(:,5),Neptune_r_below(:,6),'LineWidth',2,'Color','m', 'LineStyle', ':')

row =find(r_Neptune_path(:,4)==Perihelion_Neptune);
plot(r_Neptune_path(row,5),r_Neptune_path(row,6),'LineStyle','none','Marker','o','MarkerSize',10,'Mar
kerEdgeColor','m','MarkerFaceColor','m')

row =find(r_Neptune_path(:,4)==Aphelion_Neptune);
plot(r_Neptune_path(row,5),r_Neptune_path(row,6),'LineStyle','none','Marker','o','MarkerSize',10,'Mar
kerEdgeColor','m')



plot(Pluto_r_above(:,5),Pluto_r_above(:,6),'LineWidth',2,'Color',[1, 0.65, 0.35])
plot(Pluto_r_below(:,5),Pluto_r_below(:,6),'LineWidth',2,'Color',[1, 0.65, 0.35], 'LineStyle', ':')

row =find(r_Pluto_path(:,4)==Perihelion_Pluto);
plot(r_Pluto_path(row,5),r_Pluto_path(row,6),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerE
dgeColor',[1, 0.65, 0.35],'MarkerFaceColor',[1, 0.65, 0.35])

row =find(r_Pluto_path(:,4)==Aphelion_Pluto);
plot(r_Pluto_path(row,5),r_Pluto_path(row,6),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerE
dgeColor',[1, 0.65, 0.35])

plot_limit = 1.05 * max(r_Pluto_path(:,4));
xlim([-1.434*plot_limit,1.434*plot_limit])
ylim([-plot_limit,plot_limit])
xlabel('Ecliptic plane top view x axis, meters')
ylabel('Ecliptic plane top view y axis, meters')


pause(20)

hold('off');



% Side view
% plot surface of Sun
r_Sun = 695700000.0;

as= linspace(0,2*pi,360);
xs = r_Sun * cos(as);
```

```
ys = r_Sun * sin(as);

plot(xs,ys,'Color',[1, 0.65, 0.35],'LineWidth',3)

hold('on')

plot(Mercury_r_above(:,5),Mercury_r_above(:,7),'LineWidth',2,'Color','g')
plot(Mercury_r_below(:,5),Mercury_r_below(:,7),'LineWidth',2,'Color','g', 'LineStyle', ':')

row =find(r_Mercury_path(:,4)==Perihelion_Mercury);
plot(r_Mercury_path(row,5),r_Mercury_path(row,7),'LineStyle','none','Marker','o','MarkerSize',10,'Mar
kerEdgeColor','g','MarkerFaceColor','g')

row =find(r_Mercury_path(:,4)==Aphelion_Mercury);
plot(r_Mercury_path(row,5),r_Mercury_path(row,7),'LineStyle','none','Marker','o','MarkerSize',10,'Mar
kerEdgeColor','g')



plot(Venus_r_above(:,5),Venus_r_above(:,7),'LineWidth',2,'Color',[1, 0.65, 0.35])
plot(Venus_r_below(:,5),Venus_r_below(:,7),'LineWidth',2,'Color',[1, 0.65, 0.35], 'LineStyle', ':')

row =find(r_Venus_path(:,4)==Perihelion_Venus);
plot(r_Venus_path(row,5),r_Venus_path(row,7),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerE
dgeColor',[1, 0.65, 0.35],'MarkerFaceColor',[1, 0.65, 0.35])

row =find(r_Venus_path(:,4)==Aphelion_Venus);
plot(r_Venus_path(row,5),r_Venus_path(row,7),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerE
dgeColor',[1, 0.65, 0.35])



plot(Earth_r_above(:,5),Earth_r_above(:,7),'LineWidth',2,'Color','b')
plot(Earth_r_below(:,5),Earth_r_below(:,7),'LineWidth',2,'Color','b', 'LineStyle', ':')

row =find(r_Earth_path(:,4)==Perihelion_Earth);
plot(r_Earth_path(row,5),r_Earth_path(row,7),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerE
dgeColor','b','MarkerFaceColor','b')

row =find(r_Earth_path(:,4)==Aphelion_Earth);
plot(r_Earth_path(row,5),r_Earth_path(row,7),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerE
dgeColor','b')



plot(Mars_r_above(:,5),Mars_r_above(:,7),'LineWidth',2,'Color','r')
plot(Mars_r_below(:,5),Mars_r_below(:,7),'LineWidth',2,'Color','r', 'LineStyle', ':')

row =find(r_Mars_path(:,4)==Perihelion_Mars);
plot(r_Mars_path(row,5),r_Mars_path(row,7),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerEdg
eColor','r','MarkerFaceColor','r')

row =find(r_Mars_path(:,4)==Aphelion_Mars);
plot(r_Mars_path(row,5),r_Mars_path(row,7),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerEdg
eColor','r')

plot_limit = 1.05 * max(r_Mars_path(:,4));
xlim([-1.434*plot_limit,1.434*plot_limit])
ylim([-plot_limit,plot_limit])
xlabel('Ecliptic plane side view x axis, meters')
ylabel('Ecliptic plane side view z axis, meters')

pause(20)



plot(Jupiter_r_above(:,5),Jupiter_r_above(:,7),'LineWidth',2,'Color','r')
plot(Jupiter_r_below(:,5),Jupiter_r_below(:,7),'LineWidth',2,'Color','r', 'LineStyle', ':')


row =find(r_Jupiter_path(:,4)==Perihelion_Jupiter);
plot(r_Jupiter_path(row,5),r_Jupiter_path(row,7),'LineStyle','none','Marker','o','MarkerSize',10,'Mar
kerEdgeColor','r','MarkerFaceColor','r')

row =find(r_Jupiter_path(:,4)==Aphelion_Jupiter);
plot(r_Jupiter_path(row,5),r_Jupiter_path(row,7),'LineStyle','none','Marker','o','MarkerSize',10,'Mar
kerEdgeColor','r')
```

```
plot(Saturn_r_above(:,5),Saturn_r_above(:,7),'LineWidth',2,'Color',[1, 0.65, 0.35])
plot(Saturn_r_below(:,5),Saturn_r_below(:,7),'LineWidth',2,'Color',[1, 0.65, 0.35], 'LineStyle', ':')

row =find(r_Saturn_path(:,4)==Perihelion_Saturn);
plot(r_Saturn_path(row,5),r_Saturn_path(row,7),'LineStyle','none','Marker','o','MarkerSize',10,'Marke
rEdgeColor',[1, 0.65, 0.35],'MarkerFaceColor',[1, 0.65, 0.35])

row =find(r_Saturn_path(:,4)==Aphelion_Saturn);
plot(r_Saturn_path(row,5),r_Saturn_path(row,7),'LineStyle','none','Marker','o','MarkerSize',10,'Marke
rEdgeColor',[1, 0.65, 0.35])



plot(Uranus_r_above(:,5),Uranus_r_above(:,7),'LineWidth',2,'Color','c')
plot(Uranus_r_below(:,5),Uranus_r_below(:,7),'LineWidth',2,'Color','c', 'LineStyle', ':')

row =find(r_Uranus_path(:,4)==Perihelion_Uranus);
plot(r_Uranus_path(row,5),r_Uranus_path(row,7),'LineStyle','none','Marker','o','MarkerSize',10,'Marke
rEdgeColor','c','MarkerFaceColor','c')

row =find(r_Uranus_path(:,4)==Aphelion_Uranus);
plot(r_Uranus_path(row,5),r_Uranus_path(row,7),'LineStyle','none','Marker','o','MarkerSize',10,'Marke
rEdgeColor','c')



plot(Neptune_r_above(:,5),Neptune_r_above(:,7),'LineWidth',2,'Color','m')
plot(Neptune_r_below(:,5),Neptune_r_below(:,7),'LineWidth',2,'Color','m', 'LineStyle', ':')

row =find(r_Neptune_path(:,4)==Perihelion_Neptune);
plot(r_Neptune_path(row,5),r_Neptune_path(row,7),'LineStyle','none','Marker','o','MarkerSize',10,'Mar
kerEdgeColor','m','MarkerFaceColor','m')

row =find(r_Neptune_path(:,4)==Aphelion_Neptune);
plot(r_Neptune_path(row,5),r_Neptune_path(row,7),'LineStyle','none','Marker','o','MarkerSize',10,'Mar
kerEdgeColor','m')



plot(Pluto_r_above(:,5),Pluto_r_above(:,7),'LineWidth',2,'Color',[1, 0.65, 0.35])
plot(Pluto_r_below(:,5),Pluto_r_below(:,7),'LineWidth',2,'Color',[1, 0.65, 0.35], 'LineStyle', ':')

row =find(r_Pluto_path(:,4)==Perihelion_Pluto);
plot(r_Pluto_path(row,5),r_Pluto_path(row,7),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerE
dgeColor',[1, 0.65, 0.35],'MarkerFaceColor',[1, 0.65, 0.35])

row =find(r_Pluto_path(:,4)==Aphelion_Pluto);
plot(r_Pluto_path(row,5),r_Pluto_path(row,7),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerE
dgeColor',[1, 0.65, 0.35])

plot_limit = 1.05 * max(r_Pluto_path(:,4));
xlim([-1.434*plot_limit,1.434*plot_limit])
ylim([-plot_limit,plot_limit])
xlabel('Ecliptic plane side view x axis, meters')
ylabel('Ecliptic plane side view z axis, meters')


hold('off');
```

Example_3a_input_for_Mercury.m

```
%  Example_3a_input_for_Mercury.m

%  This script is called by Example_3_Solar_system.m

disp(['This example calculates the orbital motion of Mercury'])

disp(['Initial conditions taken from https://nssdc.gsfc.nasa.gov/planetary/factsheet/
mercuryfact.html'])

%  From  https://nssdc.gsfc.nasa.gov/planetary/factsheet/mercuryfact.html
%  Mercury Mean Orbital Elements (J2000)
%  Semimajor axis (AU)                  0.38709893
%  Orbital eccentricity                 0.20563069
%  Orbital inclination (deg)            7.00487
%  Longitude of ascending node (deg)   48.33167
%  Longitude of perihelion (deg)       77.45645
%  Mean Longitude (deg)               252.25084

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

Orb_Inc_Kepler     = rdir * 7.00487;        % degrees, Orbital Inclination
Orb_LA_Kepler      = rdir * 48.33167;       % degrees, Orbital Longitude of Ascending Node
Orb_Ecc_Kepler     = 0.20563069;     % Orbital eccentricity
Orb_LP_Kepler      = rdir * 77.45645;       % degrees, Orbital Longitude of Perihelion
Orb_Period_Kepler  = 87.969 * 86400; % seconds, Orbital Period

%  Calculate the semi-major axis with Kepler's third law to be consistent with the NASA data.

r_semi_Kepler  = ( Orb_Period_Kepler^2 * m* g / 4 / pi / pi )^(1/3);

Perihelion_Kepler = r_semi_Kepler * (1-Orb_Ecc_Kepler); % meters, applied as initial condition
Aphelion_Kepler  = r_semi_Kepler * (1+Orb_Ecc_Kepler); % meters, for comparison with solution

% 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
%   For this example it is done for 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;


%  Define the initial x, y, z,  position coordinates with respect to the center of mass of the
%  governing body, m, in meters.  For this example, it is the closest orbital distance between
%  the Sun and the center of Mercury.

x = rhat(4,1) * Perihelion_Kepler;
y = rhat(4,2) * Perihelion_Kepler;
z = rhat(4,3) * Perihelion_Kepler;

r(1,4) = Perihelion_Kepler;

disp(['Position   x = ' num2str(x,6) ' m,   y = ' num2str(y,6) ' m,  z = ' num2str(z,6) ' m' ])

%   Define vx, vy, vz, Initial velocity in m/s.  If beta_option = 1, these values are treated as
%   bx=vx/c, by=vy/c, bz=vz/c
%
%   For this example it is the maximum orbital velocity (this is at the Perihelion).
%
%   Start with the circular orbital velocity corresponding to the semimajor axis.

%   Calculate beta with TMG equation 7-12

b_circular = sqrt( 1 / ( (cs*cs*r_semi_Kepler / m / g) -1 ) );

%   Calculate c with TMG equation 4-10

c_circular = cr * exp( (2*m*g/cs/cs ) * ( ( r_semi_Kepler-rr ) / (rr*r_semi_Kepler)) );

c_Perihelion  = cr * exp( (2*m*g/cs/cs ) * ( ( Perihelion_Kepler-rr ) / (rr*Perihelion_Kepler)) );

%   Calculate dc with TMG equation 4-14 to reduce numerical errors

g1 = ( 2*m*g/cs/cs )  *  ( ( Perihelion_Kepler - r_semi_Kepler ) / (r_semi_Kepler *
Perihelion_Kepler));

g2 = g1 * g1;
g3 = g2 * g1;
g4 = g3 * g1;
g5 = g4 * g1;
g6 = g5 * g1;
g7 = g6 * g1;
```

```
g8 = g7 * g1;

dc = c_circular*(-g1 -g2/2. -g3/6. -g4/24. -g5/120.0 -g6/720. -g7/5040. -g8/40320);

%   Calculate the initial beta with TMG equations 4-18

b_Perihelion = sqrt( (dc + b_circular*b_circular * c_Perihelion) / c_circular );

beta_option = 1;
bx = b_Perihelion * bhat(4,1);
by = b_Perihelion * bhat(4,2);
bz = b_Perihelion * bhat(4,3);

disp(['Velocity  vx = ' num2str(bx*c_Perihelion,6) ' m/s,  vy = ' num2str(by*c_Perihelion,6) ' m/s,
vz = ' num2str(bz*c_Perihelion,6) ' m/s,  v = ' num2str(b_Perihelion*c_Perihelion,6) ' m/s'])

disp(['Beta    vx/c = ' num2str(bx,6) ', vy/c = ' num2str(by,6) ', vz/c = ' num2str(bz,6) ',   v/c =
' num2str(b_Perihelion,6)])


%   Calculate the Orbit
path_option = 3;

disp(' ')
disp('--- Calculating Mercury orbit ---')

%   Define total rotation, number of points, and arc segments per point for this calculation

atr = 360 * 1.0;
np  = 360;
nac = 1;

disp(['Total rotation angle for calculation = ' num2str(atr,4) ' degrees' ])

disp(['Calculating solution with ' num2str(np) ' points and ' num2str(nac) ' arc segments per
point'])


% Convert total rotation angle to radians for calculation

atr = atr * pi / 180;

TMG_solver

 r_Mercury_path = r;
np_Mercury_path = np;

%   Post-process and display results

disp('--- Mercury Orbital Solution Results -----')

error_t = t - Orb_Period_Kepler;

disp(['Orbital Period = ' num2str( t/86400,6) ' days, delta vs Kepler = ' num2str(error_t,4) '
seconds' ])

disp(['Circumference  = ' num2str( s,12) ' meters' ])

Perihelion_Mercury = min(r(:,4));
  Aphelion_Mercury = max(r(:,4));


error_Perihelion  =  Perihelion_Mercury -  Perihelion_Kepler;
error_Aphelion    =   Aphelion_Mercury -   Aphelion_Kepler;

disp(['Perihelion     = ' num2str(min(r(:,4)),12) ' meters, delta vs Kepler = '
num2str(error_Perihelion,4) ' meters' ])

disp(['Aphelion       = ' num2str(max(r(:,4)),12) ' meters, delta vs Kepler = '
num2str(error_Aphelion,4) ' meters' ])

eccentricity_Mercury = ( max(r(:,4)) - min(r(:,4)) ) / ( max(r(:,4)) + min(r(:,4)) );

error_eccentricity = eccentricity_Mercury - Orb_Ecc_Kepler;

disp(['Eccentricity   = ' num2str( eccentricity_Mercury,8) ', delta vs Kepler = '
num2str(error_eccentricity,4)])
```

```
%  Calculate Mercury's orbital precession

path_option = 1;

disp(' ')
disp('--- Calculating the orbital precession of Mercury caused by the gravitational field of the Sun
---')

atr = 360 * 1.0 * pi / 180;  % Total rotation angle converted to radians for calculation
np  = 32;
nac = 1000;

disp(['Calculating solution with ' num2str(np) ' points and ' num2str(nac) ' arc segments per
point'])

TMG_solver

ab = b(np+1,1)*b(1,1) + b(np+1,2)*b(1,2) + b(np+1,3)*b(1,3);

ab = acos(ab)*180.0/pi;

disp('--- Mercury Orbital Precession Solution Results -----')
disp(['Rotation      = ' num2str((atr*180.0/pi),15) ' degrees' ])
disp(['Rotation time = ' num2str((t/3600/24),15) ' days' ])
disp(['Precession    = ' num2str(ab) ' degrees per orbit' ])
disp(['Precession    = ' num2str((ab*60*60*365.256363004*100*24*3600/t),15) ' arcsecond per
century' ])

%   The following calculation provides a comparison with the General Relativity value.
grp = 24*pi^3*r_semi_Kepler^2/(Orb_Period_Kepler^2*cs^2*(1-Orb_Ecc_Kepler^2));
%   convert to arcseconds per century
grp = (grp*180/pi)*60*60*365.256363004*100*24*3600/t;
disp('--- For comparison only -----')
disp(['--- General Relativity predicted value = ' num2str(grp,6) ' arcsecond per century' ])
```

Example_3b_input_for_Venus.m

```
%  Example_3b_input_for_Venus.m

%  This script is called by Example_3_Solar_system.m

disp(['This example calculates the orbital motion of Venus'])

disp(['Initial conditions taken from https://nssdc.gsfc.nasa.gov/planetary/factsheet/
venusfact.html'])

%  From  https://nssdc.gsfc.nasa.gov/planetary/factsheet/venusfact.html
%  Venus Mean Orbital Elements (J2000)
%  Semimajor axis (AU)                  0.72333199
%  Orbital eccentricity                 0.00677323
%  Orbital inclination (deg)            3.39471
%  Longitude of ascending node (deg)   76.68069
%  Longitude of perihelion (deg)      131.53298
%  Mean Longitude (deg)               181.97973

Orb_Inc_Kepler     = 3.39471;         % degrees, Orbital Inclination
Orb_LA_Kepler      = 76.68069;        % degrees, Orbital Longitude of Ascending Node
Orb_Ecc_Kepler     = 0.00677323;      % Orbital eccentricity
Orb_LP_Kepler      = 131.53298;       % degrees, Orbital Longitude of Perihelion
Orb_Period_Kepler  = 224.701 * 86400; % seconds, Orbital Period

%  Calculate the semi-major axis with Kepler's third law to be consistent with the NASA data.

r_semi_Kepler  = ( Orb_Period_Kepler^2 * m* g / 4 / pi / pi )^(1/3);

Perihelion_Kepler = r_semi_Kepler * (1-Orb_Ecc_Kepler); % meters, applied as initial condition
Aphelion_Kepler  = r_semi_Kepler * (1+Orb_Ecc_Kepler); % meters, for comparison with solution

% 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
%   For this example it is done for 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) =  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;


%  Define the initial x, y, z,  position coordinates with respect to the center of mass of the
%  governing body, m, in meters.  For this example, it is the closest orbital distance between
%  the Sun and the center of Venus.

x = rhat(4,1) * Perihelion_Kepler;
y = rhat(4,2) * Perihelion_Kepler;
z = rhat(4,3) * Perihelion_Kepler;

r(1,4) = Perihelion_Kepler;

disp(['Position   x = ' num2str(x,6) ' m,   y = ' num2str(y,6) ' m,  z = ' num2str(z,6) ' m' ])

%   Define vx, vy, vz, Initial velocity in m/s.  If beta_option = 1, these values are treated as
%   bx=vx/c, by=vy/c, bz=vz/c
%
%   For this example it is the maximum orbital velocity (this is at the Perihelion).
%
%   Start with the circular orbital velocity corresponding to the semimajor axis.

