Periapsis precession of all viable planets

Lasse M. Dragsjø
Supervisor: Jonas R.[1]

[a]*Department of Physics, Norwegian University of Science and Technology, 7491 Trondheim.*

Abstract

General relativity is at the forefront of modern physics. More tests of the theory are highly valued, to push the limits of physics. One particular field this can be applied to, is periapsis precession of planets. This paper offers a Python program to analytically and numerically predict the periapsis precession of all planets with non-zero eccentricities. The program is an open source that anyone can use for any reason, whether its for education, program improving, archiving, comparing, etc. This allows for more availability in this field of study. Our Solar System planets, and some exoplanets are highlighted to show the program's performance. The observational, analytical, and numerical values match very well with each other for most eccentricities, however, a small deviation of up to 4% occurs for some high eccentricity values. Therefore, these calculated values with non-high eccentricity orbits gives very similar prediction values on the periapsis precession of the planet, while high eccentricity orbits still gives a very good prediction, but with a deviation of 4% at most.

Generell relativitetsteori er i forkant av moderne fysikk. Flere tester på teorien er høyt verdsatt, for a skyve fysikkens grenser. Ett spesifikt felt dette kan anvendes på er periapsis presesjon av planeter. Denne oppgaven tibyr et Python program til å analytisk og numerisk forutsi periapsis presesjon for alle planeter med mer enn null eksentrisitet. Programmet er et åpent kilde som hven som helst kan bruke for hviklen som helst grunn, om det er for læring, programforbedring, arkivering, sammenligninger, osv. Dette tillater for mer tilgjengelighet i dette fagfeltet. Vårt solsystems planeter, og noen eksoplaneter er uthevet for å vise programmets ytelse. De observasjonelle, analytiske, og numeriske verdiene samsvarer veldig godt med hveradnre for de fleste eksentrisiteter, men noe avvik av opp til 4% oppstår for noen høye eksentrisitetsverdier. Derfor gir disse berregnede verdier for lave og middels lave eksentrisitetsverdier veldig like forutsigende verdier for periapsis presesjonen av en planet, mens planetbaner med høy eksentrisitet gir likevel en veldig god forutsigelse, men med et avvik av opp til 4%.

1. Introduction

General Relativity (GR) is the best theory of gravity as of today. A great amount of contributions have been put into the theory to make it reach the level of status it currently has. Ever since the Renaissance, and especially since the Scientific Revolution in the 16th century, physics have been at the forefront of knowledge. It has been pushed by great historical figures like Copernicus, Gallileo, Kepler, and Newton; each one of them performing major contributions to science. Arguably, the greatest achievements of them all, is Newton's book of "Philosophiae naturalis principia mathematica", which was published the 5th of July 1687, that introduced Newtonian Mechanics (N) to the world. Amongst other revelations, the book described the universal law of gravitation (ULOG), which explained the orbits of the planets, and described their motions from a gravitational force. Until then, the true motion of the planets had only been able to be partially described by Kepler with his laws. Kepler's laws did not explain why they moved the way they did. Newton's ULOG changed this.

The ultimate verification came when the prediction of the return of Halley's Comet became correct. In 1705, Edmond Halley used Newton's ULOG, and a list of 24 known comets, to predict the comets path, taking into account the influence of Jupiter and Saturn [1]. He noticed that three of them had very similar orbital parameters. He concluded it's the same comet, and predicted its closest arrival in 1758. In 1758, it was observed. The closest arrival came 3 months later at the 13th of March 1759. This deviation was small enough that the prediction was confirmed to be correct.

It showed that Newton's ULOG successfully predicted the orbital path of a comet, something that has never been done before, and therefore showing that the ULOG was powerful. It was a great step towards our understanding of the physics in the skies above us. More evidence of the ULOG would come as time went on; one of those was the discovery of Neptune.

After scientists calculated the orbital path of Uranus by using Newton's ULOG, and accounting for all the other planet's gravitational tugs and pulls, and some clever approximations, they discovered that Uranus was lagging behind compared to the observed orbital path [2]. This was a mystery no one could explain. In 1845, Le Verrier picked up on this problem. He reasoned there must be an-

other planet tugging and pulling on Uranus to account for this, and predicted its location. One year later in 1846, the predicted location was sent to the Berlin Observatory, and almost immediately, around one degree off of the prediction, they found Neptune [2].

This was shown to be yet another testament to the power of Newton's ULOG, and how it was the correct equation to describe the motion of the planets. A similar problem arose some time later. However, this time, it would prove to be a different story.

Le Verrier figured out the same perihelion precession deviation with Mercury by using human calculators and 14 accurate transit times [2]. From the experiences of discovering Neptune, he did the same calculations, and predicted a new planet. He would later give this planet the name Vulcan. The difference here was, such a planet would be so bright, it would already have been observed. An alternative explanation was given, as in, multiple small objects collaboratively tugging and pulling on Mercury, called Vulcanoids. But they were never found either. After decades of observation, the Vulcanoids were in practice proven not to exist due to lack of evidence [2]. That means there were no planet to deviate Mercury's path. Therefore, Newton's ULOG was unable to solve this deviation, showing that there was something else needed to explain it.