%   Calculate beta with TMG equation 7-12

b_circular = sqrt( 1 / ( (cs*cs*r_semi_Kepler / m / g) -1 ) );

%   Calculate c with TMG equation 4-10

c_circular = cr * exp( (2*m*g/cs/cs ) * ( ( r_semi_Kepler-rr ) / (rr*r_semi_Kepler)) );

c_Perihelion  = cr * exp( (2*m*g/cs/cs ) * ( ( Perihelion_Kepler-rr ) / (rr*Perihelion_Kepler)) );

%   Calculate dc with TMG equation 4-14 to reduce numerical errors

g1 = ( 2*m*g/cs/cs )  *  ( ( Perihelion_Kepler - r_semi_Kepler ) / (r_semi_Kepler *
Perihelion_Kepler));

g2 = g1 * g1;
g3 = g2 * g1;
g4 = g3 * g1;
g5 = g4 * g1;
g6 = g5 * g1;
g7 = g6 * g1;
g8 = g7 * g1;
```

```
dc = c_circular*(-g1 -g2/2. -g3/6. -g4/24. -g5/120.0 -g6/720. -g7/5040. -g8/40320);

%   Calculate the initial beta with TMG equations 4-18

b_Perihelion = sqrt( (dc + b_circular*b_circular * c_Perihelion) / c_circular );

beta_option = 1;
bx = b_Perihelion * bhat(4,1);
by = b_Perihelion * bhat(4,2);
bz = b_Perihelion * bhat(4,3);

disp(['Velocity  vx = ' num2str(bx*c_Perihelion,6) ' m/s,  vy = ' num2str(by*c_Perihelion,6) ' m/s,
vz = ' num2str(bz*c_Perihelion,6) ' m/s,  v = ' num2str(b_Perihelion*c_Perihelion,6) ' m/s'])

disp(['Beta    vx/c = ' num2str(bx,6) ', vy/c = ' num2str(by,6) ', vz/c = ' num2str(bz,6) ',   v/c =
' num2str(b_Perihelion,6)])


%   Calculate the Orbit
path_option = 3;

disp(' ')
disp('--- Calculating Venus orbit ---')

%   Define total rotation, number of points and arc segments per point for this calculation

atr = 360 * 1.0;
np  = 360;
nac = 1;

disp(['Total rotation angle for calculation = ' num2str(atr,4) ' degrees' ])

disp(['Calculating solution with ' num2str(np) ' points and ' num2str(nac) ' arc segments per
point'])


% Convert total rotation angle to radians for calculation

atr = atr * pi / 180;

TMG_solver

 r_Venus_path = r;
np_Venus_path = np;

%   Post-process and display results

disp('--- Venus Orbital Solution Results -----')

error_t = t - Orb_Period_Kepler;

disp(['Orbital Period = ' num2str( t/86400,6) ' days, delta vs Kepler = ' num2str(error_t,4) '
seconds' ])

disp(['Circumference  = ' num2str( s,12) ' meters' ])

Perihelion_Venus = min(r(:,4));
  Aphelion_Venus = max(r(:,4));


error_Perihelion  =  Perihelion_Venus -  Perihelion_Kepler;
error_Aphelion    =   Aphelion_Venus -   Aphelion_Kepler;

disp(['Perihelion     = ' num2str(min(r(:,4)),12) ' meters, delta vs Kepler = '
num2str(error_Perihelion,4) ' meters' ])

disp(['Aphelion       = ' num2str(max(r(:,4)),12) ' meters, delta vs Kepler = '
num2str(error_Aphelion,4) ' meters' ])

eccentricity_Venus = ( max(r(:,4)) - min(r(:,4)) ) / ( max(r(:,4)) + min(r(:,4)) );

error_eccentricity = eccentricity_Venus - Orb_Ecc_Kepler;

disp(['Eccentricity   = ' num2str( eccentricity_Venus,8) ', delta vs Kepler = '
num2str(error_eccentricity,4)])

%  Calculate Venus's orbital precession
```

```
path_option = 1;

disp(' ')
disp('--- Calculating the orbital precession of Venus caused by the gravitational field of the Sun
---')

atr = 360 * 1.0 * pi / 180;  % Total rotation angle converted to radians for calculation
np  = 32;
nac = 100;

disp(['Calculating solution with ' num2str(np) ' points and ' num2str(nac) ' arc segments per
point'])

TMG_solver

ab = b(np+1,1)*b(1,1) + b(np+1,2)*b(1,2) + b(np+1,3)*b(1,3);

ab = acos(ab)*180.0/pi;

disp('--- Venus Orbital Precession Solution Results -----')
disp(['Rotation      = ' num2str((atr*180.0/pi),15) ' degrees' ])
disp(['Rotation time = ' num2str((t/3600/24),15) ' days' ])
disp(['Precession    = ' num2str(ab) ' degrees per orbit' ])
disp(['Precession    = ' num2str((ab*60*60*365.256363004*100*24*3600/t),15) ' arcsecond per
century' ])
%   The following calculation provides a comparison with the General Relativity value.
grp = 24*pi^3*r_semi_Kepler^2/(Orb_Period_Kepler^2*cs^2*(1-Orb_Ecc_Kepler^2));
%   convert to arcseconds per century
grp = (grp*180/pi)*60*60*365.256363004*100*24*3600/t;
disp('--- For comparison only -----')
disp(['--- General Relativity predicted value = ' num2str(grp,6) ' arcsecond per century' ])
```

Example_3c_input_for_Earth.m

```
%  Example_3c_input_for_Earth.m

%  This script is called by Example_3_Solar_system.m

disp(['This example calculates the orbital motion of Earth'])

disp(['Initial conditions taken from https://nssdc.gsfc.nasa.gov/planetary/factsheet/
earthfact.html'])

%  From  https://nssdc.gsfc.nasa.gov/planetary/factsheet/earthfact.html
%  Earth Mean Orbital Elements (J2000)
%  Semimajor axis (AU)                  1.00000011
%  Orbital eccentricity                 0.01671022
%  Orbital inclination (deg)            0.00005
%  Longitude of ascending node (deg)  -11.26064
%  Longitude of perihelion (deg)      102.94719
%  Mean Longitude (deg)               100.46435

Orb_Inc_Kepler     = 0.00005;    % degrees, Orbital Inclination
Orb_LA_Kepler      = -11.26064;  % degrees, Orbital Longitude of Ascending Node
Orb_Ecc_Kepler     = 0.01671022; % Orbital eccentricity
Orb_LP_Kepler      = 102.94719;  % degrees, Orbital Longitude of Perihelion
Orb_Period_Kepler  = 365.256363004 * 86400; % seconds, Orbital Period

%  Calculate the semi-major axis with Kepler's third law to be consistent with the NASA data.

r_semi_Kepler  = ( Orb_Period_Kepler^2 * m* g / 4 / pi / pi )^(1/3);

Perihelion_Kepler = r_semi_Kepler * (1-Orb_Ecc_Kepler); % meters, applied as initial condition
Aphelion_Kepler  = r_semi_Kepler * (1+Orb_Ecc_Kepler); % meters, for comparison with solution

% 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
%   For this example it is done for 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) =  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;


%  Define the initial x, y, z,  position coordinates with respect to the center of mass of the
%  governing body, m, in meters.  For this example, it is the closest orbital distance between
%  the Sun and the center of Earth.

x = rhat(4,1) * Perihelion_Kepler;
y = rhat(4,2) * Perihelion_Kepler;
z = rhat(4,3) * Perihelion_Kepler;

r(1,4) = Perihelion_Kepler;

disp(['Position   x = ' num2str(x,6) ' m,   y = ' num2str(y,6) ' m,  z = ' num2str(z,6) ' m' ])

%   Define vx, vy, vz, Initial velocity in m/s.  If beta_option = 1, these values are treated as
%   bx=vx/c, by=vy/c, bz=vz/c
%
%   For this example it is the maximum orbital velocity (this is at the Perihelion).
%
%   Start with the circular orbital velocity corresponding to the semimajor axis.

%   Calculate beta with TMG equation 7-12

b_circular = sqrt( 1 / ( (cs*cs*r_semi_Kepler / m / g) -1 ) );

%   Calculate c with TMG equation 4-10

c_circular = cr * exp( (2*m*g/cs/cs ) * ( ( r_semi_Kepler-rr ) / (rr*r_semi_Kepler)) );

c_Perihelion  = cr * exp( (2*m*g/cs/cs ) * ( ( Perihelion_Kepler-rr ) / (rr*Perihelion_Kepler)) );

%   Calculate dc with TMG equation 4-14 to reduce numerical errors

g1 = ( 2*m*g/cs/cs )  *  ( ( Perihelion_Kepler - r_semi_Kepler ) / (r_semi_Kepler *
Perihelion_Kepler));

g2 = g1 * g1;
g3 = g2 * g1;
g4 = g3 * g1;
g5 = g4 * g1;
g6 = g5 * g1;
g7 = g6 * g1;
g8 = g7 * g1;
```

```
dc = c_circular*(-g1 -g2/2. -g3/6. -g4/24. -g5/120.0 -g6/720. -g7/5040. -g8/40320);

%   Calculate the initial beta with TMG equations 4-18

b_Perihelion = sqrt( (dc + b_circular*b_circular * c_Perihelion) / c_circular );

beta_option = 1;
bx = b_Perihelion * bhat(4,1);
by = b_Perihelion * bhat(4,2);
bz = b_Perihelion * bhat(4,3);

disp(['Velocity  vx = ' num2str(bx*c_Perihelion,6) ' m/s,  vy = ' num2str(by*c_Perihelion,6) ' m/s,
vz = ' num2str(bz*c_Perihelion,6) ' m/s,  v = ' num2str(b_Perihelion*c_Perihelion,6) ' m/s'])

disp(['Beta    vx/c = ' num2str(bx,6) ', vy/c = ' num2str(by,6) ', vz/c = ' num2str(bz,6) ',   v/c =
' num2str(b_Perihelion,6)])


%   Calculate the Orbit
path_option = 3;

disp(' ')
disp('--- Calculating Earth orbit ---')

%   Define total rotation, number of points, and arc segments per point for this calculation

atr = 360 * 1.0;
np  = 360;
nac = 1;

disp(['Total rotation angle for calculation = ' num2str(atr,4) ' degrees' ])

disp(['Calculating solution with ' num2str(np) ' points and ' num2str(nac) ' arc segments per
point'])


% Convert total rotation angle to radians for calculation

atr = atr * pi / 180;

TMG_solver

 r_Earth_path = r;
np_Earth_path = np;

%   Post-process and display results

disp('--- Earth Orbital Solution Results -----')

error_t = t - Orb_Period_Kepler;

disp(['Orbital Period = ' num2str( t/86400,6) ' days, delta vs Kepler = ' num2str(error_t,4) '
seconds' ])

disp(['Circumference  = ' num2str( s,12) ' meters' ])

Perihelion_Earth = min(r(:,4));
 Aphelion_Earth = max(r(:,4));


error_Perihelion  =  Perihelion_Earth -  Perihelion_Kepler;
error_Aphelion    =   Aphelion_Earth -   Aphelion_Kepler;

disp(['Perihelion     = ' num2str(min(r(:,4)),12) ' meters, delta vs Kepler = '
num2str(error_Perihelion,4) ' meters' ])

disp(['Aphelion       = ' num2str(max(r(:,4)),12) ' meters, delta vs Kepler = '
num2str(error_Aphelion,4) ' meters' ])

eccentricity_Earth = ( max(r(:,4)) - min(r(:,4)) ) / ( max(r(:,4)) + min(r(:,4)) );

error_eccentricity = eccentricity_Earth - Orb_Ecc_Kepler;

disp(['Eccentricity   = ' num2str( eccentricity_Earth,8) ', delta vs Kepler = '
num2str(error_eccentricity,4)])

%  Calculate Earth's orbital precession
```

```
path_option = 1;

disp(' ')
disp('--- Calculating the orbital precession of Earth caused by the gravitational field of the Sun
---')

atr = 360 * 1.0 * pi / 180;  % Total rotation angle converted to radians for calculation
np  = 32;
nac = 100;

disp(['Calculating solution with ' num2str(np) ' points and ' num2str(nac) ' arc segments per
point'])

TMG_solver

ab = b(np+1,1)*b(1,1) + b(np+1,2)*b(1,2) + b(np+1,3)*b(1,3);

ab = acos(ab)*180.0/pi;

disp('--- Earth Orbital Precession Solution Results -----')
disp(['Rotation      = ' num2str((atr*180.0/pi),15) ' degrees' ])
disp(['Rotation time = ' num2str((t/3600/24),15) ' days' ])
disp(['Precession    = ' num2str(ab) ' degrees per orbit' ])
disp(['Precession    = ' num2str((ab*60*60*365.256363004*100*24*3600/t),15) ' arcsecond per
century' ])
%   The following calculation provides a comparison with the General Relativity value.
grp = 24*pi^3*r_semi_Kepler^2/(Orb_Period_Kepler^2*cs^2*(1-Orb_Ecc_Kepler^2));
%   convert to arcseconds per century
grp = (grp*180/pi)*60*60*365.256363004*100*24*3600/t;
disp('--- For comparison only -----')
disp(['--- General Relativity predicted value = ' num2str(grp,6) ' arcsecond per century' ])
```

Example_3d_input_for_Mars.m

```
%  Example_3d_input_for_Mars.m

%  This script is called by Example_3_Solar_system.m

disp(['This example calculates the orbital motion of Mars'])

disp(['Initial conditions taken from https://nssdc.gsfc.nasa.gov/planetary/factsheet/Marsfact.html'])

%  From  https://nssdc.gsfc.nasa.gov/planetary/factsheet/Marsfact.html
%  Mars Mean Orbital Elements (J2000)
%  Semimajor axis (AU)                  1.52366231
%  Orbital eccentricity                 0.09341233
%  Orbital inclination (deg)            1.85061
%  Longitude of ascending node (deg)   49.57854
%  Longitude of perihelion (deg)      336.04084
%  Mean Longitude (deg)               355.45332

Orb_Inc_Kepler     = 1.85061;    % degrees, Orbital Inclination
Orb_LA_Kepler      = 49.57854;   % degrees, Orbital Longitude of Ascending Node
Orb_Ecc_Kepler     = 0.09341233; % Orbital eccentricity
Orb_LP_Kepler      = 336.04084;  % degrees, Orbital Longitude of Perihelion
Orb_Period_Kepler  = 1.88081578 * 365.256363004 * 86400; % seconds, Orbital Period

%  Calculate the semi-major axis with Kepler's third law to be consistent with the NASA data.

r_semi_Kepler  = ( Orb_Period_Kepler^2 * m* g / 4 / pi / pi )^(1/3);

Perihelion_Kepler = r_semi_Kepler * (1-Orb_Ecc_Kepler); % meters, applied as initial condition
Aphelion_Kepler  = r_semi_Kepler * (1+Orb_Ecc_Kepler); % meters, for comparison with solution

% 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
%   For this example it is done for 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) =  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;


%  Define the initial x, y, z,  position coordinates with respect to the center of mass of the
%  governing body, m, in meters.  For this example, it is the closest orbital distance between
%  the Sun and the center of Mars.

x = rhat(4,1) * Perihelion_Kepler;
y = rhat(4,2) * Perihelion_Kepler;
z = rhat(4,3) * Perihelion_Kepler;

r(1,4) = Perihelion_Kepler;

disp(['Position   x = ' num2str(x,6) ' m,   y = ' num2str(y,6) ' m,  z = ' num2str(z,6) ' m' ])

%   Define vx, vy, vz, Initial velocity in m/s.  If beta_option = 1, these values are treated as
%   bx=vx/c, by=vy/c, bz=vz/c
%
%   For this example it is the maximum orbital velocity (this is at the Perihelion).
%
%   Start with the circular orbital velocity corresponding to the semimajor axis.

%   Calculate beta with TMG equation 7-12

b_circular = sqrt( 1 / ( (cs*cs*r_semi_Kepler / m / g) -1 ) );

%   Calculate c with TMG equation 4-10

c_circular = cr * exp( (2*m*g/cs/cs ) * ( ( r_semi_Kepler-rr ) / (rr*r_semi_Kepler)) );

c_Perihelion  = cr * exp( (2*m*g/cs/cs ) * ( ( Perihelion_Kepler-rr ) / (rr*Perihelion_Kepler)) );

%   Calculate dc with TMG equation 4-14 to reduce numerical errors

g1 = ( 2*m*g/cs/cs )  *  ( ( Perihelion_Kepler - r_semi_Kepler ) / (r_semi_Kepler *
Perihelion_Kepler));

g2 = g1 * g1;
g3 = g2 * g1;
g4 = g3 * g1;
g5 = g4 * g1;
g6 = g5 * g1;
g7 = g6 * g1;
g8 = g7 * g1;

dc = c_circular*(-g1 -g2/2. -g3/6. -g4/24. -g5/120.0 -g6/720. -g7/5040. -g8/40320);
```

```
%   Calculate the initial beta with TMG equations 4-18

b_Perihelion = sqrt( (dc + b_circular*b_circular * c_Perihelion) / c_circular );

beta_option = 1;
bx = b_Perihelion * bhat(4,1);
by = b_Perihelion * bhat(4,2);
bz = b_Perihelion * bhat(4,3);

disp(['Velocity  vx = ' num2str(bx*c_Perihelion,6) ' m/s,  vy = ' num2str(by*c_Perihelion,6) ' m/s,
vz = ' num2str(bz*c_Perihelion,6) ' m/s,  v = ' num2str(b_Perihelion*c_Perihelion,6) ' m/s'])

disp(['Beta    vx/c = ' num2str(bx,6) ', vy/c = ' num2str(by,6) ', vz/c = ' num2str(bz,6) ',   v/c =
' num2str(b_Perihelion,6)])


%   Calculate the Orbit
path_option = 3;

disp(' ')
disp('--- Calculating Mars orbit ---')

%   Define total rotation, number of points, and arc segments per point for this calculation

atr = 360 * 1.0;
np  = 360;
nac = 1;

disp(['Total rotation angle for calculation = ' num2str(atr,4) ' degrees' ])

disp(['Calculating solution with ' num2str(np) ' points and ' num2str(nac) ' arc segments per
point'])


% Convert total rotation angle to radians for calculation

atr = atr * pi / 180;

TMG_solver

 r_Mars_path = r;
np_Mars_path = np;

%   Postprocess and display results

disp('--- Mars Orbital Solution Results -----')

error_t = t - Orb_Period_Kepler;

disp(['Orbital Period = ' num2str( t/86400,6) ' days, delta vs Kepler = ' num2str(error_t,4) '
seconds' ])

disp(['Circumference  = ' num2str( s,12) ' meters' ])

Perihelion_Mars = min(r(:,4));
  Aphelion_Mars = max(r(:,4));


error_Perihelion  =  Perihelion_Mars -  Perihelion_Kepler;
error_Aphelion    =   Aphelion_Mars -   Aphelion_Kepler;

disp(['Perihelion     = ' num2str(min(r(:,4)),12) ' meters, delta vs Kepler = '
num2str(error_Perihelion,4) ' meters' ])

disp(['Aphelion       = ' num2str(max(r(:,4)),12) ' meters, delta vs Kepler = '
num2str(error_Aphelion,4) ' meters' ])

eccentricity_Mars = ( max(r(:,4)) - min(r(:,4)) ) / ( max(r(:,4)) + min(r(:,4)) );

error_eccentricity = eccentricity_Mars - Orb_Ecc_Kepler;

disp(['Eccentricity   = ' num2str( eccentricity_Mars,8) ', delta vs Kepler = '
num2str(error_eccentricity,4)])

%  Calculate Mars's orbital precession

path_option = 1;
```

```
disp(' ')
disp('--- Calculating the orbital precession of Mars caused by the gravitational field of the Sun
---')

atr = 360 * 1.0 * pi / 180;  % Total rotation angle converted to radians for calculation
np  = 32;
nac = 100;

disp(['Calculating solution with ' num2str(np) ' points and ' num2str(nac) ' arc segments per
point'])

TMG_solver

ab = b(np+1,1)*b(1,1) + b(np+1,2)*b(1,2) + b(np+1,3)*b(1,3);

ab = acos(ab)*180.0/pi;

disp('--- Mars Orbital Precession Solution Results -----')
disp(['Rotation      = ' num2str((atr*180.0/pi),15) ' degrees' ])
disp(['Rotation time = ' num2str((t/3600/24),15) ' days' ])
disp(['Precession    = ' num2str(ab) ' degrees per orbit' ])
disp(['Precession    = ' num2str((ab*60*60*365.256363004*100*24*3600/t),15) ' arcsecond per
century' ])
%   The following calculation provides a comparison with the General Relativity value.
grp = 24*pi^3*r_semi_Kepler^2/(Orb_Period_Kepler^2*cs^2*(1-Orb_Ecc_Kepler^2));
%   convert to arcseconds per century
grp = (grp*180/pi)*60*60*365.256363004*100*24*3600/t;
disp('--- For comparison only -----')
disp(['--- General Relativity predicted value = ' num2str(grp,6) ' arcsecond per century' ])
```

Example_3e_input_for_Jupiter.m

```
%  Example_3e_input_for_Jupiter.m

%  This script is called by Example_3_Solar_system.m

disp(['This example calculates the orbital motion of Jupiter'])

disp(['Initial conditions taken from https://nssdc.gsfc.nasa.gov/planetary/factsheet/
jupiterfact.html'])

%  From  https://nssdc.gsfc.nasa.gov/planetary/factsheet/jupiterfact.html
%  Jupiter Mean Orbital Elements (J2000)

%  Semimajor axis (AU)                  5.20336301
%  Orbital eccentricity                 0.04839266
%  Orbital inclination (deg)            1.30530
%  Longitude of ascending node (deg)  100.55615
%  Longitude of perihelion (deg)       14.75385
%  Mean Longitude (deg)                34.40438

Orb_Inc_Kepler     = 1.30530;    % degrees, Orbital Inclination
Orb_LA_Kepler      = 100.55615;  % degrees, Orbital Longitude of Ascending Node
Orb_Ecc_Kepler     = 0.04839266; % Orbital eccentricity
Orb_LP_Kepler      = 14.75385;   % degrees, Orbital Longitude of Perihelion
Orb_Period_Kepler  = 4332.589 * 86400; % seconds, Orbital Period

%  Calculate the semi-major axis with Kepler's third law to be consistent with the NASA data.

r_semi_Kepler  = ( Orb_Period_Kepler^2 * m* g / 4 / pi / pi )^(1/3);

Perihelion_Kepler = r_semi_Kepler * (1-Orb_Ecc_Kepler); % meters, applied as initial condition
Aphelion_Kepler  = r_semi_Kepler * (1+Orb_Ecc_Kepler); % meters, for comparison with solution

% 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
%   For this example it is done for 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) =  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;


%  Define the initial x, y, z,  position coordinates with respect to the center of mass of the
%  governing body, m, in meters.  For this example, it is the closest orbital distance between
%  the Sun and the center of Jupiter.

x = rhat(4,1) * Perihelion_Kepler;
y = rhat(4,2) * Perihelion_Kepler;
z = rhat(4,3) * Perihelion_Kepler;

r(1,4) = Perihelion_Kepler;

disp(['Position   x = ' num2str(x,6) ' m,   y = ' num2str(y,6) ' m,  z = ' num2str(z,6) ' m' ])

%   Define vx, vy, vz, Initial velocity in m/s.  If beta_option = 1, these values are treated as
%   bx=vx/c, by=vy/c, bz=vz/c
%
%   For this example it is the maximum orbital velocity (this is at the Perihelion).
%
%   Start with the circular orbital velocity corresponding to the semimajor axis.

%   Calculate beta with TMG equation 7-12

b_circular = sqrt( 1 / ( (cs*cs*r_semi_Kepler / m / g) -1 ) );

%   Calculate c with TMG equation 4-10

c_circular = cr * exp( (2*m*g/cs/cs ) * ( ( r_semi_Kepler-rr ) / (rr*r_semi_Kepler)) );

c_Perihelion  = cr * exp( (2*m*g/cs/cs ) * ( ( Perihelion_Kepler-rr ) / (rr*Perihelion_Kepler)) );

%   Calculate dc with TMG equation 4-14 to reduce numerical errors

g1 = ( 2*m*g/cs/cs )  *  ( ( Perihelion_Kepler - r_semi_Kepler ) / (r_semi_Kepler *
Perihelion_Kepler));

g2 = g1 * g1;
g3 = g2 * g1;
g4 = g3 * g1;
g5 = g4 * g1;
g6 = g5 * g1;
g7 = g6 * g1;
g8 = g7 * g1;
```

```
dc = c_circular*(-g1 -g2/2. -g3/6. -g4/24. -g5/120.0 -g6/720. -g7/5040. -g8/40320);

%   Calculate the initial beta with TMG equations 4-18

b_Perihelion = sqrt( (dc + b_circular*b_circular * c_Perihelion) / c_circular );

beta_option = 1;
bx = b_Perihelion * bhat(4,1);
by = b_Perihelion * bhat(4,2);
bz = b_Perihelion * bhat(4,3);

disp(['Velocity  vx = ' num2str(bx*c_Perihelion,6) ' m/s,  vy = ' num2str(by*c_Perihelion,6) ' m/s,
vz = ' num2str(bz*c_Perihelion,6) ' m/s,  v = ' num2str(b_Perihelion*c_Perihelion,6) ' m/s'])

disp(['Beta    vx/c = ' num2str(bx,6) ', vy/c = ' num2str(by,6) ', vz/c = ' num2str(bz,6) ',   v/c =
' num2str(b_Perihelion,6)])


%   Calculate the Orbit
path_option = 3;

disp(' ')
disp('--- Calculating Jupiter orbit ---')

%   Define total rotation, number of points, and arc segments per point for this calculation

atr = 360 * 1.0;
np  = 360;
nac = 1;

disp(['Total rotation angle for calculation = ' num2str(atr,4) ' degrees' ])

disp(['Calculating solution with ' num2str(np) ' points and ' num2str(nac) ' arc segments per
point'])


% Convert total rotation angle to radians for calculation

atr = atr * pi / 180;

TMG_solver

 r_Jupiter_path = r;
np_Jupiter_path = np;

%   Post-process and display results

disp('--- Jupiter Orbital Solution Results -----')

error_t = t - Orb_Period_Kepler;

disp(['Orbital Period = ' num2str( t/86400,6) ' days, delta vs Kepler = ' num2str(error_t,4) '
seconds' ])

disp(['Circumference  = ' num2str( s,12) ' meters' ])

Perihelion_Jupiter = min(r(:,4));
  Aphelion_Jupiter = max(r(:,4));


error_Perihelion  =  Perihelion_Jupiter -  Perihelion_Kepler;
error_Aphelion    =   Aphelion_Jupiter -   Aphelion_Kepler;

disp(['Perihelion     = ' num2str(min(r(:,4)),12) ' meters, delta vs Kepler = '
num2str(error_Perihelion,4) ' meters' ])

disp(['Aphelion       = ' num2str(max(r(:,4)),12) ' meters, delta vs Kepler = '
num2str(error_Aphelion,4) ' meters' ])

eccentricity_Jupiter = ( max(r(:,4)) - min(r(:,4)) ) / ( max(r(:,4)) + min(r(:,4)) );

error_eccentricity = eccentricity_Jupiter - Orb_Ecc_Kepler;

disp(['Eccentricity   = ' num2str( eccentricity_Jupiter,8) ', delta vs Kepler = '
num2str(error_eccentricity,4)])

%  Calculate Jupiter's orbital precession
```

```
path_option = 1;

disp(' ')
disp('--- Calculating the orbital precession of Jupiter caused by the gravitational field of the Sun
---')

atr = 360 * 1.0 * pi / 180;  % Total rotation angle converted to radians for calculation
np  = 32;
nac = 100;

disp(['Calculating solution with ' num2str(np) ' points and ' num2str(nac) ' arc segments per
point'])

TMG_solver

ab = b(np+1,1)*b(1,1) + b(np+1,2)*b(1,2) + b(np+1,3)*b(1,3);

if ab>1
    ab = 1;
end

ab = acos(ab)*180.0/pi;

disp('--- Jupiter Orbital Precession Solution Results -----')
disp(['Rotation      = ' num2str((atr*180.0/pi),15) ' degrees' ])
disp(['Rotation time = ' num2str((t/3600/24),15) ' days' ])
disp(['Precession    = ' num2str(ab) ' degrees per orbit' ])
disp(['Precession    = ' num2str((ab*60*60*365.256363004*100*24*3600/t),15) ' arcsecond per
century' ])
%   The following calculation provides a comparison with the General Relativity value.
grp = 24*pi^3*r_semi_Kepler^2/(Orb_Period_Kepler^2*cs^2*(1-Orb_Ecc_Kepler^2));
%   convert to arcseconds per century
grp = (grp*180/pi)*60*60*365.256363004*100*24*3600/t;
disp('--- For comparison only -----')
disp(['--- General Relativity predicted value = ' num2str(grp,6) ' arcsecond per century' ])
```

Example_3f_input_for_Saturn.m

```
%  Example_3f_input_for_Saturn.m

%  This script is called by Example_3_Solar_system.m

disp(['This example calculates the orbital motion of Saturn'])

disp(['Initial conditions taken from https://nssdc.gsfc.nasa.gov/planetary/factsheet/
saturnfact.html'])

%  From  https://nssdc.gsfc.nasa.gov/planetary/factsheet/saturnfact.html
%  Saturn Mean Orbital Elements (J2000)

%  Semimajor axis (AU)                  9.53707032
%  Orbital eccentricity                 0.05415060
%  Orbital inclination (deg)            2.48446
%  Longitude of ascending node (deg)  113.71504
%  Longitude of perihelion (deg)       92.43194
%  Mean Longitude (deg)                49.94432

Orb_Inc_Kepler     = 2.48446;    % degrees, Orbital Inclination
Orb_LA_Kepler      = 113.71504;  % degrees, Orbital Longitude of Ascending Node
Orb_Ecc_Kepler     = 0.05415060; % Orbital eccentricity
Orb_LP_Kepler      = 92.43194;   % degrees, Orbital Longitude of Perihelion
Orb_Period_Kepler  = 10759.22 * 86400; % seconds, Orbital Period

%  Calculate the semi-major axis with Kepler's third law to be consistent with the NASA data.

r_semi_Kepler  = ( Orb_Period_Kepler^2 * m* g / 4 / pi / pi )^(1/3);

Perihelion_Kepler = r_semi_Kepler * (1-Orb_Ecc_Kepler); % meters, applied as initial condition
Aphelion_Kepler  = r_semi_Kepler * (1+Orb_Ecc_Kepler); % meters, for comparison with solution

% 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
%   For this example it is done for 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) =  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;


%  Define the initial x, y, z,  position coordinates with respect to the center of mass of the
%  governing body, m, in meters.  For this example, it is the closest orbital distance between
%  the Sun and the center of Saturn.

x = rhat(4,1) * Perihelion_Kepler;
y = rhat(4,2) * Perihelion_Kepler;
z = rhat(4,3) * Perihelion_Kepler;

r(1,4) = Perihelion_Kepler;

disp(['Position   x = ' num2str(x,6) ' m,   y = ' num2str(y,6) ' m,  z = ' num2str(z,6) ' m' ])

%   Define vx, vy, vz, Initial velocity in m/s.  If beta_option = 1, these values are treated as
%   bx=vx/c, by=vy/c, bz=vz/c
%
%   For this example it is the maximum orbital velocity (this is at the Perihelion).
%
%   Start with the circular orbital velocity corresponding to the semimajor axis.

%   Calculate beta with TMG equation 7-12

b_circular = sqrt( 1 / ( (cs*cs*r_semi_Kepler / m / g) -1 ) );

%   Calculate c with TMG equation 4-10

c_circular = cr * exp( (2*m*g/cs/cs ) * ( ( r_semi_Kepler-rr ) / (rr*r_semi_Kepler)) );

c_Perihelion  = cr * exp( (2*m*g/cs/cs ) * ( ( Perihelion_Kepler-rr ) / (rr*Perihelion_Kepler)) );

%   Calculate dc with TMG equation 4-14 to reduce numerical errors

g1 = ( 2*m*g/cs/cs )  *  ( ( Perihelion_Kepler - r_semi_Kepler ) / (r_semi_Kepler *
Perihelion_Kepler));

g2 = g1 * g1;
g3 = g2 * g1;
g4 = g3 * g1;
g5 = g4 * g1;
g6 = g5 * g1;
g7 = g6 * g1;
g8 = g7 * g1;
```

```
dc = c_circular*(-g1 -g2/2. -g3/6. -g4/24. -g5/120.0 -g6/720. -g7/5040. -g8/40320);

%   Calculate the initial beta with TMG equations 4-18

b_Perihelion = sqrt( (dc + b_circular*b_circular * c_Perihelion) / c_circular );

beta_option = 1;
bx = b_Perihelion * bhat(4,1);
by = b_Perihelion * bhat(4,2);
bz = b_Perihelion * bhat(4,3);

disp(['Velocity  vx = ' num2str(bx*c_Perihelion,6) ' m/s,  vy = ' num2str(by*c_Perihelion,6) ' m/s,
vz = ' num2str(bz*c_Perihelion,6) ' m/s,  v = ' num2str(b_Perihelion*c_Perihelion,6) ' m/s'])

disp(['Beta    vx/c = ' num2str(bx,6) ', vy/c = ' num2str(by,6) ', vz/c = ' num2str(bz,6) ',   v/c =
' num2str(b_Perihelion,6)])


%   Calculate the Orbit
path_option = 3;

disp(' ')
disp('--- Calculating Saturn orbit ---')

%   Define total rotation, number of points, and arc segments per point for this calculation

atr = 360 * 1.0;
np  = 360;
nac = 1;

disp(['Total rotation angle for calculation = ' num2str(atr,4) ' degrees' ])

disp(['Calculating solution with ' num2str(np) ' points and ' num2str(nac) ' arc segments per
point'])


% Convert total rotation angle to radians for calculation

atr = atr * pi / 180;

TMG_solver

 r_Saturn_path = r;
np_Saturn_path = np;

%   Post-process and display results

disp('--- Saturn Orbital Solution Results -----')

error_t = t - Orb_Period_Kepler;

disp(['Orbital Period = ' num2str( t/86400,6) ' days, delta vs Kepler = ' num2str(error_t,4) '
seconds' ])

disp(['Circumference  = ' num2str( s,12) ' meters' ])

Perihelion_Saturn = min(r(:,4));
  Aphelion_Saturn = max(r(:,4));


error_Perihelion  =  Perihelion_Saturn -  Perihelion_Kepler;
error_Aphelion    =   Aphelion_Saturn -   Aphelion_Kepler;

disp(['Perihelion     = ' num2str(min(r(:,4)),12) ' meters, delta vs Kepler = '
num2str(error_Perihelion,4) ' meters' ])

disp(['Aphelion       = ' num2str(max(r(:,4)),12) ' meters, delta vs Kepler = '
num2str(error_Aphelion,4) ' meters' ])

eccentricity_Saturn = ( max(r(:,4)) - min(r(:,4)) ) / ( max(r(:,4)) + min(r(:,4)) );

error_eccentricity = eccentricity_Saturn - Orb_Ecc_Kepler;

disp(['Eccentricity   = ' num2str( eccentricity_Saturn,8) ', delta vs Kepler = '
num2str(error_eccentricity,4)])

%  Calculate Saturn's orbital precession
```

```
path_option = 1;

disp(' ')
disp('--- Calculating the orbital precession of Saturn caused by the gravitational field of the Sun
---')

atr = 360 * 1.0 * pi / 180;  % Total rotation angle converted to radians for calculation
np  = 32;
nac = 100;

disp(['Calculating solution with ' num2str(np) ' points and ' num2str(nac) ' arc segments per
point'])

TMG_solver

ab = b(np+1,1)*b(1,1) + b(np+1,2)*b(1,2) + b(np+1,3)*b(1,3);

if ab>1
    ab = 1;
end

ab = acos(ab)*180.0/pi;

disp('--- Saturn Orbital Precession Solution Results -----')
disp(['Rotation      = ' num2str((atr*180.0/pi),15) ' degrees' ])
disp(['Rotation time = ' num2str((t/3600/24),15) ' days' ])
disp(['Precession    = ' num2str(ab) ' degrees per orbit' ])
disp(['Precession    = ' num2str((ab*60*60*365.256363004*100*24*3600/t),15) ' arcsecond per
century' ])
%   The following calculation provides a comparison with the General Relativity value.
grp = 24*pi^3*r_semi_Kepler^2/(Orb_Period_Kepler^2*cs^2*(1-Orb_Ecc_Kepler^2));
%   convert to arcseconds per century
grp = (grp*180/pi)*60*60*365.256363004*100*24*3600/t;
disp('--- For comparison only -----')
disp(['--- General Relativity predicted value = ' num2str(grp,6) ' arcsecond per century' ])
```

Example_3g_input_for_Uranus.m

```
%  Example_3g_input_for_Uranus.m

%  This script is called by Example_3_Solar_system.m

disp(['This example calculates the orbital motion of Uranus'])

disp(['Initial conditions taken from https://nssdc.gsfc.nasa.gov/planetary/factsheet/
uranusfact.html'])

%  From  https://nssdc.gsfc.nasa.gov/planetary/factsheet/uranusfact.html
%  Uranus Mean Orbital Elements (J2000)

%  Semimajor axis (AU)                 19.19126393
%  Orbital eccentricity                 0.04716771
%  Orbital inclination (deg)            0.76986
%  Longitude of ascending node (deg)   74.22988
%  Longitude of perihelion (deg)      170.96424
%  Mean Longitude (deg)               313.23218

Orb_Inc_Kepler     = 0.76986;     % degrees, Orbital Inclination
Orb_LA_Kepler      = 74.22988;    % degrees, Orbital Longitude of Ascending Node
Orb_Ecc_Kepler     = 0.04716771;  % Orbital eccentricity
Orb_LP_Kepler      = 170.96424;   % degrees, Orbital Longitude of Perihelion
Orb_Period_Kepler  = 30685.4 * 86400; % seconds, Orbital Period

%  Calculate the semi-major axis with Kepler's third law to be consistent with the NASA data.

r_semi_Kepler  = ( Orb_Period_Kepler^2 * m* g / 4 / pi / pi )^(1/3);

Perihelion_Kepler = r_semi_Kepler * (1-Orb_Ecc_Kepler); % meters, applied as initial condition
Aphelion_Kepler  = r_semi_Kepler * (1+Orb_Ecc_Kepler); % meters, for comparison with solution

% 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
%   For this example it is done for 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) =  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;


%  Define the initial x, y, z,  position coordinates with respect to the center of mass of the
%  governing body, m, in meters.  For this example, it is the closest orbital distance between
%  the Sun and the center of Uranus.

x = rhat(4,1) * Perihelion_Kepler;
y = rhat(4,2) * Perihelion_Kepler;
z = rhat(4,3) * Perihelion_Kepler;

r(1,4) = Perihelion_Kepler;

disp(['Position   x = ' num2str(x,6) ' m,   y = ' num2str(y,6) ' m,  z = ' num2str(z,6) ' m' ])

%   Define vx, vy, vz, Initial velocity in m/s.  If beta_option = 1, these values are treated as
%   bx=vx/c, by=vy/c, bz=vz/c
%
%   For this example it is the maximum orbital velocity (this is at the Perihelion).
%
%   Start with the circular orbital velocity corresponding to the semimajor axis.