Many years later the solution would come. Published in the 25th of November 1915 by Albert Einstein, the theory of GR is an extension to Special Relativity, describing the behaviour of the space-time curvature [3]. The theory predicted, as a consequence, that perihelion precession occurs without any object other than the parent star to tug and pull on the planet. When compared to the observations, it matched the missing observed perihelion precession of around 43 arcseconds per century, with only a very slight error. It solved Mercury's mystery. It also predicted that the light from stars will be bent, and the redshift due to the gravitational well, and both were later confirmed [4] [5]. An explanation, and a confirmed prediction, made GR the superior theory of gravitation, as Newton's ULOG becomes the approximation to GR. It was another step towards finding the solution to the orbital mechanics we observe in our skies.

Over a hundred years later, GR remains the leading theory of gravity in science, explaining more phenomenons than any other theory of gravity. However, observations have been building up that GR can't explain [6].

Therefore there is a desire to test GR for every prediction it gives. Despite expecting it to give deviations only on extreme gravity and extreme distances, where the periapsis precession doesn't have either of those, a test on it on exoplanets has never been done before. This gives a basis on spreading availability on this problem. The usual site of cataloging these exoplanets, "NASA Exoplanet Archive", does not include their periapsis precession values. This is because there currently doesn't exist any method of observing and measuring the periapsis precession. However, they can be predicted.

This is exactly what this paper aims to achieve: Make a program to automatically calculate and predict the numerical and analytical periapsis precession of exoplanets, and give error estimates of it. The program is free to be used by anyone for any reason.

2. Mathematical theory

GR explained the missing perihelion precession observed in Mercury, something the Newtonian ULOG was unable to do. To get the GR version of Newton's ULOG, The GR acceleration is needed. The way to fully derive it analytically and numerically, is very complex and long. We include both analytical and numerical predictions, since two different methods of predicting the same target is stronger than only one. The full derivation can be read from the two references it is acquired from [7] [2]. This paper shows the basis of how the derivation is done.

GR is based off of 2 postulates. Physics is the same in all inertial frames, and the speed of light is a constant value, always [3]. The second postulate gives a 4th coordinate: ct. So the dimensions become x, y, z, and ct. Altogether, they are called spacetime, because time makes up a coordinate alongside the 3 normal spatial coordinates, linking space and time together.

The two postulates means that a normal straight line for a moving object might not be a normal straight line for an observer, when GR effects are at play. Therefore, the normal equations of mechanics needs to be rewritten in order to take this GR effect into account. To study the periapsis precession, this equation should specifically include GR effects about relativistic orbital mechanics, for example what effects a massive object will have on a planet's orbit. This site [7] calculates such an equation using many aspects of GR, boundary conditions, and approximations. It results in the Schwarzschild solution:

$$\vec{ds}^{\,2} = (1+rs/\vec{r})^{-1}\vec{dr}^{\,2} - r^2(d\theta^2+\sin\theta^2 d\phi^2) - c^2(1+rs/\vec{r})dt^2 \tag{1}$$

where $rs = 2GM/c^2$

This is the "invariant interval around a mass M in a static spherically symmetric spacetime" [7]. This is useful, as this object with a mass M is a good approximation for what a star is, and what a star can do on the planets in the GR physics. With this, another set of difficult calculations are made by this site [2], and it finds that the angular movement per orbit for GR is approximately:

$$2\pi(1+\epsilon) \tag{2}$$

where $\epsilon = 3G^2M^2/(c^2\vec{h}^2)$ and $\vec{h} = \vec{r}\times\vec{v}$

This equation is the normal period of a Newtonian orbit, except it adds onto an ϵ. It is the equation that will be used to determine the analytical perihelion shift in the orbit when the Newtonian part is removed, so $2\pi\epsilon$. The reason this can be done is explained in the derivation of

the numerical equation. The ϵ comes from the average of the $(1 - r\vec{s}/\vec{r})$ part from the Schwarzschild solution (1). It says that dr will be inversely higher for lower r. This means that for ds to be compensated, more time is needed on the radial part. Therefore, less change on the radius is made. This comes from the fact that the speed of light is constant for all reference frames. That means the planet has more time than usual to do the rotation around its host star. Because a Newtonian orbit keeps the periapsis still when there are no other planets, extra rotation from GR will force the periapsis to rotate. That is how GR causes periapsis precession.

In the following discussion, the procedure of Körber et al's paper [8] of numerical relativistic calculations of Mercury's orbit will be used. The paper uses an ansatz that the GR acceleration is:

$$\vec{a} = (c^2 r\vec{s}/(2r^2))(1 + \alpha(r\vec{s}/\vec{r}) + \beta(\vec{rl}/\vec{r})^2)\vec{r}/r \tag{3}$$

where $r\vec{s} = 2GM/c^2$ and $\vec{rl} = (\vec{r} \times \vec{v}/c)^2$. Setting $\alpha = 0$ and $\beta = 3$ from GR, replacing rs with its variables, and seeing $\vec{r}/r$ as a direction, gives:

$$\vec{a} = (GM/\vec{r}^2)(1 + 3(\vec{rl}/\vec{r})^2)$$

Notice how this has a similar structure to the $2\pi(1 + \epsilon)$ (2). The period of the normal Newtonian orbit is:

$$2\pi(1 + 0) \tag{4}$$

The Newtonian acceleration is:

$$\vec{a} = GM/\vec{r}^2 * (1 + 0) \tag{5}$$

Comparing the two orbital paths (4) (2) and the two acceleration equations (5) (2), it can be concluded that a linear approximation will make the GR acceleration (2) give the GR orbital path (2), if $\epsilon = 3G^2M^2/(c^2\vec{h}^2) = 3(\vec{rl}/\vec{r})^2$. This is why the analytical equation works.