%   Calculate beta with TMG equation 7-12

b_circular = sqrt( 1 / ( (cs*cs*r_semi_Kepler / m / g) -1 ) );

%   Calculate c with TMG equation 4-10

c_circular = cr * exp( (2*m*g/cs/cs ) * ( ( r_semi_Kepler-rr ) / (rr*r_semi_Kepler)) );

c_Perihelion  = cr * exp( (2*m*g/cs/cs ) * ( ( Perihelion_Kepler-rr ) / (rr*Perihelion_Kepler)) );

%   Calculate dc with TMG equation 4-14 to reduce numerical errors

g1 = ( 2*m*g/cs/cs )  *  ( ( Perihelion_Kepler - r_semi_Kepler ) / (r_semi_Kepler *
Perihelion_Kepler));

g2 = g1 * g1;
g3 = g2 * g1;
g4 = g3 * g1;
g5 = g4 * g1;
g6 = g5 * g1;
g7 = g6 * g1;
g8 = g7 * g1;
```

```
dc = c_circular*(-g1 -g2/2. -g3/6. -g4/24. -g5/120.0 -g6/720. -g7/5040. -g8/40320);

%   Calculate the initial beta with TMG equations 4-18

b_Perihelion = sqrt( (dc + b_circular*b_circular * c_Perihelion) / c_circular );

beta_option = 1;
bx = b_Perihelion * bhat(4,1);
by = b_Perihelion * bhat(4,2);
bz = b_Perihelion * bhat(4,3);

disp(['Velocity  vx = ' num2str(bx*c_Perihelion,6) ' m/s,  vy = ' num2str(by*c_Perihelion,6) ' m/s,
vz = ' num2str(bz*c_Perihelion,6) ' m/s,  v = ' num2str(b_Perihelion*c_Perihelion,6) ' m/s'])

disp(['Beta    vx/c = ' num2str(bx,6) ', vy/c = ' num2str(by,6) ', vz/c = ' num2str(bz,6) ',   v/c =
' num2str(b_Perihelion,6)])


%   Calculate the Orbit
path_option = 3;

disp(' ')
disp('--- Calculating Uranus orbit ---')

%   Define total rotation, number of points, and arc segments per point for this calculation

atr = 360 * 1.0;
np  = 360;
nac = 1;

disp(['Total rotation angle for calculation = ' num2str(atr,4) ' degrees' ])

disp(['Calculating solution with ' num2str(np) ' points and ' num2str(nac) ' arc segments per
point'])


% Convert total rotation angle to radians for calculation

atr = atr * pi / 180;

TMG_solver

 r_Uranus_path = r;
np_Uranus_path = np;

%   Post-process and display results

disp('--- Uranus Orbital Solution Results -----')

error_t = t - Orb_Period_Kepler;

disp(['Orbital Period = ' num2str( t/86400,6) ' days, delta vs Kepler = ' num2str(error_t,4) '
seconds' ])

disp(['Circumference  = ' num2str( s,12) ' meters' ])

Perihelion_Uranus = min(r(:,4));
  Aphelion_Uranus = max(r(:,4));


error_Perihelion  =  Perihelion_Uranus -  Perihelion_Kepler;
error_Aphelion    =   Aphelion_Uranus -   Aphelion_Kepler;

disp(['Perihelion     = ' num2str(min(r(:,4)),12) ' meters, delta vs Kepler = '
num2str(error_Perihelion,4) ' meters' ])

disp(['Aphelion       = ' num2str(max(r(:,4)),12) ' meters, delta vs Kepler = '
num2str(error_Aphelion,4) ' meters' ])

eccentricity_Uranus = ( max(r(:,4)) - min(r(:,4)) ) / ( max(r(:,4)) + min(r(:,4)) );

error_eccentricity = eccentricity_Uranus - Orb_Ecc_Kepler;

disp(['Eccentricity   = ' num2str( eccentricity_Uranus,8) ', delta vs Kepler = '
num2str(error_eccentricity,4)])

%  Calculate Uranus's orbital precession
```

```
path_option = 1;

disp(' ')
disp('--- Calculating the orbital precession of Uranus caused by the gravitational field of the Sun
---')

atr = 360 * 1.0 * pi / 180;  % Total rotation angle converted to radians for calculation
np  = 32;
nac = 100;

disp(['Calculating solution with ' num2str(np) ' points and ' num2str(nac) ' arc segments per
point'])

TMG_solver

ab = b(np+1,1)*b(1,1) + b(np+1,2)*b(1,2) + b(np+1,3)*b(1,3);

if ab>1
    ab = 1;
end

ab = acos(ab)*180.0/pi;

disp('--- Uranus Orbital Precession Solution Results -----')
disp(['Rotation      = ' num2str((atr*180.0/pi),15) ' degrees' ])
disp(['Rotation time = ' num2str((t/3600/24),15) ' days' ])
disp(['Precession    = ' num2str(ab) ' degrees per orbit' ])
disp(['Precession    = ' num2str((ab*60*60*365.256363004*100*24*3600/t),15) ' arcsecond per
century' ])
%   The following calculation provides a comparison with the General Relativity value.
grp = 24*pi^3*r_semi_Kepler^2/(Orb_Period_Kepler^2*cs^2*(1-Orb_Ecc_Kepler^2));
%   convert to arcseconds per century
grp = (grp*180/pi)*60*60*365.256363004*100*24*3600/t;
disp('--- For comparison only -----')
disp(['--- General Relativity predicted value = ' num2str(grp,6) ' arcsecond per century' ])
```

Example_3h_input_for_Neptune.m

```
%  Example_3h_input_for_Neptune.m

%  This script is called by Example_3_Solar_system.m

disp(['This example calculates the orbital motion of Neptune'])

disp(['Initial conditions taken from https://nssdc.gsfc.nasa.gov/planetary/factsheet/
neptunefact.html'])

%  From  https://nssdc.gsfc.nasa.gov/planetary/factsheet/neptunefact.html
%  Neptune Mean Orbital Elements (J2000)

%  Semimajor axis (AU)                 30.06896348
%  Orbital eccentricity                 0.00858587
%  Orbital inclination (deg)            1.76917
%  Longitude of ascending node (deg)  131.72169
%  Longitude of perihelion (deg)       44.97135
%  Mean Longitude (deg)               304.88003

Orb_Inc_Kepler     = 1.76917;     % degrees, Orbital Inclination
Orb_LA_Kepler      = 131.72169;    % degrees, Orbital Longitude of Ascending Node
Orb_Ecc_Kepler     = 0.00858587;  % Orbital eccentricity
Orb_LP_Kepler      = 44.97135;   % degrees, Orbital Longitude of Perihelion
Orb_Period_Kepler  = 60189.0 * 86400; % seconds, Orbital Period

%  Calculate the semi-major axis with Kepler's third law to be consistent with the NASA data.

r_semi_Kepler  = ( Orb_Period_Kepler^2 * m* g / 4 / pi / pi )^(1/3);

Perihelion_Kepler = r_semi_Kepler * (1-Orb_Ecc_Kepler); % meters, applied as initial condition
Aphelion_Kepler  = r_semi_Kepler * (1+Orb_Ecc_Kepler); % meters, for comparison with solution

% 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
%   For this example it is done for 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) =  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;


%  Define the initial x, y, z,  position coordinates with respect to the center of mass of the
%  governing body, m, in meters.  For this example, it is the closest orbital distance between
%  the Sun and the center of Neptune.

x = rhat(4,1) * Perihelion_Kepler;
y = rhat(4,2) * Perihelion_Kepler;
z = rhat(4,3) * Perihelion_Kepler;

r(1,4) = Perihelion_Kepler;

disp(['Position   x = ' num2str(x,6) ' m,   y = ' num2str(y,6) ' m,  z = ' num2str(z,6) ' m' ])

%   Define vx, vy, vz, Initial velocity in m/s.  If beta_option = 1, these values are treated as
%   bx=vx/c, by=vy/c, bz=vz/c
%
%   For this example it is the maximum orbital velocity (this is at the Perihelion).
%
%   Start with the circular orbital velocity corresponding to the semimajor axis.

%   Calculate beta with TMG equation 7-12

b_circular = sqrt( 1 / ( (cs*cs*r_semi_Kepler / m / g) -1 ) );

%   Calculate c with TMG equation 4-10

c_circular = cr * exp( (2*m*g/cs/cs ) * ( ( r_semi_Kepler-rr ) / (rr*r_semi_Kepler)) );

c_Perihelion  = cr * exp( (2*m*g/cs/cs ) * ( ( Perihelion_Kepler-rr ) / (rr*Perihelion_Kepler)) );

%   Calculate dc with TMG equation 4-14 to reduce numerical errors

g1 = ( 2*m*g/cs/cs )  *  ( ( Perihelion_Kepler - r_semi_Kepler ) / (r_semi_Kepler *
Perihelion_Kepler));

g2 = g1 * g1;
g3 = g2 * g1;
g4 = g3 * g1;
g5 = g4 * g1;
g6 = g5 * g1;
g7 = g6 * g1;
g8 = g7 * g1;
```

```
dc = c_circular*(-g1 -g2/2. -g3/6. -g4/24. -g5/120.0 -g6/720. -g7/5040. -g8/40320);

%   Calculate the initial beta with TMG equations 4-18

b_Perihelion = sqrt( (dc + b_circular*b_circular * c_Perihelion) / c_circular );

beta_option = 1;
bx = b_Perihelion * bhat(4,1);
by = b_Perihelion * bhat(4,2);
bz = b_Perihelion * bhat(4,3);

disp(['Velocity  vx = ' num2str(bx*c_Perihelion,6) ' m/s,  vy = ' num2str(by*c_Perihelion,6) ' m/s,
vz = ' num2str(bz*c_Perihelion,6) ' m/s,  v = ' num2str(b_Perihelion*c_Perihelion,6) ' m/s'])

disp(['Beta    vx/c = ' num2str(bx,6) ', vy/c = ' num2str(by,6) ', vz/c = ' num2str(bz,6) ',   v/c =
' num2str(b_Perihelion,6)])


%   Calculate the Orbit
path_option = 3;

disp(' ')
disp('--- Calculating Neptune orbit ---')

%   Define total rotation, number of points, and arc segments per point for this calculation

atr = 360 * 1.0;
np  = 360;
nac = 1;

disp(['Total rotation angle for calculation = ' num2str(atr,4) ' degrees' ])

disp(['Calculating solution with ' num2str(np) ' points and ' num2str(nac) ' arc segments per
point'])


% Convert total rotation angle to radians for calculation

atr = atr * pi / 180;

TMG_solver

 r_Neptune_path = r;
np_Neptune_path = np;

%   Post-process and display results

disp('--- Neptune Orbital Solution Results -----')

error_t = t - Orb_Period_Kepler;

disp(['Orbital Period = ' num2str( t/86400,6) ' days, delta vs Kepler = ' num2str(error_t,4) '
seconds' ])

disp(['Circumference  = ' num2str( s,12) ' meters' ])

Perihelion_Neptune = min(r(:,4));
  Aphelion_Neptune = max(r(:,4));


error_Perihelion  =  Perihelion_Neptune -  Perihelion_Kepler;
error_Aphelion    =   Aphelion_Neptune -   Aphelion_Kepler;

disp(['Perihelion     = ' num2str(min(r(:,4)),12) ' meters, delta vs Kepler = '
num2str(error_Perihelion,4) ' meters' ])

disp(['Aphelion       = ' num2str(max(r(:,4)),12) ' meters, delta vs Kepler = '
num2str(error_Aphelion,4) ' meters' ])

eccentricity_Neptune = ( max(r(:,4)) - min(r(:,4)) ) / ( max(r(:,4)) + min(r(:,4)) );

error_eccentricity = eccentricity_Neptune - Orb_Ecc_Kepler;

disp(['Eccentricity   = ' num2str( eccentricity_Neptune,8) ', delta vs Kepler = '
num2str(error_eccentricity,4)])

%  Calculate Neptune's orbital precession
```

```
path_option = 1;

disp(' ')
disp('--- Calculating the orbital precession of Neptune caused by the gravitational field of the Sun
---')

atr = 360 * 1.0 * pi / 180;  % Total rotation angle converted to radians for calculation
np  = 32;
nac = 100;

disp(['Calculating solution with ' num2str(np) ' points and ' num2str(nac) ' arc segments per
point'])

TMG_solver

ab = b(np+1,1)*b(1,1) + b(np+1,2)*b(1,2) + b(np+1,3)*b(1,3);

if ab>1
    ab = 1;
end

ab = acos(ab)*180.0/pi;

disp('--- Neptune Orbital Precession Solution Results -----')
disp(['Rotation      = ' num2str((atr*180.0/pi),15) ' degrees' ])
disp(['Rotation time = ' num2str((t/3600/24),15) ' days' ])
disp(['Precession    = ' num2str(ab) ' degrees per orbit' ])
disp(['Precession    = ' num2str((ab*60*60*365.256363004*100*24*3600/t),15) ' arcsecond per
century' ])
%   The following calculation provides a comparison with the General Relativity value.
grp = 24*pi^3*r_semi_Kepler^2/(Orb_Period_Kepler^2*cs^2*(1-Orb_Ecc_Kepler^2));
%   convert to arcseconds per century
grp = (grp*180/pi)*60*60*365.256363004*100*24*3600/t;
disp('--- For comparison only -----')
disp(['--- General Relativity predicted value = ' num2str(grp,6) ' arcsecond per century' ])
```

Example_3i_input_for_Pluto.m

```
%  Example_3i_input_for_Pluto.m

%  This script is called by Example_3_Solar_system.m

disp(['This example calculates the orbital motion of Pluto'])

disp(['Initial conditions taken from https://nssdc.gsfc.nasa.gov/planetary/factsheet/
Plutofact.html'])

%  From  https://nssdc.gsfc.nasa.gov/planetary/factsheet/Plutofact.html
%  Pluto Mean Orbital Elements (J2000)

%  Semimajor axis (AU)                 39.48168677
%  Orbital eccentricity                 0.24880766
%  Orbital inclination (deg)           17.14175
%  Longitude of ascending node (deg)  110.30347
%  Longitude of perihelion (deg)      224.06676  =
%  Mean longitude (deg)               238.92881

Orb_Inc_Kepler     = 17.14175;     % degrees, Orbital Inclination
Orb_LA_Kepler      = 110.30347;    % degrees, Orbital Longitude of Ascending Node
Orb_Ecc_Kepler     = 0.24880766;   % Orbital eccentricity
Orb_LP_Kepler      = 224.06676;    % degrees, Orbital Longitude of Perihelion
Orb_Period_Kepler  = 90560.0 * 86400; % seconds, Orbital Period

%  Calculate the semi-major axis with Kepler's third law to be consistent with the NASA data.

r_semi_Kepler  = ( Orb_Period_Kepler^2 * m* g / 4 / pi / pi )^(1/3);

Perihelion_Kepler = r_semi_Kepler * (1-Orb_Ecc_Kepler); % meters, applied as initial condition
Aphelion_Kepler  = r_semi_Kepler * (1+Orb_Ecc_Kepler); % meters, for comparison with solution

% 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
%   For this example it is done for 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) =  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;


%  Define the initial x, y, z,  position coordinates with respect to the center of mass of the
%  governing body, m, in meters.  For this example, it is the closest orbital distance between
%  the Sun and the center of Pluto.

x = rhat(4,1) * Perihelion_Kepler;
y = rhat(4,2) * Perihelion_Kepler;
z = rhat(4,3) * Perihelion_Kepler;

r(1,4) = Perihelion_Kepler;

disp(['Position   x = ' num2str(x,6) ' m,   y = ' num2str(y,6) ' m,  z = ' num2str(z,6) ' m' ])

%   Define vx, vy, vz, Initial velocity in m/s.  If beta_option = 1, these values are treated as
%   bx=vx/c, by=vy/c, bz=vz/c
%
%   For this example it is the maximum orbital velocity (this is at the Perihelion).
%
%   Start with the circular orbital velocity corresponding to the semimajor axis.

%   Calculate beta with TMG equation 7-12

b_circular = sqrt( 1 / ( (cs*cs*r_semi_Kepler / m / g) -1 ) );

%   Calculate c with TMG equation 4-10

c_circular = cr * exp( (2*m*g/cs/cs ) * ( ( r_semi_Kepler-rr ) / (rr*r_semi_Kepler)) );

c_Perihelion  = cr * exp( (2*m*g/cs/cs ) * ( ( Perihelion_Kepler-rr ) / (rr*Perihelion_Kepler)) );

%   Calculate dc with TMG equation 4-14 to reduce numerical errors

g1 = ( 2*m*g/cs/cs )  *  ( ( Perihelion_Kepler - r_semi_Kepler ) / (r_semi_Kepler *
Perihelion_Kepler));

g2 = g1 * g1;
g3 = g2 * g1;
g4 = g3 * g1;
g5 = g4 * g1;
g6 = g5 * g1;
g7 = g6 * g1;
g8 = g7 * g1;
```

```
dc = c_circular*(-g1 -g2/2. -g3/6. -g4/24. -g5/120.0 -g6/720. -g7/5040. -g8/40320);

%   Calculate the initial beta with TMG equations 4-18

b_Perihelion = sqrt( (dc + b_circular*b_circular * c_Perihelion) / c_circular );

beta_option = 1;
bx = b_Perihelion * bhat(4,1);
by = b_Perihelion * bhat(4,2);
bz = b_Perihelion * bhat(4,3);

disp(['Velocity  vx = ' num2str(bx*c_Perihelion,6) ' m/s,  vy = ' num2str(by*c_Perihelion,6) ' m/s,
vz = ' num2str(bz*c_Perihelion,6) ' m/s,  v = ' num2str(b_Perihelion*c_Perihelion,6) ' m/s'])

disp(['Beta    vx/c = ' num2str(bx,6) ', vy/c = ' num2str(by,6) ', vz/c = ' num2str(bz,6) ',   v/c =
' num2str(b_Perihelion,6)])


%   Calculate the Orbit
path_option = 3;

disp(' ')
disp('--- Calculating Pluto orbit ---')

%   Define total rotation, number of points, and arc segments per point for this calculation

atr = 360 * 1.0;
np  = 360;
nac = 1;

disp(['Total rotation angle for calculation = ' num2str(atr,4) ' degrees' ])

disp(['Calculating solution with ' num2str(np) ' points and ' num2str(nac) ' arc segments per
point'])


% Convert total rotation angle to radians for calculation

atr = atr * pi / 180;

TMG_solver

 r_Pluto_path = r;
np_Pluto_path = np;

%   Post-process and display results

disp('--- Pluto Orbital Solution Results -----')

error_t = t - Orb_Period_Kepler;

disp(['Orbital Period = ' num2str( t/86400,6) ' days, delta vs Kepler = ' num2str(error_t,4) '
seconds' ])

disp(['Circumference  = ' num2str( s,12) ' meters' ])

Perihelion_Pluto = min(r(:,4));
  Aphelion_Pluto = max(r(:,4));


error_Perihelion  =  Perihelion_Pluto -  Perihelion_Kepler;
error_Aphelion    =   Aphelion_Pluto -   Aphelion_Kepler;

disp(['Perihelion     = ' num2str(min(r(:,4)),12) ' meters, delta vs Kepler = '
num2str(error_Perihelion,4) ' meters' ])

disp(['Aphelion       = ' num2str(max(r(:,4)),12) ' meters, delta vs Kepler = '
num2str(error_Aphelion,4) ' meters' ])

eccentricity_Pluto = ( max(r(:,4)) - min(r(:,4)) ) / ( max(r(:,4)) + min(r(:,4)) );

error_eccentricity = eccentricity_Pluto - Orb_Ecc_Kepler;

disp(['Eccentricity   = ' num2str( eccentricity_Pluto,8) ', delta vs Kepler = '
num2str(error_eccentricity,4)])

%  Calculate Pluto's orbital precession
```

```
path_option = 1;

disp(' ')
disp('--- Calculating the orbital precession of Pluto caused by the gravitational field of the Sun
---')

atr = 360 * 1.0 * pi / 180;  % Total rotation angle converted to radians for calculation
np  = 32;
nac = 100;

disp(['Calculating solution with ' num2str(np) ' points and ' num2str(nac) ' arc segments per
point'])

TMG_solver

ab = b(np+1,1)*b(1,1) + b(np+1,2)*b(1,2) + b(np+1,3)*b(1,3);

if ab>1
    ab = 1;
end

ab = acos(ab)*180.0/pi;

disp('--- Pluto Orbital Precession Solution Results -----')
disp(['Rotation      = ' num2str((atr*180.0/pi),15) ' degrees' ])
disp(['Rotation time = ' num2str((t/3600/24),15) ' days' ])
disp(['Precession    = ' num2str(ab) ' degrees per orbit' ])
disp(['Precession    = ' num2str((ab*60*60*365.256363004*100*24*3600/t),15) ' arcsecond per
century' ])
%   The following calculation provides a comparison with the General Relativity value.
grp = 24*pi^3*r_semi_Kepler^2/(Orb_Period_Kepler^2*cs^2*(1-Orb_Ecc_Kepler^2));
%   convert to arcseconds per century
grp = (grp*180/pi)*60*60*365.256363004*100*24*3600/t;
disp('--- For comparison only -----')
disp(['--- General Relativity predicted value = ' num2str(grp,6) ' arcsecond per century' ])
```

Example_4_S2_galactic_center.m

```
%   Example_4_S2_galactic_center.m
%
%   MatLab script for calculating the S2 star orbit in the galactic center.
%
%
%   Copyright 2021 Morris G. Anderson
%
%
%   This script will be documented in the following paper to be ePublished on https://vixra.org
%
%
%   Curvature of Motion in a Gravitational Field
%                       by
%               Morris G. Anderson
%
%
%   It will also be uploaded to https://www.mathworks.com/matlabcentral/fileexchange under the same
name.
%
%
%   Information source
%   1)  Time Matter and Gravity (TMG) by Morris G. Anderson provides a derivation of the
%       equations used in this calculation.
%       https://vixra.org/abs/2102.0123
%       https://vixra.org/pdf/2102.0123v1.pdf
%
%   2)  Physical constants
%       https://physics.nist.gov/cuu/pdf/wall_2018.pdf
%
%   3)  Astronomical data
%       https://ssd.jpl.nasa.gov/horizons.cgi
%
%   4)  Relativistic redshift of the star S0-2 orbiting the Galactic center supermassive black hole
%       https://arxiv.org/pdf/1907.10731.pdf
%
%   5)  Detection of the Schwarzschild precession in the orbit of the star S2 near the
%       Galactic centre massive black hole  https://arxiv.org/pdf/2004.07187
%
%
%   The path of motion is calculated with the TMG_path.m MatLab function.  The TMG_solver.m
%   script applies the TMG_path.m function to obtain solutions for:
%
%   path_option = 1; % Calculate the precession of an orbit.  The solution is calculated for one
%                       revolution if the max angle (atr) is input as 360.  The solution is calculated
for
%                       two revolutions if the max angle (atr) is input as 2 * 360 and so forth.
%                       All arc segments have the same rotation angle.
%                       The total rotation angle is part of the orbital precession solution.
%
%                atr = The total rotation angle for calculation, initially an input for the
"TMG_solver",
%                       is overwritten as an output of the "TMG_path" solution.  units = radians
%
%
%   path_option = 2; % Calculate the bending of light and the Shapiro time delay for a given path
length.
%                       Arc segment length is a function of radius from the governing body center.
%
%                st =  The intended maximum path length for the calculation
%
%
%   path_option = 3; % Calculate a simple trajectory or orbit with equal arc angle segments
%
%                atr = The total rotation angle for calculation, initially an input for the
"TMG_solver",
%                       is overwritten as an output of the "TMG_path" solution.  units = radians
%
%
%   path_option = 4; % Calculate a simple trajectory with equal arc length segments
%
%                 st = The intended maximum path length for calculation
%
%
%
%   Process for this example:
%
```

```
%     1)  Define the universal gravitational constant, g, and the standard speed of light, cs.
%         o  Define the mass of the Earth   (for calculating cr).
%         o  Define the radius of the Earth (for calculating cr).
%
%     2)  Calculate cr which is the speed of light away from the Sun at rr, the solar system distance
from
%         the Galactic core.
%
%     3)  Define the mass of the governing body, m, to be equal to the mass of the Galactic core.
%
%     4)  Calculate the input values for position and velocity
%
%     5)  Define path_option = 3 for calculating the orbit.
%
%     6)  Define total rotation, number of points, and number of arc segments per point for this
calculation
%
%     7)  Call TMG_solver.m which in turn calls TMG_path.m to calculate the path.
%
%     8)  Post-process and plot the results.
%
clear
clc

disp(['This example calculates the orbital motion and precession of the star S2 near the galactic
core'])

%  g, universal gravitational constant. Units = m^3/(kg-s),  See Information source 2

g  = 0.66743e-10;

%   The standard speed of light, cs.
%   From Information source 1 (see TMG section 2.4.1  The Speed of light at the standard location)

cs = 299792458.0;

disp(['Newtons universal gravitational constant, G = ' num2str(g,6) ' m^3/(kg-s)'])

disp(['The standard speed of light, cs = ' num2str(cs,10) ' m/s'])

%   For this calculation, cr is the speed of light near the solar system away from the gravitational
influence of the sun.

%   rr is the distance to the galactic core.

rr = 8178.0 *    (648000.0/pi) * 149597870700.0;
     % parsec to meter conversion, distance from https://arxiv.org/pdf/1904.05721.pdf

cr = 299792464.335799;  %  This is the speed of light away from the influence of the Sun
                        %  See Example_5_Sun_light_deflection_Shapiro_time_delay.m for calculation

disp(['Reference speed of light  = ' num2str(cr,12) ' m/s at ' num2str(rr,8) ' meters from the
galactic center'])

%     Define m, This is the mass of the governing body in kilograms
%     For this example it is the mass of the galactic center, see Information source 4

m_Sun =  1.32712e20/g;   % from https://nssdc.gsfc.nasa.gov/planetary/factsheet/sunfact.html

n_solar_mass = 1.0 * 3.975e6;  % from Information source 4 Table 1

m = n_solar_mass * m_Sun;

disp(['Galactic center mass = ' num2str((m),15) ' kg' ])

%  Conditions taken from:
%  https://arxiv.org/pdf/1907.10731.pdf Table 1
%  Relativistic redshift of the star S0-2 orbiting the Galactic center supermassive black hole


%  calculation done in the plane of the sky viewing the galactic center.

Orb_Inc_Kepler     = (180 - 133.82);        % degrees, Orbital Inclination
Orb_LA_Kepler      = (360 - 227.49);        % degrees, Orbital Longitude of Ascending Node
Orb_Ecc_Kepler     = 0.8858;                % Orbital eccentricity
Orb_LP_Kepler      = 66.1 + Orb_LA_Kepler; % degrees, Orbital Longitude of Perihelion
Orb_Period_Kepler  = 16.042 * 365.256363004*24*3600; % seconds, Orbital Period
```

```
dtk = 1.519e5 + 64.48;

%  dtk is needed to match the Kepler period.  Kepler's equation is not as accurate in strong
%  gravitational fields.

%  Calculate the semi-major axis with Kepler's third law.

r_semi_Kepler  = ( (Orb_Period_Kepler + dtk)^2 * m* g / 4 / pi / pi )^(1/3);

Perihelion_Kepler = r_semi_Kepler * (1-Orb_Ecc_Kepler); % meters, applied as initial condition
Aphelion_Kepler  = r_semi_Kepler * (1+Orb_Ecc_Kepler); % meters, for comparison with solution

%  970 * 149597870700 / ((Perihelion_Kepler + Aphelion_Kepler)/2); %  compare with wikipedia value

% 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

%   Define the initial position unit vector
rhat(1,1) =  0.0;
rhat(1,2) =  1.0;
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) =  1.0;
bhat(1,2) =  0.0;
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;


%  Define the initial x, y, z,  position coordinates with respect to the center of mass of the
%  governing body, m, in meters.  For this example, it is the closest orbital distance between
%  the center of mass of the galactic core and S2.

x = rhat(4,1) * Perihelion_Kepler;
y = rhat(4,2) * Perihelion_Kepler;
z = rhat(4,3) * Perihelion_Kepler;

r(1,4) = Perihelion_Kepler;

disp(['Position   x = ' num2str(x,6) ' m,   y = ' num2str(y,6) ' m,  z = ' num2str(z,6) ' m' ])

%   Define vx, vy, vz, Initial velocity in m/s.  If beta_option = 1, these values are treated as
%   bx=vx/c, by=vy/c, bz=vz/c
%
%   For this example it is the maximum orbital velocity (at the Perihelion).
%
%   Start with the circular orbital velocity corresponding to the semi-major axis.

%   Calculate beta with TMG equation 7-12

b_circular = sqrt( 1 / ( (cs*cs*r_semi_Kepler / m / g) -1 ) );

%   Calculate c with TMG equation 4-10

c_circular = cr * exp( (2*m*g/cs/cs ) * ( ( r_semi_Kepler-rr ) / (rr*r_semi_Kepler)) );

c_Perihelion  = cr * exp( (2*m*g/cs/cs ) * ( ( Perihelion_Kepler-rr ) / (rr*Perihelion_Kepler)) );

%   Calculate dc with TMG equation 4-14 to reduce numerical errors

g1 = ( 2*m*g/cs/cs )  *  ( ( Perihelion_Kepler - r_semi_Kepler ) / (r_semi_Kepler *
Perihelion_Kepler));

g2 = g1 * g1;
g3 = g2 * g1;
g4 = g3 * g1;
g5 = g4 * g1;
g6 = g5 * g1;
g7 = g6 * g1;
g8 = g7 * g1;

dc = c_circular*(-g1 -g2/2. -g3/6. -g4/24. -g5/120.0 -g6/720. -g7/5040. -g8/40320);

%   Calculate the initial beta with TMG equation 4-18

b_Perihelion = sqrt( (dc + b_circular*b_circular * c_Perihelion) / c_circular );

beta_option = 1;
bx = b_Perihelion * bhat(4,1);
by = b_Perihelion * bhat(4,2);
bz = b_Perihelion * bhat(4,3);

disp(['Velocity  vx = ' num2str(bx*c_Perihelion,6) ' m/s,  vy = ' num2str(by*c_Perihelion,6) ' m/s,
vz = ' num2str(bz*c_Perihelion,6) ' m/s,  v = ' num2str(b_Perihelion*c_Perihelion,6) ' m/s'])

disp(['Beta    vx/c = ' num2str(bx,6) ', vy/c = ' num2str(by,6) ', vz/c = ' num2str(bz,6) ',   v/c =
' num2str(b_Perihelion,6)])


%   Calculate the Orbit
path_option = 1;
```

```
disp(' ')
disp('--- Calculating S2 orbital precession ---')

%   Define total rotation, number of points, and arc segments per point for this calculation

atr = 360;
np  = 360;
nac = 10;

disp(['Total rotation angle for calculation = ' num2str(atr,4) ' degrees' ])

disp(['Calculating solution with ' num2str(np) ' points and ' num2str(nac) ' arc segments per
point'])


%  Convert total rotation angle to radians for calculation

atr = atr * pi / 180;

TMG_solver

ab = b(np+1,1)*b(1,1) + b(np+1,2)*b(1,2) + b(np+1,3)*b(1,3);

ab = acos(ab)*180.0/pi;

disp('--- S2 Orbital Precession Solution Results -----')
disp(['Rotation      = ' num2str((atr*180.0/pi),15) ' degrees' ])
disp(['Rotation time = ' num2str((t/(3600*24*365.256363004)),15) ' days' ])
disp(['Precession    = ' num2str(ab) ' degrees per orbit' ])
disp(['Precession    = ' num2str(ab*60) ' arc-minutes per orbit' ])


%  Reset orbital starting point to about the Aphelion for comparing the gravitation redshift
%  with observational data contained in Information source 4


np = 1 + np/2;

bx = b(np,1)*b(np,4);
by = b(np,2)*b(np,4);
bz = b(np,3)*b(np,4);

x = r(np,5);
y = r(np,6);
z = r(np,7);

atr = 360;
np  = 360;
nac = 10;

disp(['Recalculating results for a simple 360-degree rotation starting at the Aphelion for plotting
and comparing the gravitational redshift with data' ])

%  Convert total rotation angle to radians for calculation

atr = atr * pi / 180;
path_option = 3;

TMG_solver

%   Post-process and display results

r_S2_path = r;
np_S2_path = np;

disp('--- S2 Orbital Solution Results -----')

error_t = t - Orb_Period_Kepler;

disp(['Orbital Period = ' num2str( t/(365.256363004*86400),6) ' years, delta vs Kepler = '
num2str(error_t,4) ' seconds' ])

disp(['Circumference  = ' num2str( s,12) ' meters' ])

Perihelion_S2 = min(r(:,4));
  Aphelion_S2 = max(r(:,4));
```

```
error_Perihelion  =  Perihelion_S2 -  Perihelion_Kepler;
error_Aphelion    =   Aphelion_S2 -   Aphelion_Kepler;

disp(['Perihelion     = ' num2str(min(r(:,4)),12) ' meters, delta vs Kepler = '
num2str(error_Perihelion,4) ' meters' ])

disp(['Aphelion       = ' num2str(max(r(:,4)),12) ' meters, delta vs Kepler = '
num2str(error_Aphelion,4) ' meters' ])

eccentricity_S2 = ( max(r(:,4)) - min(r(:,4)) ) / ( max(r(:,4)) + min(r(:,4)) );

error_eccentricity = eccentricity_S2 - Orb_Ecc_Kepler;

disp(['Eccentricity   = ' num2str( eccentricity_S2,8) ', delta vs Kepler = '
num2str(error_eccentricity,4)])



r  =  r_S2_path;
np = np_S2_path;
[r_aboves, r_belows] = sort_for_plotting_orbit(np, r);

% swap above and below for viewing from earth

S2_r_above = r_aboves;
S2_r_below = r_belows;


plot(S2_r_above(:,5),S2_r_above(:,6),'LineWidth',2,'Color','b')

hold('on')

plot(S2_r_below(:,5),S2_r_below(:,6),'LineWidth',2,'Color','b', 'LineStyle', ':')


row =find(r_S2_path(:,4)==Perihelion_S2);
plot(r_S2_path(row,5),r_S2_path(row,6),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerEdgeCol
or','b','MarkerFaceColor','b')

row =find(r_S2_path(:,4)==Aphelion_S2);
plot(r_S2_path(row,5),r_S2_path(row,6),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerEdgeCol
or','b')


plot_limit = abs(1.1 * max(r_S2_path(:,4)) * sin(Orb_Inc_Kepler));
xlim([-1.434*plot_limit,1.434*plot_limit])
ylim([-plot_limit,plot_limit])
xlabel('Plane of the sky x axis, meters')
ylabel('Plane of the sky y axis, meters')
pause(20)
hold('off');


plot(S2_r_above(:,7),S2_r_above(:,5),'LineWidth',2,'Color','b')

hold('on')

plot(S2_r_below(:,7),S2_r_below(:,5),'LineWidth',2,'Color','b', 'LineStyle', ':')

row =find(r_S2_path(:,4)==Perihelion_S2);
plot(r_S2_path(row,7),r_S2_path(row,5),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerEdgeCol
or','b','MarkerFaceColor','b')

row =find(r_S2_path(:,4)==Aphelion_S2);
plot(r_S2_path(row,7),r_S2_path(row,5),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerEdgeCol
or','b')

%plot_limit = 1.1 * max(r_S2_path(:,4));
xlim([-1.434*plot_limit,1.434*plot_limit])
ylim([-plot_limit,plot_limit])
xlabel('Plane of the sky z axis, meters')
ylabel('Plane of the sky x axis, meters')
hold('off');
pause(20)


plot(S2_r_above(:,7),S2_r_above(:,6),'LineWidth',2,'Color','b')
```

```
hold('on')

plot(S2_r_below(:,7),S2_r_below(:,6),'LineWidth',2,'Color','b', 'LineStyle', ':')


row =find(r_S2_path(:,4)==Perihelion_S2);
plot(r_S2_path(row,7),r_S2_path(row,6),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerEdgeCol
or','b','MarkerFaceColor','b')

row =find(r_S2_path(:,4)==Aphelion_S2);
plot(r_S2_path(row,7),r_S2_path(row,6),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerEdgeCol
or','b')

xlim([-1.434*plot_limit,1.434*plot_limit])
ylim([-plot_limit,plot_limit])
xlabel('Plane of the sky z axis, meters')
ylabel('Plane of the sky y axis, meters')
hold('off');
pause(20)


t_shift = 2018.3763 - 16.042/2;

%   Calculate the apparent velocity shift caused by the gravitational influence on
%   the natural frequency of matter.  The Doppler equation for this is dv = c * df/f.
%   For this example df/f is calculated with equation TMG 5-29.

v_SS_g_rot = 230000.0;  % This is the solar system rotation velocity about the galactic center
                        % "The Solar System is traveling at an average speed of 230 km/s ... around
                        % the galactic center"   https://en.wikipedia.org/wiki/Galactic_year

v_SS_g_rv  =  20000.0;  % This is the solar system radial velocity from the galactic center
                        % See Information source 4 bottom of page 6

for n = 1:np+1
    zg(n,1) =  t_shift + r(n,9)/(365.256363004*24*3600);                                  % time
    zg(n,2) =  c(n) * (1 - sqrt((c(n)/cr) * (1 - b(n,4)^2 ))/(1- (v_SS_g_rot/cr)^2));  % df/f
    zg(n,3) =  zg(n,2) - c(n)* b(n,3)*b(n,4);                                             % dvr
    zg(n,4) =  v_SS_g_rv - c(n)*b(n,3)*b(n,4);                                            % Radial
velocity
    zg(n,5) =  c(n)*b(n,4);                                                               % Orbital
velocity
end

plot(zg(:,1), zg(:,2),'LineWidth',2,'Color','b')
xlim([2017,2019])
ylim([0,200000])
xlabel('Epoch, year')
ylabel('RV deviation caused by frequency shift, m/s')

pause(20)

plot(zg(:,1), zg(:,3),'LineWidth',2,'Color','b')

hold('on')

%   Spectroscopy data, see Information source 4 Figure 1

xl(1,1)=2017 + 208/365; % 2017-07-27
xl(2,1)=2018 +  76/365; % 2018-03-17
xl(3,1)=2018 + 114/365; % 2018-04-24
xl(4,1)=2018 + 133/365; % 2018-05-13
xl(5,1)=2018 + 143/365; % 2018-05-23
xl(6,1)=2018 + 212/365; % 2018-07-31

xl(1,2) = 2014000.0;
xl(2,2) = 3798000.0;
xl(3,2) = 3966000.0;
xl(4,2) = 3000000.0;
xl(5,2) = 2062000.0;
xl(6,2) = -1626000.0;
plot(xl(:,1), xl(:,2),'LineStyle','none','Marker','o','MarkerSize',10,'MarkerEdgeColor','b')

xlim([2017,2019])
ylim([-2500000,4500000])
xlabel('Epoch, year')
ylabel('RV, m/s')
```

```
plot(zg(:,1), zg(:,4),'LineWidth',2,'LineStyle', ':','Color','b')
legend('Calculated Apparent RV', 'Spectroscopy data', 'Actual RV')

hold('off')

pause(20)

plot(zg(:,1), zg(:,5),'LineWidth',2,'LineStyle', '-','Color','b')

xlim([2010,2026])
ylim([0.0 ,8000000])
xlabel('Epoch, year')
ylabel('Orbital velocity, m/s')
```

Example_5_strong_gravity_field_precession.m

```
%   Example_5_strong_gravity_field_precession.m
%
%   MatLab script for calculating orbital precession in a strong gravitational field
%
%
%   Copyright 2021 Morris G. Anderson
%
%
%   This script will be documented in the following paper to be ePublished on https://vixra.org
%
%
%   Curvature of Motion in a Gravitational Field
%                       by
%               Morris G. Anderson
%
%
%   It will also be uploaded to https://www.mathworks.com/matlabcentral/fileexchange under the same
name.
%
%
%   Information source
%   1)  Time Matter and Gravity (TMG) by Morris G. Anderson provides a derivation of the
%       equations used in this calculation.
%       https://vixra.org/abs/2102.0123
%       https://vixra.org/pdf/2102.0123v1.pdf
%
%   2)  Physical constants
%       https://physics.nist.gov/cuu/pdf/wall_2018.pdf
%
%   3)  Astronomical data
%       https://ssd.jpl.nasa.gov/horizons.cgi
%
%   4)  Satellite TLE data obtained from https://celestrak.com
%
%   5)  Fractional frequency shift data obtained and from
%       Relativity in the Global Positioning System by Neil Ashby
%       Published on 28 January 2003
%       https://link.springer.com/article/10.12942/lrr-2003-1
%
%
%   The path of motion is calculated with the TMG_path.m MatLab function.  The TMG_solver.m
%   script applies the TMG_path.m function to obtain solutions for:
%
%   path_option = 1; % Calculate the precession of an orbit.  The solution is calculated for one
%                       revolution if the max angle (atr) is input as 360.  The solution is calculated
for
%                       two revolutions if the max angle (atr) is input as 2 * 360 and so forth.
%                       All arc segments have the same rotation angle.
%                       The total rotation angle is part of the orbital precession solution.
%
%                atr = The total rotation angle for calculation, initially an input for the
"TMG_solver",
%                       is overwritten as an output of the "TMG_path" solution.  units = radians
%
%
%   path_option = 2; % Calculate the bending of light and the Shapiro time delay for a given path
length.
%                       Arc segment length is a function of radius from the governing body center.
%
%                st =  The intended maximum path length for the calculation
%
%
%   path_option = 3; % Calculate a simple trajectory or orbit with equal arc angle segments
%
%                atr = The total rotation angle for calculation, initially an input for the
"TMG_solver",
%                       is overwritten as an output of the "TMG_path" solution.  units = radians
%
%
%   path_option = 4; % Calculate a simple trajectory with equal arc length segments
%
%                 st = The intended maximum path length for calculation
%
%
%
%   Process for this example:
```

```
%
%     1)  Define the universal gravitational constant, g, and the standard speed of light, cs.
%
%     2)  Define the mass of the governing body.
%
%     3)  Define phi at the orbital periapsis.
%
%     4)  Calculate the orbital periapsis from phi
%
%     5)  Define the reference speed of light, cr, and the reference radius, rr.  This is used for
%         calculating the speed of light as a function of position from the governing body.
%
%     6)  Calculate beta for a circular orbit at the orbital periapsis.
%
%     7)  Calculate beta for escape at the orbital periapsis
%
%     8)  Define the fraction_escape - should be greater than 0 and less than 1
%
%     9)  Define initial position and beta for orbit
%
%    10)  Define path_option = 1 for precession or 3 for calculating a simple trajectory.
%
%    11)  Define total rotation, number of points and arc segments per point for this calculation
%
%    12)  Call TMG_solver.m which in turn calls TMG_path.m to calculate the path for this case.
%
%    13)  Post-process and plot the results.

clear
clc

%  g, universal gravitational constant. Units = m^3/(kg s)

g  = 0.66743e-10;

%   By definition cs.  This is on the earth at the standard location (see TMG page 8 Definition 10).

cs = 299792458.0;

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

%   For this example let us assume a region of space such that the speed of light equals cs away from
the influence of the governing body.

m_Sun = 1.32712e20/g; % from https://nssdc.gsfc.nasa.gov/planetary/factsheet/sunfact.html

n_solar_mass = 1;

m = n_solar_mass * m_Sun;

disp(['Governing body = ' num2str(n_solar_mass,2) ' solar mass = ' num2str(m) ' kg' ])

%   define phi of the governing body at the orbit periapsis

phi = -0.3;  % -0.1, -0.3

redshift= (1/sqrt(exp(phi))) - 1; %This is the gravitational redshift at the orbit periapsis

%   calculate the orbital periapsis from phi, see TMG equation 4-11 and 4-12

r_periapsis = -2*g*m/cs/cs/phi;

%   For this example set c far away from m equal to cs, set rr equal to the periapsis.
%   Calculate cr with TMG eq 4-11 as:

rr = r_periapsis;
cr = cs * exp(-2.0 * m * g / cs / cs / rr);

disp(['Reference speed of light  = ' num2str(cr,12) ' m/s at ' num2str(rr,8) ' meters from the
governing body'])

%   calculate beta for a circular orbit at rr with TMG equation 7-12

b_circular = sqrt( 1 / ( (cs*cs*rr / m / g) -1 ) );

%   Calculate the escape beta for rr with TMG eq 4-18.  For this calculation, the variables c1 = cs,
b1 = 0, and b_escape = b in TMG eq 4-18).
```

```
b_escape = sqrt( (cs - cr) / cs);

%     Define vx, vy, vz, Initial velocity in m/s.  If beta_option = 1,
%     then these values are treated as vx/c, vy/c, vz,c



% >>>>>>   Select the fraction escape, should be greater than 0 and less than 1



fraction_escape_beta = 0.95;  % 0.5, 0.95,



b(1,4) = (b_circular + fraction_escape_beta * ( b_escape - b_circular ));

%    Define the initial x, y, z,  location coordinates with respect to
%    the center of mass of the governing body, m, in meters.

x = 0;
y = r_periapsis;
z = 0;

%     Define vx, vy, vz, Initial velocity in m/s.  If beta_option = 1,
%     then these values are treated as vx/c, vy/c, vz,c

beta_option = 1;
bx = - b(1,4);
by = 0;
bz = 0;

disp(' ')
disp('--- At the orbital periapsis --- ')
disp(['phi = ' num2str(phi) ', gravitational redshift = ' num2str(redshift,6)])
disp(['v/c equal to ' num2str(b_circular,6) ' results in a circular orbit.  v/c = '
num2str(b_escape,6) ' equals escape velocity.'])
disp(['v/c for this orbit = ' num2str(b(1,4),6),' at the periapsis'])
disp(['fraction between circular orbit and escape velocity = ' num2str(fraction_escape_beta,2)])

disp(' ')
disp('--- Initial conditions at the orbital periapsis ---')
disp(['Position   x = ' num2str(x,6) ' m,   y = ' num2str(y,6) ' m,  z = ' num2str(z,6) ' m' ])
disp(['Beta, vx/c = ' num2str(bx,6) ', vy/c = ' num2str(by,6) ', vz/c = ' num2str(bz,6) ])



% >>>>>>   Select the path option



%   path_option = 1; % Calculate the precession of an orbit
path_option = 3; % Calculate a simple trajectory or orbit with equal arc angle segments - for
multiple orbits


disp('--- Calculating the orbital precession caused by the gravitational field of the governing body
---')



% >>>>>>   Define the total rotation, example, 360 degrees for one orbit



%  atr = (1.0 * 360) * pi / 180;  % Total rotation angle converted to radians for calculation - for
precession
atr = (3.5 * 360) * pi / 180;  % Total rotation angle converted to radians for calculation - for
multiple orbits



np  = 360; % number of points in addition to the starting position to be stored in memory
nac = 10; % number of arc segments integrated to calculate each point that is stored in memory

disp(['Calculating solution with ' num2str(np) ' points and ' num2str(nac) ' arc segments per
point'])
```

```
TMG_solver

ab = b(np+1,1)*b(1,1) + b(np+1,2)*b(1,2) + b(np+1,3)*b(1,3);

ab = acos(ab)*180.0/pi;

disp('--- Orbital Solution Results -----')
disp(['Rotation      = ' num2str((atr*180.0/pi),15) ' degrees' ])
disp(['Rotation time = ' num2str((t/3600/24),15) ' days' ])

rp = min(r(:,4));
ra = max(r(:,4));


if( path_option == 1)
    disp(['Precession    = ' num2str(atr*180.0/pi - 360) ' degrees per orbit' ])

%  The following calculation provides a comparison with the General Relativity value.
%  Calculate the semi-major axis with Kepler's third law.



    Orb_Ecc_Kepler = 1 - 2/(1 + ra/rp);

    r_semi_Kepler  = ( t^2 * m* g / 4 / pi / pi )^(1/3);

    grp = 24*pi^3*r_semi_Kepler^2/(t^2*cs^2*(1-Orb_Ecc_Kepler^2));

    grp = (grp*180/pi); %   convert to degrees per orit

    disp('--- For comparison only -----')
    disp(['--- General Relativity predicted value = ' num2str(grp,6) ' degrees per orbit' ])
end


plot(r(:,5),r(:,6),'LineWidth',2,'Color','b')

xlabel('x axis, meters')
ylabel('y axis, meters')

xlim([-1.44*ra, 1.44*ra])
ylim([-ra,ra])
```

Example_6_Sunlight_deflection_Shapiro_time_delay.m

```
%   Example_6_Sunlight_deflection_Shapiro_time_delay.m
%
%   Matlab script for calculating the deflection of starlight and the Shapiro time delay
%
%   Copyright 2021 Morris G. Anderson
%
%
%   This script will be documented in the following paper to be ePublished on https://vixra.org
%
%
%   Curvature of Motion in a Gravitational Field
%                       by
%               Morris G. Anderson
%
%
%   It will also be uploaded to https://www.mathworks.com/matlabcentral/fileexchange under the same
name.
%
%
%   Information source
%   1)  Time Matter and Gravity (TMG) by Morris G. Anderson provides a derivation of the
%       equations used in this calculation.
%       https://vixra.org/abs/2102.0123
%       https://vixra.org/pdf/2102.0123v1.pdf
%
%   2)  Physical constants
%       https://physics.nist.gov/cuu/pdf/wall_2018.pdf
%
%   3)  Astronomical data
%       https://ssd.jpl.nasa.gov/horizons.cgi
%       https://nssdc.gsfc.nasa.gov/planetary/factsheet
%
%
%
%   The path of motion is calculated with the TMG_path.m MatLab function.  The TMG_solver.m
%   script applies the TMG_path.m function to obtain solutions for:
%
%   path_option = 1; % Calculate the precession of an orbit.  The solution is calculated for one
%                       revolution if the max angle (atr) is input as 360.  The solution is calculated
for
%                       two revolutions if the max angle (atr) is input as 2 * 360 and so forth.
%                       All arc segments have the same rotation angle.
%                       The total rotation angle is part of the orbital precession solution.
%
%                   atr = The total rotation angle for calculation, initially an input for the
"TMG_solver",
%                       is overwritten as an output of the "TMG_path" solution.  units = radians
%
%
%   path_option = 2; % Calculate the bending of light and the Shapiro time delay for a given path
length.
%                       Arc segment length is a function of radius from the governing body center.
%
%                   st =  The intended maximum path length for the calculation
%
%
%   path_option = 3; % Calculate a simple trajectory or orbit with equal arc angle segments
%
%                   atr = The total rotation angle for calculation, initially an input for the
"TMG_solver",
%                       is overwritten as an output of the "TMG_path" solution.  units = radians
%
%
%   path_option = 4; % Calculate a simple trajectory with equal arc length segments
%
%                    st = The intended maximum path length for calculation
%
%
%
%   Process for this example:
%
%     1)  Define the path length, st, for this calculation.
%
%     2)  Define the universal gravitational constant, g, and the standard speed of light, cs.
%         o  Define the mass of the Earth   (for calculating cr).
%         o  Define the radius of the Earth (for calculating cr).
```

```
%
%     3)  Calculate the reference speed of light, cr, at the selected reference radius, rr, from
%         the governing body (the Sun).  This is used for calculating the speed of light as a
%         function of position from the Sun.
%
%     4)  Define the mass of the governing body, m, to be equal to the mass of the Sun.
%
%     5)  Define the initial position and velocity for the calculation.
%
%     6)  Define the number of points and arc segments per point for this calculation.
%
%     7)  Define path_option = 2 to calculate the bending of light and the Shapiro time delay.
%
%     8)  Call TMG_solver.m which in turn calls TMG_path.m to calculate the path for this case.
%
%     9)  Post-process and plot the results.
%


clear
clc

disp('Calculating the deflection of light passing near the Sun')

au = 149597870700.0; % see https://cneos.jpl.nasa.gov/glossary/au.html

%   The distance between Earth and the planet is calculated as the sum of the semimajor axis of both

%st = (0.38709893 +  1.00000011) * au; % Mercury
%disp(['Total path length for Earth to Mercury superior conjunction= ' num2str(st,6) ' meters' ])

st = (0.72333199 +  1.00000011) * au; % Venus
disp(['Total path length for Earth to Venus superior conjunction= ' num2str(st,6) ' meters' ])

%st = (1.52366231 +  1.00000011) * au; % Mars
%disp(['Total path length for Earth to Mars superior conjunction= ' num2str(st,6) ' meters' ])

%st = (39.48168677 +  1.00000011) * au; % Outside the solar system
%disp(['Total path length for Earth to Pluto superior conjunction= ' num2str(st,6) ' meters' ])


disp(['Total path length for calculation = ' num2str(st,6) ' meters' ])

disp('Constants:')

%  g, universal gravitational constant. Units = m^3/(kg-s)

g  = 0.66742e-10;

%   The standard speed of light, cs.  This is on the earth at re (see TMG Definition 10).

cs = 299792458.0;


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


me = 5.9723e24;
re = 6.36745e6;

disp(['Earth mass = ' num2str(me,6) ' kg,  Earth radius = ' num2str(re,8) ' m'])

%   For this calculation, cr is the speed of light away from the earth at the Earth's average
%   distance from the Sun.  It is calculated with TMG eq 4-11.

cr = cs * exp( 2.0 * me * g / cs / cs / re );

rr = 0.1496e12;

disp(['Reference speed of light  = ' num2str(cr,12) ' m/s at ' num2str(rr,8) ' meters from the Sun
which is Earths average orbital radius'])


%     Define m, This is the mass of the governing body in kilograms
%     For this example it is the mass of the Sun, see Reference 3

m_Sun =  1.32712e20/g; % from https://nssdc.gsfc.nasa.gov/planetary/factsheet/sunfact.html
```

```
m = m_Sun;
r_Sun = 695700000.0;

disp(['Mass of the Sun   = ' num2str(m,11) ' kg' ])
disp(['Radius of the Sun = ' num2str(r_Sun,11) ' m' ])

%   Define the initial x, y, z, coordinates relative to the center of mass of the governing body, m,
in meters.

x = -au;
y = r_Sun + 3050.0;
z = 0;

disp(['Initial Position x = ' num2str(x,15) ' y = ' num2str(y,15) ' z = ' num2str(z,15) ' meters' ])


%   Define vx, vy, vz, Initial velocity in m/s.  If beta_option = 1, then these values are treated as
bx=vx/c, by=vy/c, bz=vz,c

beta_option = 1;
bx = 1.0;
by = 0;
bz = 0;

disp(['Initial velocity (b = v/c),  bx = ' num2str(bx,4) ' by = ' num2str(by,4) ' bz = '
num2str(bz,4) ])


%   Define:
%   np  = number of points in addition to the starting position to be stored in memory for the
solution which can then be plotted.
%   nap = number of arc segments integrated to calculate each point that is stored in memory

np  = 200;
nac = 10;

path_option = 2;  % Calculate the bending of light and the Shapiro time delay for a given path
length.

disp(['Calculating solution with = ' num2str(np*nac) ' steps' ])

%     Calculate the path of motion with the TMG solver.


TMG_solver


%     Display the results


disp(['Final Position x = ' num2str(r((np+1),5),15) ' y = ' num2str(r((np+1),6),15) ' z = '
num2str(r((np+1),7),15) ' meters' ])

disp(['Path   time = ' num2str(t/3600/24) ' days = ' num2str(t,15) ' seconds' ])
disp(['Path length = ' num2str(s,15) ' meters' ])

EucD = sqrt( (r((np+1),5) - r(1,5))^2 + (r((np+1),6) - r(1,6))^2 +(r((np+1),7) - r(1,7))^2);

disp(['Euclidian distance between the beginning and end = ' num2str(EucD,15) ' meters'])

disp(['(Path length - Euclidian distance) = ' num2str((s - EucD),15) ' meters'])

delay_cr = 2.0 * ( t - (EucD/cr) );

disp(['Shapiro time delay = ' num2str(delay_cr,15) ' seconds with respect to electromagnetic wave
propagation at the earths orbital radius from the Sun' ])

%   For this calculation, c_inf is the speed of light away from the influence of the Sun.  It is
calculated with TMG eq 4-11.

c_inf = cr * exp( 2.0 * m * g / cs / cs / rr );

delay_inf = 2.0 * ( t - (EucD/c_inf) );

disp(['Speed of light away from the influence of the Sun = ' num2str(c_inf,15) ' m/s' ])

disp(['Shapiro time delay = ' num2str(delay_inf,15) ' seconds with respect electromagnetic wave
propagation away from the influence of the Sun' ])
```

```
%  Calculate the Angle between the beginning and ending directions

    ab = b((np+1),1)*b(1,1) + b((np+1),2)*b(1,2) + b((np+1),3)*b(1,3);

    ab = acos(ab)*180/pi;

    disp(['Angle between the beginning and ending directions = ' num2str(ab) ' degrees = '
num2str(ab*60*60) ' arcseconds'])

    disp(['Trajectory radius: Min = ' num2str(min(r(:,4)),15) ', Max = ' num2str(max(r(:,4)),15) '
meters' ])


%window_ar = 7.05/4.90; %x/y old ipad full
%window_ar = 3.00/4.75; %x/y old ipad half
window_ar = 31.45/45.5; %x/y new ipad full


% plot surface of Sun


as= linspace(0,2*pi,361);
xs = r_Sun * cos(as);
ys = r_Sun * sin(as);

plot(xs,ys,'Color',[1, 0.65, 0.35],'LineWidth',3)

hold('on')

plot(r(:,5),r(:,6),'Color',[1, 0.65,
0.35],'LineWidth',1,'LineStyle','-','Marker','o','MarkerSize',2,'MarkerEdgeColor',[1, 0.65,
0.35],'MarkerFaceColor','none')
plot(r(1,5),r(1,6),'Color','b','LineWidth',1,'LineStyle','none','Marker','o','MarkerSize',2,'MarkerEd
geColor','b','MarkerFaceColor','b')
plot(r(np+1,5),r(np+1,6),'Color','r','LineWidth',1,'LineStyle','none','Marker','o','MarkerSize',2,'Ma
rkerEdgeColor','r','MarkerFaceColor','r')


hold('off')

xlabel('x, meters')
ylabel('y, meters')

plot_limit = 1.05*(r(np+1,5)-r(1,5));

xlim([r(1,5)*1.025,r(1,5)*1.025+plot_limit])
ylim([-plot_limit*window_ar/2,plot_limit*window_ar/2])

pause(20);

xlim([-2*r_Sun, 2*r_Sun])
ylim([r_Sun-10000, r_Sun+4000])

pause(20);



%  Plot reduction in c as a function of time along the path

[M,I] = min(r(:,4));   %  This finds the point of closest approach

plot(r(:,9)-r(I,9),((c(:)-c(I))/(cr-c(I))),'Color',[1, 0.65, 0.35],'LineWidth',3,'LineStyle',':')

hold('on');

%  Plot time delay as a function of path time

for nn = 1:np;
    EucD = sqrt( (r((nn+1),5) - r(1,5))^2 + (r((nn+1),6) - r(1,6))^2 +(r((nn+1),7) - r(1,7))^2);
    tdn((nn+1),1)= ( r((nn+1),9) - (EucD/cr));
end

plot(r(:,9)-r(I,9), 2*tdn(:,1)/delay_cr,'Color','b','LineWidth',3,'LineStyle','-');  %the factor 2
simulates a time delay for)a reflected signal

%   Plot deflection angle as a function of time along the path
```

```
plot(r(:,9)-r(I,9), r(:,8)/max(r(:,8)),'Color','r','LineWidth',3,'LineStyle','-.')
xlabel('Time from closest approach, seconds')
ylabel('Fraction of maximum')
xlim([-30,   30])
ylim([0,    1.0])

legend(strcat('(c - c min) / (cr - c min): (cr - c min), m/s = ', num2str((cr - c(I)),4)),
strcat('Round trip Time delay, s = ', num2str((delay_cr),4)), strcat('Max deflection angle,
arcseconds = ', num2str((max(r(:,8))*60*60),4)))

hold('off');

pause(20);

plot(r(:,9)-r(I,9), r(:,4)/r_Sun)
xlabel('Time from closest approach, seconds')
ylabel('Radial distance from center / solar radius')
xlim([-30,   30])
ylim([1, 15])

pause(20);

%  The following results were calculated for the Earth to Venus superior conjunction to illustrate
the solution convergence

sac(1) = 20.0; % Total solution arc segments
 td(1) = 2.12287669455691e-4;  % Shapiro time delay in seconds
 da(1) = 1.429; % Bending or deflection in arcseconds

sac(2) = 40.0; % Total solution arc segments
 td(2) = 2.01328970433678e-4;  % Shapiro time delay in seconds
 da(2) = 1.664; % Bending or deflection in arcseconds

sac(3) = 80.0; % Total solution arc segments
 td(3) = 1.99264683715228e-4;  % Shapiro time delay in seconds
 da(3) = 1.729; % Bending or deflection in arcseconds

sac(4) = 160.0; % Total solution arc segments
 td(4) = 1.98781929157121e-4;  % Shapiro time delay in seconds
 da(4) = 1.746; % Bending or deflection in arcseconds

sac(5) = 320.0; % Total solution arc segments
 td(5) = 1.98663261699039e-4;  % Shapiro time delay in seconds
 da(5) = 1.750; % Bending or deflection in arcseconds

sac(6) = 640.0; % Total solution arc segments
 td(6) = 1.98633729951325e-4;  % Shapiro time delay in seconds
 da(6) = 1.751; % Bending or deflection in arcseconds

sac(7) = 1280.0; % Total solution arc segments
 td(7) = 1.98626351902931e-4;  % Shapiro time delay in seconds
 da(7) = 1.751; % Bending or deflection in arcseconds

sac(8) = 2560.0; % Total solution arc segments
 td(8) = 1.98624512449896e-4;  % Shapiro time delay in seconds
 da(8) = 1.751; % Bending or deflection in arcseconds

plot(log10(sac),td/
td(8),'Color','b','LineWidth',3,'Marker','o','MarkerSize',10,'MarkerEdgeColor','b')

hold('on');

plot(log10(sac),da/
da(8),'Color','r','LineWidth',3,'Marker','^','MarkerSize',10,'MarkerEdgeColor','r')

legend( strcat('Round trip Time delay, s = ', num2str((delay_cr),4)), strcat('Max deflection angle,
arcseconds = ', num2str((max(r(:,8))*60*60),4)))
xlabel('log10(Number of solution Arc Segments)')
ylabel('Fraction of converged value')
xlim([1, 4])
ylim([0.6, 1.4])

hold('off');
```

Example_7_Neutron_star_light_deflection.m

```
%   Example_7_Neutron_star_light_deflection.m
%
%   Matlab script for calculating the deflection of starlight and cosmic rays passing near
%   a Neutron Star.
%
%   This example simulates Neutron Star RX J0720.4-3125, see https://arxiv.org/abs/1911.08107
%
%   Copyright 2021 Morris G. Anderson
%
%
%   This script will be documented in the following paper to be ePublished on https://vixra.org
%
%
%   Curvature of Motion in a Gravitational Field
%                       by
%               Morris G. Anderson
%
%
%   It will also be uploaded to https://www.mathworks.com/matlabcentral/fileexchange under the same
name.
%
%
%   Information source
%   1)  Time Matter and Gravity (TMG) by Morris G. Anderson provides a derivation of the
%       equations used in this calculation.
%       https://vixra.org/abs/2102.0123
%       https://vixra.org/pdf/2102.0123v1.pdf
%
%   2)  Physical constants
%       https://physics.nist.gov/cuu/pdf/wall_2018.pdf
%
%   3)  Astronomical data
%       https://ssd.jpl.nasa.gov/horizons.cgi
%       https://nssdc.gsfc.nasa.gov/planetary/factsheet
%
%
%
%   The path of motion is calculated with the TMG_path.m MatLab function.  The TMG_solver.m
%   script applies the TMG_path.m function to obtain solutions for:
%
%   path_option = 1; % Calculate the precession of an orbit.  The solution is calculated for one
%                       revolution if the max angle (atr) is input as 360.  The solution is calculated
for
%                       two revolutions if the max angle (atr) is input as 2 * 360 and so forth.
%                       All arc segments have the same rotation angle.
%                       The total rotation angle is part of the orbital precession solution.
%
%               atr = The total rotation angle for calculation, initially an input for the
"TMG_solver",
%                       is overwritten as an output of the "TMG_path" solution.  units = radians
%
%
%   path_option = 2; % Calculate the bending of light and the Shapiro time delay for a given path
length.
%                       Arc segment length is a function of radius from the governing body center.
%
%               st =  The intended maximum path length for the calculation
%
%
%   path_option = 3; % Calculate a simple trajectory or orbit with equal arc angle segments
%
%               atr = The total rotation angle for calculation, initially an input for the
"TMG_solver",
%                       is overwritten as an output of the "TMG_path" solution.  units = radians
%
%
%   path_option = 4; % Calculate a simple trajectory with equal arc length segments
%
%                st = The intended maximum path length for calculation
%
%
%
%   Process for this example:
%
%   see embedded comments
```

```
clear
clc

path_option = 4; % Calculate a simple trajectory with equal arc length segments

disp('Calculate the deflection of light passing near a Neutron star')

disp('Constants:')

%  g, universal gravitational constant. Units = m^3/(kg-s)

g  = 0.66742e-10;

%   The standard speed of light, cs.  This is on the earth at re (see TMG Definition 10).

cs = 299792458.0;

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


me = 5.9723e24;
re = 6.36745e6;

disp(['Earth mass = ' num2str(me,6) ' kg,  Earth radius = ' num2str(re,8) ' m'])

%   For this calculation, cr is the speed of light away from the earth at the earths average
%   distance from the Sun.  It is calculated with TMG eq 4-11.
%   cr and rr are assumed to be the same for the neutron star.

cr = cs * exp( 2.0 * me * g / cs / cs / re );

rr = 0.1496e12;

disp(['Reference speed of light  = ' num2str(cr,12) ' m/s at ' num2str(rr,8) ' meters from the
neutron star'])


%     Define m, This is the mass of the governing body in kilograms
%     For this example it is the mass of the neutron star.

m_Sun = 1.32712e20/g; % from https://nssdc.gsfc.nasa.gov/planetary/factsheet/sunfact.html

m = 1.23 * m_Sun;

disp(['Mass of governing body = ' num2str((m/m_Sun),11) ' solar masses' ])

%   Calculate the radius of Neutron Star RX J0720.4-3125 from its gravitational redshift.
%   see https://arxiv.org/pdf/1911.08107.pdf

zg = 0.205;
phi = log(sqrt(1/(zg+1)));
r_Neutron_star = -2*g*m/cs/cs/phi;
%phi = -2*g*m/cs/cs/r_Neutron_star

%  Define total path length for this calculation.

disp(['Neutron star radius  = ' num2str(r_Neutron_star,9) ' meters' ])
disp(['Neutron star density = ' num2str(m/((4/3)*pi*(r_Neutron_star^3)),6) ' kg/m^3' ])

st =  26 * r_Neutron_star;

disp(['Total path length for calculation = ' num2str(st,6) ' meters' ])


%   Define vx, vy, vz, Initial velocity in m/s.  If beta_option = 1, then these values are treated as
bx=vx/c, by=vy/c, bz=vz,c

beta_option = 1;
bx = -1.0;
by = 0;
bz = 0;

disp(['Initial velocity (b = v/c),  bx = ' num2str(bx,4) ' by = ' num2str(by,4) ' bz = '
num2str(bz,4) ])


%   Define:
%   np  = number of points in addition to the starting position to be stored in memory
```

```
%   nac = number of arc segments integrated to calculate each point that is stored in memory

np = 200;
nac = 10;


disp(['Calculate the path of motion for light with the TMG solver from multiple starting points.' ])

%   Define the initial x, y, z, coordinates with respect to the center of mass of the governing body,
m, in meters.

yscalar1 = 1.2;
yscalar2 = 1.4;

x = 13.0 * r_Neutron_star;
y =  (yscalar1 * yscalar2^0) * r_Neutron_star;
z = 0;

disp(['Initial Position x = ' num2str(x,15) ' y = ' num2str(y,15) ' z = ' num2str(z,15) ' meters' ])

phi = -2*g*m/cs/cs/y;
zg = (1/sqrt(exp(phi))) -1;  % This is the gravitational redshift.


disp(['Calculate solution for initial radial offset of phi = ' num2str(phi), ' and zg = '
num2str(zg)])

TMG_solver
plot_1_light = r;
   t_1_light = t;



x = 13.0 * r_Neutron_star;
y =  (yscalar1 * yscalar2^2) * r_Neutron_star;
z = 0;

disp(['Initial Position x = ' num2str(x,15) ' y = ' num2str(y,15) ' z = ' num2str(z,15) ' meters' ])

phi = -2*g*m/cs/cs/y;
zg = (1/sqrt(exp(phi))) -1;  % This is the gravitational redshift.


disp(['Calculate solution for initial radial offset of phi = ' num2str(phi), ' and zg = '
num2str(zg)])

TMG_solver
plot_2_light = r;
   t_2_light = t;



x = 13.0 * r_Neutron_star;
y =  (yscalar1 * yscalar2^3) * r_Neutron_star;
z = 0;

disp(['Initial Position x = ' num2str(x,15) ' y = ' num2str(y,15) ' z = ' num2str(z,15) ' meters' ])

phi = -2*g*m/cs/cs/y;
zg = (1/sqrt(exp(phi))) -1;  % This is the gravitational redshift.


disp(['Calculate solution for initial radial offset of phi = ' num2str(phi), ' and zg = '
num2str(zg)])

TMG_solver
plot_3_light = r;
   t_3_light = t;



x = 13.0 * r_Neutron_star;
y =  (yscalar1 * yscalar2^4) * r_Neutron_star;
z = 0;

disp(['Initial Position x = ' num2str(x,15) ' y = ' num2str(y,15) ' z = ' num2str(z,15) ' meters' ])

phi = -2*g*m/cs/cs/y;
```

```
zg = (1/sqrt(exp(phi))) -1;  % This is the gravitational redshift.


disp(['Calculate solution for initial radial offset of phi = ' num2str(phi), ' and zg = '
num2str(zg)])

TMG_solver
plot_4_light = r;
   t_4_light = t;



x = 13.0 * r_Neutron_star;
y =  (yscalar1 * yscalar2^5) * r_Neutron_star;
z = 0;

disp(['Initial Position x = ' num2str(x,15) ' y = ' num2str(y,15) ' z = ' num2str(z,15) ' meters' ])

phi = -2*g*m/cs/cs/y;
zg = (1/sqrt(exp(phi))) -1;  % This is the gravitational redshift.


disp(['Calculate solution for initial radial offset of phi = ' num2str(phi), ' and zg = '
num2str(zg)])

TMG_solver
plot_5_light = r;
   t_5_light = t;


% Top view
% plot surface of Sun

as= linspace(0,2*pi,360);
xs = r_Neutron_star * cos(as);
ys = r_Neutron_star * sin(as);

plot(xs,ys,'Color',[1, 0.65, 0.35],'LineWidth',4)

hold('on')

plot(plot_1_light(:,5),plot_1_light(:,6),'LineWidth',2,'Color',[0, 0, 1], 'LineStyle', '-')
plot(plot_1_light(:,5),-plot_1_light(:,6),'LineWidth',2,'Color',[0, 0, 1], 'LineStyle', '-')

plot(plot_2_light(:,5),plot_2_light(:,6),'LineWidth',2,'Color',[0, 0, 1], 'LineStyle', '-')
plot(plot_2_light(:,5),-plot_2_light(:,6),'LineWidth',2,'Color',[0, 0, 1], 'LineStyle', '-')

plot(plot_3_light(:,5),plot_3_light(:,6),'LineWidth',2,'Color',[0, 0, 1], 'LineStyle', '-')
plot(plot_3_light(:,5),-plot_3_light(:,6),'LineWidth',2,'Color',[0, 0, 1], 'LineStyle', '-')

plot(plot_4_light(:,5),plot_4_light(:,6),'LineWidth',2,'Color',[0, 0, 1], 'LineStyle', '-')
plot(plot_4_light(:,5),-plot_4_light(:,6),'LineWidth',2,'Color',[0, 0, 1], 'LineStyle', '-')

plot(plot_5_light(:,5),plot_5_light(:,6),'LineWidth',2,'Color',[0, 0, 1], 'LineStyle', '-')
plot(plot_5_light(:,5),-plot_5_light(:,6),'LineWidth',2,'Color',[0, 0, 1], 'LineStyle', '-')

%hold('off')

plot_limit = 10.0 * r_Neutron_star;
xlim([-1.434*plot_limit,1.434*plot_limit])
ylim([-plot_limit,plot_limit])
xlabel('x axis, meters')
ylabel('y axis, meters')



%     Calculate the path of motion for a cosmic ray with the TMG solver from multiple starting
points.
%     Set cosmic ray initial velocity.

beta_option = 1;
bx = -0.7;
by = 0;
bz = 0;

disp(['Calculate the deflection of a cosmic ray passing near a Neutron star, v/c = ' num2str(bx,15)])

x = 13.0 * r_Neutron_star;
```

```
y =  (yscalar1 * yscalar2^0) * r_Neutron_star;
z = 0;

phi = -2*g*m/cs/cs/y;
zg = (1/sqrt(exp(phi))) -1;  % This is the gravitational redshift.


disp(['Calculate solution for initial radial offset of phi = ' num2str(phi), ' and zg = '
num2str(zg)])

TMG_solver
plot_1_cr= r;
   t_1_cr = t;

x = 13.0 * r_Neutron_star;
y =  (yscalar1 * yscalar2^2) * r_Neutron_star;
z = 0;

phi = -2*g*m/cs/cs/y;
zg = (1/sqrt(exp(phi))) -1;  % This is the gravitational redshift.


disp(['Calculate for initial radial offset of phi = ' num2str(phi), ' and zg = ' num2str(zg)])

TMG_solver
plot_2_cr = r;
   t_2_cr = t;

x = 13.0 * r_Neutron_star;
y =  (yscalar1 * yscalar2^3) * r_Neutron_star;
z = 0;

phi = -2*g*m/cs/cs/y;
zg = (1/sqrt(exp(phi))) -1;  % This is the gravitational redshift.


disp(['Calculate solution for initial radial offset of phi = ' num2str(phi), ' and zg = '
num2str(zg)])

TMG_solver
plot_3_cr = r;
   t_3_cr = t;

x = 13.0 * r_Neutron_star;
y =  (yscalar1 * yscalar2^4) * r_Neutron_star;
z = 0;

phi = -2*g*m/cs/cs/y;
zg = (1/sqrt(exp(phi))) -1;  % This is the gravitational redshift.


disp(['Calculate solution for initial radial offset of phi = ' num2str(phi), ' and zg = '
num2str(zg)])

TMG_solver
plot_4_cr = r;
   t_4_cr = t;


x = 13.0 * r_Neutron_star;
y =  (yscalar1 * yscalar2^5) * r_Neutron_star;
z = 0;

phi = -2*g*m/cs/cs/y;
zg = (1/sqrt(exp(phi))) -1;  % This is the gravitational redshift.


disp(['Calculate solution for initial radial offset of phi = ' num2str(phi), ' and zg = '
num2str(zg)])

TMG_solver
plot_5_cr = r;
   t_5_cr = t;


plot(plot_1_cr(:,5),plot_1_cr(:,6),'LineWidth',2,'Color',[1, 0, 0], 'LineStyle', ':')
plot(plot_1_cr(:,5),-plot_1_cr(:,6),'LineWidth',2,'Color',[1, 0, 0], 'LineStyle', ':')
```

```
plot(plot_2_cr(:,5),plot_2_cr(:,6),'LineWidth',2,'Color',[1, 0, 0], 'LineStyle', ':')
plot(plot_2_cr(:,5),-plot_2_cr(:,6),'LineWidth',2,'Color',[1, 0, 0], 'LineStyle', ':')

plot(plot_3_cr(:,5),plot_3_cr(:,6),'LineWidth',2,'Color',[1, 0, 0], 'LineStyle', ':')
plot(plot_3_cr(:,5),-plot_3_cr(:,6),'LineWidth',2,'Color',[1, 0, 0], 'LineStyle', ':')

plot(plot_4_cr(:,5),plot_4_cr(:,6),'LineWidth',2,'Color',[1, 0, 0], 'LineStyle', ':')
plot(plot_4_cr(:,5),-plot_4_cr(:,6),'LineWidth',2,'Color',[1, 0, 0], 'LineStyle', ':')

plot(plot_5_cr(:,5),plot_5_cr(:,6),'LineWidth',2,'Color',[1, 0, 0], 'LineStyle', ':')
plot(plot_5_cr(:,5),-plot_5_cr(:,6),'LineWidth',2,'Color',[1, 0, 0], 'LineStyle', ':')


plot_limit = 10.0 * r_Neutron_star;
xlim([-1.434*plot_limit,1.434*plot_limit])
ylim([-plot_limit,plot_limit])
xlabel('x axis, meters')
ylabel('y axis, meters')

title(strcat('light ray = blue line, cosmic ray = red dashed line for v/c = ', num2str(bx)))

hold('off')
```

Example_8_Apparent_Black_Hole_Shadow_Diameter_and_Emissivity.m

```
%   Example_8_Apparent_Black_Hole_Shadow_Diameter_and_Emissivity.m
%
%   Matlab script for calculating the emissivity, shadow diameter, and curved path of light escaping
from or
%   passing near an apparent black hole or quasar.  This is done for Sgr A* at the the galactic
center and for
%   the central gravitational body of M87*
%
%   Copyright 2021 Morris G. Anderson
%
%   The path of motion is calculated with the TMG_path.m MatLab function.  The TMG_solver.m
%   script applies the TMG_path.m function to obtain the solution.
%
%   Information source
%   1)  Time Matter and Gravity (TMG) by Morris G. Anderson provides a derivation of the
%       equations used in this calculation.
%       https://vixra.org/abs/2102.0123
%       https://vixra.org/pdf/2102.0123v1.pdf
%
%   2)  Physical constants
%       https://physics.nist.gov/cuu/pdf/wall_2018.pdf
%
%   3)  Astronomical data
%       https://ssd.jpl.nasa.gov/horizons.cgi
%
%   4)  Relativistic redshift of the star S0-2 orbiting the Galactic center supermassive black hole
%       https://arxiv.org/pdf/1907.10731.pdf
%
%   5)  Detection of the Schwarzschild precession in the orbit of the star S2 near the
%       Galactic centre massive black hole  https://arxiv.org/pdf/2004.07187
%
%   6)  A Method for Predicting Quasar Luminosity Consistent With the NASA/IPAC Extragalactic
Database
%       https://www.rxiv.org/abs/2102.0165
%       https://www.rxiv.org/pdf/2102.0165v1.pdf
%
%   7)  Luminet, Jean-Pierre 2018-04  Seeing Black Holes :from the Computer to the Telescope
%       https://arxiv.org/abs/1804.03909
%
%   8)  First M87 Event Horizon Telescope Results. VI. The Shadow and Mass of the Central Black Hole
%       https://arxiv.org/pdf/1906.11243.pdf
%
%   Process for this example:
%
%   Modify this script to select either Sgr A* or M87*
%
%   see embedded comments

clear
clc

%   >>>  Choose Sgr A* or M87*

core = 2;  %  set core = to 1 for Sgr A*,  set core = to 2 for M87*

%  g, universal gravitational constant. Units = m^3/(kg-s),  See Reference 2
g  = 0.66743e-10;

%   The standard speed of light, cs.
%   From Reference 1 (see TMG section 2.4.1  The Speed of light at the standard location)
cs = 299792458.0;

disp(['Newtons universal gravitational constant, G = ' num2str(g,6) ' m^3/(kg-s)'])

disp(['The standard speed of light, cs = ' num2str(cs,10) ' m/s'])

%   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


if (core == 1)

    disp(['This example calculates the emissivity as a function of radius and the apparent black hole
shadow diameter of Sgr A*'])

    m = 3.975e6;  %  Sgr* mass in multiples of our Sun - from Information source 8
```

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

    m = m_Sun * m;  %  convert to kg

    rr = 7.959; % kpc, rr is the distance to the galactic center of Sgr*.
    disp(['Sgr A* galactic center distance = ' num2str((rr),4) ' Mpc' ])
    rr = rr * 1e3 * (648000.0/pi) * 149597870700.0; % kilo-parsec to meter conversion

else

    disp(['This example calculates the emissivity as a function of radius and the apparent black hole
shadow diameter 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' ])

    m = m_Sun * m;  %  convert to kg

    rr = 16.8; % Mpc, rr is the distance to the galactic center od M87*.
    disp(['M87* galactic center distance = ' num2str((rr),4) ' Mpc' ])
    rr = rr * 1e6 * (648000.0/pi) * 149597870700.0; % Mega-parsec to meter conversion
end


cr = 299792464.335799;  %  This is the speed of light away from the influence of the Sun
                        %  See Example_5_Sun_light_deflection_Shapiro_time_delay.m for calculation

disp(['Reference speed of light  = ' num2str(cr,12) ' m/s at ' num2str(rr,8) ' meters from the
galactic center'])


%   Define the initial x, y, z, coordinates with respect to the center of mass of the governing body,
m, in meters for phi = -1.  This is the radius for which light orbits the governing body - see TMG
eq. 7-12.

phi_m1 = -1.0;
r_phi_m1 = -2*g*m/cs/cs/phi_m1;

x = 0.0;
y = r_phi_m1;
z = 0.0;

zr = (1/sqrt(exp(phi_m1))) -1; %This is the gravitational redshift (see TMG eq 4-11 and 5-27)

disp(['Initial Position x = ' num2str(x,15) ' y = ' num2str(y,15) ' z = ' num2str(z,15) ' meters' ])

disp(['Calculate light path from the photon sphere spiraling into the core'])

% ------------------------------------------------------------------------------------------
%   path_option = 2, Calculate the bending of light and the Shapiro time delay.
%                    Arc segment length is a function of radius from the governing body center.
%
%                    if st > 1, st = The intended maximum path length for the calculation
%
%                    if st < 1, sr = st. Arc segment length = sr * r(n,4) for each calculation step.
% ------------------------------------------------------------------------------------------

path_option = 2;  % Arc segment length is a function of radius from the governing body center

st  = 0.003; % arc segment path lenngth / position radius from the governing body.
np  = 248;   % number of points in addition to the starting position to be stored in memory.
nac =  20;   % number of arc segments integrated to calculate each point that is stored in memory

%   Calculate the capture path.  This is done by starting with a slightly negative angle at a
%   radius equivalent to the orbital radius for light. This radius corresponds to phi = -1.
%   (see TMG eq 4-11, 4-12 and 7-12)
%
%   Note:  A radius that gives phi = -1 is the same as the Schwarzschild radius associated with
General Relativity.
%   However in General Relativity an artificial (not real) event horizon exists at the Schwarzschild
radius.
%   This appears to be caused by the General Relativity 1st order approximation of the natural
exponential solution.
%   (see Information source 6 section 3.2 Quasar Redshift https://www.rxiv.org/pdf/2102.0165v1.pdf)

%   Define vx, vy, vz, Initial velocity in m/s.
```

```
beta_option = 1;  %If beta_option = 1, these values are treated as bx=vx/c, by=vy/c, bz=vz,c

alpha = -0.001; % degrees, slightly less than horizontal
bx    = 1.0 * cos(alpha*pi/180);
by    = 1.0 * sin(alpha*pi/180);
bz    = 0;

TMG_solver  %  call solver scrip the calculate the path

%  Calculate emissivity based on the angle between radial and the direction of motion.
%  use method provided in Information source 6, https://www.rxiv.org/pdf/2102.0165v1.pdf

for n = 1:np+1
    capture(n,1) = 180 - acos(dot( b(n,1:3), r(n,1:3) )) * 180 / pi;  % angle beteen r and b
    capture(n,2) = -2*g*m/cs/cs/r(n,4);                               % phi
    capture(n,3) = -1 + 1/sqrt(exp(capture(n,2)));                    % z, gravitational redshift
    capture(n,4) = 100*(1 - cos(capture(n,1)*pi/180));                % black body emisivity
end

plot(capture(:,2),capture(:,1),'LineWidth',2,'Color',[0, 0, 1], 'LineStyle', '-')
hold('on')
plot(capture(:,2),capture(:,4),'LineWidth',4,'Color','r', 'LineStyle', '-')

hold('off')

xlim([-10,0])
ylim([0,100])

xlabel('phi')
ylabel(' max escape angle  or  % of black body emissivity')
legend('Max escape angle to radial for Light','Equivalent % of blackbody emissivity' )

pause(20);

b_capture = b;
c_capture = c;
r_capture = r;

%   Calculate escape path.

%   Define vx, vy, vz, Initial velocity in m/s.  If beta_option = 1, then these values are treated as
bx=vx/c, by=vy/c, bz=vz,c

alpha = 0.001;  % degrees, slightly less than horizontal

beta_option = 1;
bx = -1.0 * cos(alpha*pi/180);
by =  1.0 * sin(alpha*pi/180);
bz =  0;

disp(['Calculate escape path' ])

path_option = 2;  % Arc segment length is a function of radius from the governing body center

st  = 0.006; % arc segment path lenngth / position radius from the governing body.
np  = 300;   % number of points in addition to the starting position to be stored in memory.
nac =  10;   % number of arc segments integrated to calculate each point that is stored in memory

TMG_solver  %  call solver scrip the calculate the path

b_escape = b;
c_escape = c;
r_escape = r;


%  Calculate shadow diameter.

%   Note:  A radius that gives phi = -1 is the same as the Schwarzschild radius associated with
General Relativity.
%   However in General Relativity an artificial (not real) event horizon exists at the Schwarzschild
radius.
%   This appears to be caused by the General Relativity 1st order approximation of the natural
exponential solution.
%   (see Information source 6 section 3.2 Quasar Redshift https://www.rxiv.org/pdf/2102.0165v1.pdf)

%   Schwarzschild radius = rs = r_phi_m1
```

```
for n = 1:np+1
    shadow(n,1) = r(n,4) * sin(acos(dot( b(n,1:3), r(n,1:3) ))) / r_phi_m1;
    shadow(n,2) = r(n,4) / r_phi_m1;
end

angular_diameter = 3600 * (180 / pi) * atan( shadow(end,1) * r_phi_m1 / rr ) * 2.0; % arcseconds

disp(['apparent Black Hole shadow diameter = ' num2str(shadow(end,1)) ' * 2 * rs,  rs is the
Schwarzschild radius for which phi = -1'])
disp(['apparent Black Hole shadow diameter = ' num2str(2.0 * r_phi_m1 * shadow(end,1)) ' meters' ])
disp(['apparent Black Hole shadow diameter = ' num2str(angular_diameter) ' arcseconds' ])


%   Calculated General Relativity results for comparison - see https://arxiv.org/pdf/1804.03909.pdf

angular_diameter_GR = 3600 * (180 / pi) * atan( (3*sqrt(3)/2) * r_phi_m1 / rr ) * 2.0; % arcseconds

disp(['FOR COMPARISON ONLY - GENERAL RELATIVE apparent Black Hole shadow diameter = '
num2str(angular_diameter_GR) ' arcseconds' ])


plot(shadow(:,2),shadow(:,1),'LineWidth',2,'Color','b', 'LineStyle', '-')

%xlabel('radial distance / rs')

xlabel(strcat('(radial distance / rs),  rs (meters) = ', num2str(min(r_phi_m1))))

ylabel('shadow diameter / rs')
title('shadow diameter growth with distance' )

ylim([2, exp(1)])

pause(20);


%  plot r for min phi of capture calculation
as= linspace(0,2*pi,361);
x_phi = min(r_capture(:,4)) * cos(as);
y_phi = min(r_capture(:,4)) * sin(as);
plot(x_phi/r_phi_m1,y_phi/r_phi_m1,'Color','b','LineWidth',1, 'LineStyle', ':')

hold('on')


%  plot r for Apparent Black Hole Shadow dimeter
x_bhsd = shadow(end,1) * r_phi_m1 * cos(as);
y_bhsd = shadow(end,1) * r_phi_m1 * sin(as);
plot(x_bhsd/r_phi_m1,y_bhsd/r_phi_m1,'Color',[1, 0.65, 0.35],'LineWidth',1, 'LineStyle', ':')



%   Rotate and plot light paths to illustrate shadow
theta = (-r_escape(end,8) -180)*pi/180; % total rotation

%   Define the initial position and motion unit vectors
rhat(:,:) =  r_escape(:,1:3);
bhat(:,:) =  b_escape(:,1:3);

%   Rotate unit vectors.  Use Rodrigues' rotation formula.

kr = cross(rhat(1,:),bhat(1,:));
kr = kr / sqrt( kr(1,1)*kr(1,1) + kr(1,2)*kr(1,2)+ kr(1,3)*kr(1,3) );

np = size(r_escape,1);

for n = 1:np
    vr1 = rhat(n,:);
    vr2 = cos(theta)*vr1 + cross(kr,vr1)*sin(theta) + kr*(dot(kr,vr1))*(1 - cos(theta));
    vr2 = vr2 / sqrt( vr2(1)*vr2(1) + vr2(2)*vr2(2) + vr2(3)*vr2(3) );
    xyz_1(n,1:3) = vr2 * r_escape(n,4)/r_phi_m1;
end

plot(xyz_1(:,1), xyz_1(:,2),'Color','b','LineWidth',2, 'LineStyle', ':')
hold('on')
plot(xyz_1(:,1),-xyz_1(:,2),'Color','b','LineWidth',2, 'LineStyle', ':')
```

```
clear('rhat')
clear('bhat')

%   Define the initial position and motion unit vectors
rhat(:,:) =  r_capture(:,1:3);
bhat(:,:) =  b_capture(:,1:3);

%   Rotate unit vectors.  Use Rodrigues' rotation formula.

kr = cross(rhat(1,:),bhat(1,:));
kr = kr / sqrt( kr(1,1)*kr(1,1) + kr(1,2)*kr(1,2)+ kr(1,3)*kr(1,3) );

np = size(r_capture,1);

for n = 1:np
    vr1 = rhat(n,:);
    vr2 = cos(theta)*vr1 + cross(kr,vr1)*sin(theta) + kr*(dot(kr,vr1))*(1 - cos(theta));
    vr2 = vr2 / sqrt( vr2(1)*vr2(1) + vr2(2)*vr2(2) + vr2(3)*vr2(3) );
    xyz_2(n,1:3) = vr2 * r_capture(n,4)/r_phi_m1;
end

plot(xyz_2(:,1), xyz_2(:,2),'Color','b','LineWidth',2, 'LineStyle', ':')
hold('on')
plot(xyz_2(:,1),-xyz_2(:,2),'Color','b','LineWidth',2, 'LineStyle', ':')



%   Calculate the radius assuming a density similarto Neutron Star RX J0720.4-3125.
%   see https://arxiv.org/pdf/1911.08107.pdf
%   See example 7 for how the density is calculated

r_neutron_star_density = ( 3*m/(9.87460e15 * 4 * pi) )^(1/3); % See example 7 for neutron Star
Density

disp(['Core radius / rs  = ' num2str(r_neutron_star_density/r_phi_m1), ',  This is calculated
assuming the density is similar to that of a neutron star'])

disp(['Core radius / shadow radius = ' num2str(r_neutron_star_density/(r_phi_m1 * shadow(end,1))), ',
This is calculated assuming the density is similar to that of a neutron star'])

x_nd = shadow(end,1) * r_neutron_star_density * cos(as)/r_phi_m1;
y_nd = shadow(end,1) * r_neutron_star_density * sin(as)/r_phi_m1;
plot(x_nd,y_nd,'Color',[0.5, 0.0, 0.5],'LineWidth',1, 'LineStyle', '-.')

window_ar = 31.45/45.5; %x/y new ipad full
ylim([-3, 3])
xlim([-3/window_ar,3/window_ar])

xlabel(strcat('(radial distance / rs),  rs (meters) = ', num2str(min(r_phi_m1))))
ylabel('y axis, (radial distance / rs)')

%  legend(strcat('phi = ', num2str(min(capture(:,2)))), 'Apparent Black Hole Shadow', 'capture and
escape path')

%  pause(20);



%   plot light paths illustrating shadow



path_option = 4;   % Calculate path with equal arc length segments

np = 400;
nac = 1;

disp(['Calculate the path of motion for light from multiple starting points to illustrate the
shadow.' ])

beta_option = 1;
bx = -1.0;
by = 0;
bz = 0;

yscalar1 = 1.2;
yscalar2 = 1.4;
```

```
x = 13.0 * r_phi_m1*exp(1);
y =  (yscalar1 * yscalar2^0.1) * r_phi_m1*exp(1);
z = 0;

st =  26.0 * r_phi_m1*exp(1);  %  Total path length for calculation

disp(['Initial Position x = ' num2str(x,15) ' y = ' num2str(y,15) ' z = ' num2str(z,15) ' meters' ])

phi = -2*g*m/cs/cs/y;
zg = (1/sqrt(exp(phi))) -1;  % This is the gravitational redshift.

disp(['Calculate solution for initial radial offset of phi = ' num2str(phi), ' and zg = '
num2str(zg)])

TMG_solver
plot_1_light = r;



x = 13.0 * r_phi_m1*exp(1);
y =  (yscalar1 * yscalar2^1) * r_phi_m1*exp(1);
z = 0;

disp(['Initial Position x = ' num2str(x,15) ' y = ' num2str(y,15) ' z = ' num2str(z,15) ' meters' ])

phi = -2*g*m/cs/cs/y;
zg = (1/sqrt(exp(phi))) -1;  % This is the gravitational redshift.

disp(['Calculate solution for initial radial offset of phi = ' num2str(phi), ' and zg = '
num2str(zg)])

TMG_solver
plot_2_light = r;



x = 13.0 * r_phi_m1*exp(1);
y =  (yscalar1 * yscalar2^2) * r_phi_m1*exp(1);
z = 0;

disp(['Initial Position x = ' num2str(x,15) ' y = ' num2str(y,15) ' z = ' num2str(z,15) ' meters' ])

phi = -2*g*m/cs/cs/y;
zg = (1/sqrt(exp(phi))) -1;  % This is the gravitational redshift.

disp(['Calculate solution for initial radial offset of phi = ' num2str(phi), ' and zg = '
num2str(zg)])

TMG_solver
plot_3_light = r;



x = 13.0 * r_phi_m1*exp(1);
y =  (yscalar1 * yscalar2^3) * r_phi_m1*exp(1);
z = 0;

disp(['Initial Position x = ' num2str(x,15) ' y = ' num2str(y,15) ' z = ' num2str(z,15) ' meters' ])

phi = -2*g*m/cs/cs/y;
zg = (1/sqrt(exp(phi))) -1;  % This is the gravitational redshift.

disp(['Calculate solution for initial radial offset of phi = ' num2str(phi), ' and zg = '
num2str(zg)])

TMG_solver
plot_4_light = r;



x = 13.0 * r_phi_m1*exp(1);
y =  (yscalar1 * yscalar2^4) * r_phi_m1*exp(1);
z = 0;

disp(['Initial Position x = ' num2str(x,15) ' y = ' num2str(y,15) ' z = ' num2str(z,15) ' meters' ])

phi = -2*g*m/cs/cs/y;
zg = (1/sqrt(exp(phi))) -1;  % This is the gravitational redshift.
```

```
disp(['Calculate solution for initial radial offset of phi = ' num2str(phi), ' and zg = '
num2str(zg)])

TMG_solver
plot_5_light = r;


% Top view

plot(plot_1_light(:,5)/r_phi_m1,plot_1_light(:,6)/r_phi_m1,'LineWidth',2,'Color',[0, 0, 1],
'LineStyle', '-')
plot(plot_1_light(:,5)/r_phi_m1,-plot_1_light(:,6)/r_phi_m1,'LineWidth',2,'Color',[0, 0, 1],
'LineStyle', '-')

plot(plot_2_light(:,5)/r_phi_m1,plot_2_light(:,6)/r_phi_m1,'LineWidth',2,'Color',[0, 0, 1],
'LineStyle', '-')
plot(plot_2_light(:,5)/r_phi_m1,-plot_2_light(:,6)/r_phi_m1,'LineWidth',2,'Color',[0, 0, 1],
'LineStyle', '-')

plot(plot_3_light(:,5)/r_phi_m1,plot_3_light(:,6)/r_phi_m1,'LineWidth',2,'Color',[0, 0, 1],
'LineStyle', '-')
plot(plot_3_light(:,5)/r_phi_m1,-plot_3_light(:,6)/r_phi_m1,'LineWidth',2,'Color',[0, 0, 1],
'LineStyle', '-')

plot(plot_4_light(:,5)/r_phi_m1,plot_4_light(:,6)/r_phi_m1,'LineWidth',2,'Color',[0, 0, 1],
'LineStyle', '-')
plot(plot_4_light(:,5)/r_phi_m1,-plot_4_light(:,6)/r_phi_m1,'LineWidth',2,'Color',[0, 0, 1],
'LineStyle', '-')

plot(plot_5_light(:,5)/r_phi_m1,plot_5_light(:,6)/r_phi_m1,'LineWidth',2,'Color',[0, 0, 1],
'LineStyle', '-')
plot(plot_5_light(:,5)/r_phi_m1,-plot_5_light(:,6)/r_phi_m1,'LineWidth',2,'Color',[0, 0, 1],
'LineStyle', '-')


plot_limit = 5.0 * r_phi_m1*exp(1)/r_phi_m1;
xlim([-1.434*plot_limit,1.434*plot_limit])
ylim([-plot_limit,plot_limit])
xlabel(strcat('y axis, (radial distance / rs),  rs (meters) = ', num2str(min(r_phi_m1))))
ylabel('y axis, (radial distance / rs)')


%  Calculate light paths inside of shadow that are captured or escape path at phi = -20

phi_m20 = -20;
x = 0.0;
y = -2*g*m/cs/cs/phi_m20;
z = 0;

alpha = 89.999995; %degrees

disp(['Calculate sample light escape path for phi = ' num2str(phi_m20) ', initial angle between path
and radial = ' num2str(90 - alpha), ' deg'])

beta_option = 1;
bx = 1.0 * cos(alpha*pi/180);
by = 1.0 * sin(alpha*pi/180);
bz = 0;

path_option = 2;   % Calculate path

st  = 0.00175
np  = 400;
nac = 10;


TMG_solver

r_phi_m20_1  = r;
rc_phi_m20_1 = rc;
c_phi_m20_1  = c;
b_phi_m20_1  = b;


%   Rotate and plot light paths to illustrate shadow
clear('rhat')
clear('bhat')
```

```
theta = (-r_phi_m20_1(end,8) -270)*pi/180; % total rotation

%   Define the initial position and motion unit vectors
rhat(:,:) =  r_phi_m20_1(:,1:3);
bhat(:,:) =  b_phi_m20_1(:,1:3);

%   Rotate unit vectors.  Use Rodrigues' rotation formula.

kr = cross(rhat(1,:),bhat(1,:));
kr = kr / sqrt( kr(1,1)*kr(1,1) + kr(1,2)*kr(1,2)+ kr(1,3)*kr(1,3) );

np = size(r_phi_m20_1,1);

for n = 1:np
    vr1 = rhat(n,:);
    vr2 = cos(theta)*vr1 + cross(kr,vr1)*sin(theta) + kr*(dot(kr,vr1))*(1 - cos(theta));
    vr2 = vr2 / sqrt( vr2(1)*vr2(1) + vr2(2)*vr2(2) + vr2(3)*vr2(3) );
    xyz_3(n,1:3) = vr2 * r_phi_m20_1(n,4)/r_phi_m1;
end

plot(xyz_3(:,1), xyz_3(:,2),'LineWidth',1,'Color','r', 'LineStyle', ':')
plot(xyz_3(:,1),-xyz_3(:,2),'LineWidth',1,'Color','r', 'LineStyle', ':')



x = 0.0;
y = -2*g*m/cs/cs/phi_m20;
z = 0;

alpha = 89.99999725; %degrees

disp(['Calculate sample light escape path for phi = ' num2str(phi_m20) ', initial angle between path
and radial = ' num2str(90 - alpha), ' deg'])

beta_option = 1;
bx = 1.0 * cos(alpha*pi/180);
by = 1.0 * sin(alpha*pi/180);
bz = 0;

TMG_solver

r_phi_m20_2  = r;
rc_phi_m20_2 = rc;
c_phi_m20_2  = c;
b_phi_m20_2  = b;


%   Rotate and plot light paths to illustrate shadow
clear('rhat')
clear('bhat')

theta = (-r_phi_m20_2(end,8) -270)*pi/180; % total rotation

%   Define the initial position and motion unit vectors
rhat(:,:) =  r_phi_m20_2(:,1:3);
bhat(:,:) =  b_phi_m20_2(:,1:3);

%   Rotate unit vectors.  Use Rodrigues' rotation formula.

kr = cross(rhat(1,:),bhat(1,:));
kr = kr / sqrt( kr(1,1)*kr(1,1) + kr(1,2)*kr(1,2)+ kr(1,3)*kr(1,3) );

np = size(r_phi_m20_2,1);

for n = 1:np
    vr1 = rhat(n,:);
    vr2 = cos(theta)*vr1 + cross(kr,vr1)*sin(theta) + kr*(dot(kr,vr1))*(1 - cos(theta));
    vr2 = vr2 / sqrt( vr2(1)*vr2(1) + vr2(2)*vr2(2) + vr2(3)*vr2(3) );
    xyz_4(n,1:3) = vr2 * r_phi_m20_2(n,4)/r_phi_m1;
end

plot(xyz_4(:,1), xyz_4(:,2),'LineWidth',1,'Color','r', 'LineStyle', ':')
plot(xyz_4(:,1),-xyz_4(:,2),'LineWidth',1,'Color','r', 'LineStyle', ':')


x = 0.0;
y = -2*g*m/cs/cs/phi_m20;
```

```
z = 0;

alpha = 89.999999; %degrees

disp(['Calculate sample light escape path for phi = ' num2str(phi_m20) ', initial angle between path
and radial = ' num2str(90 - alpha), ' deg'])

beta_option = 1;
bx = 1.0 * cos(alpha*pi/180);
by = 1.0 * sin(alpha*pi/180);
bz = 0;

TMG_solver

r_phi_m20_3  = r;
rc_phi_m20_3 = rc;
c_phi_m20_3  = c;
b_phi_m20_3  = b;


%   Rotate and plot light paths to illustrate shadow
clear('rhat')
clear('bhat')

theta = (-r_phi_m20_3(end,8) -270)*pi/180; % total rotation

%   Define the initial position and motion unit vectors
rhat(:,:) =  r_phi_m20_3(:,1:3);
bhat(:,:) =  b_phi_m20_3(:,1:3);

%   Rotate unit vectors.  Use Rodrigues' rotation formula.

kr = cross(rhat(1,:),bhat(1,:));
kr = kr / sqrt( kr(1,1)*kr(1,1) + kr(1,2)*kr(1,2)+ kr(1,3)*kr(1,3) );

np = size(r_phi_m20_3,1);

for n = 1:np
    vr1 = rhat(n,:);
    vr2 = cos(theta)*vr1 + cross(kr,vr1)*sin(theta) + kr*(dot(kr,vr1))*(1 - cos(theta));
    vr2 = vr2 / sqrt( vr2(1)*vr2(1) + vr2(2)*vr2(2) + vr2(3)*vr2(3) );
    xyz_5(n,1:3) = vr2 * r_phi_m20_3(n,4)/r_phi_m1;
end

plot(xyz_5(:,1), xyz_5(:,2),'LineWidth',1,'Color','r', 'LineStyle', ':')
plot(xyz_5(:,1),-xyz_5(:,2),'LineWidth',1,'Color','r', 'LineStyle', ':')


x = 0.0;
y = -2*g*m/cs/cs/phi_m20;
z = 0;

alpha = 89.99999; %degrees

disp(['Calculate sample light capture path for phi = ' num2str(phi_m20) ', initial angle between path
and radial = ' num2str(90 - alpha), ' deg'])

beta_option = 1;
bx = 1.0 * cos(alpha*pi/180);
by = 1.0 * sin(alpha*pi/180);
bz = 0;

TMG_solver

r_phi_m20_4  = r;
rc_phi_m20_4 = rc;
c_phi_m20_4  = c;
b_phi_m20_4  = b;


%   Rotate and plot light paths to illustrate shadow
clear('rhat')
clear('bhat')

theta = (0)*pi/180; % total rotation

%   Define the initial position and motion unit vectors
rhat(:,:) =  r_phi_m20_4(:,1:3);
```

```
bhat(:,:) =  b_phi_m20_4(:,1:3);

%   Rotate unit vectors.  Use Rodrigues' rotation formula.

kr = cross(rhat(1,:),bhat(1,:));
kr = kr / sqrt( kr(1,1)*kr(1,1) + kr(1,2)*kr(1,2)+ kr(1,3)*kr(1,3) );

np = size(r_phi_m20_4,1);

for n = 1:np
    vr1 = rhat(n,:);
    vr2 = cos(theta)*vr1 + cross(kr,vr1)*sin(theta) + kr*(dot(kr,vr1))*(1 - cos(theta));
    vr2 = vr2 / sqrt( vr2(1)*vr2(1) + vr2(2)*vr2(2) + vr2(3)*vr2(3) );
    xyz_6(n,1:3) = vr2 * r_phi_m20_4(n,4)/r_phi_m1;
end

plot(xyz_6(:,1), xyz_6(:,2),'LineWidth',1,'Color','r', 'LineStyle', ':')
plot(xyz_6(:,1),-xyz_6(:,2),'LineWidth',1,'Color','r', 'LineStyle', ':')

hold('off')
```