This is what will be shown in this derivation. The centrifugal acceleration will be used to derive it:

$$\vec{a} = \vec{v}^2/\vec{r} \tag{6}$$

This is the derivation:
$3G^2M^2/(c^2\vec{h}^2)$ (starting equation)
$3\vec{a}^2\vec{r}^4/(c^2\vec{h}^2)$ (4)
$3\vec{v}^4\vec{r}^2/(c^2\vec{h}^2)$ (6)
$3\vec{v}^2/(c^2)$ $(\vec{h} = (\vec{r} \times \vec{v})$
$3(\vec{rl}/\vec{r})^2$ $(\vec{rl} = (\vec{r} \times \vec{v}/c)^2)$

This proves that (2) is the GR acceleration force. The eccentricity (e) plays a major role in determining this new term. r is proportional to $1 - e$, and v is proportional to $((1 + e)/(1 - e))^{1/2}$, which means that $rl/rr * v/r = v$ is proportional to $((1 + e)/(1 - e))^{1/2}$. It means the added term diverges to infinity when the eccentricity goes to 1. The equation also says that periapsis precession still happens when there is zero eccentricity. However, with zero eccentricity, there is no periapsis to measure, so the only way to measure it is to observe the timing of when it arrives at the same spot, which is not done on the program.

This (2) will be the equation used to numerically simulate the GR caused periapsis precession.

3. Numerical theory

The Python program is using an iterative process to go through the time evolution of the planets orbit to simulate the rate of periapsis precession. The numerical program will use Euler-Cromer's method [8]. The program takes in both N and GR acceleration for the current position, and uses $v = a * \Delta t$ and $r = v * \Delta t$ to extract a new velocity and position. The N is included as a comparison to GR. The newly calculated position is then used for calculating the new acceleration. That is how one step of the iteration process is done. This process is repeated, until the simulation has completed a set amount of orbits. Afterwards, the program finds where the periapsis has shifted for each orbit, and averages the periapsis shift to get the numerical periapsis precession value. The error of the N starting position, and the N and GR motion variance error, are put together to create the uncertainty of the periapsis precession value. The analytical uncertainties are put through Gauss' law of error propagation, and put together into one uncertainty, like the numerical one. Finally, the analytical and numerical values and uncertainties are printed out for the user. To do this, the program needs input data from the planets.

Data from the exoplanets and their uncertainties are extracted from https://exoplanetarchive.ipac.caltech.edu/ [9]. Data from the Solar System planets and their uncertainties are also extracted [10] [11] [12] [13] [14] [15] [16].

Four of the planet's parameters, and their respective uncertainties, are extracted: eccentricity, orbital period, semi-major axis, and solar mass. Orbits with an eccentricity of zero are not included. For the other three values, the program only needs two of them, since the last one can be calculated from the other two. This way, more planets are viable to be studied.

This filtered data of the planets will be used for the analytical and numerical calculations. The program uses four parameters as the input: the planet name, the set of values from a research group to be chosen from, the amount of orbital turns, and the time step accuracy.

The program uses numerous methods to improve the performance. For a relativistic orbit, the amount of the added term from the acceleration equation (2) is set at three. When it's multiplied with $(rl/r)^2$, the total added term becomes very small. In programming, very small values tend to have high relative uncertainty. This is because the absolute uncertainty remains the same for any numeric time step. This means a small value compared to the absolute uncertainty gets the relative uncertainty high. One way to solve this is to increase the value of the added term.

The equation is changed to add a variable β to the added term, and divide it by three for the purpose of making the equation look clean:

$$\vec{a} = (GM/\vec{r}^2)(1 + \beta(\vec{rl}/\vec{r})^2) \tag{7}$$

This works for three reasons. Firstly, the absolute uncertainty remains unchanged when β is changed, therefore increasing the β will decrease the relative uncertainty. Secondly, this increase doesn't interfere with the approximation of the added GR term, since the value starts off being very small. It means β can be increased without increasing the significance of a better approximated GR term. Thirdly, when the value of the added term is increased, the resulting value needs to be decreased in a proper way, to get the original value back. According to Körber et al's paper [8], the β and the periapsis shift have a linear proportion for small values. This means that a division of the resulting value with β, and multiplying it by three, will in total give the original value, but with a much reduced relative uncertainty.

This method of scaling up the β value, running the simulation, and then scaling down again, is called dimensional analysis [8]. It usually only works for the order of magnitude if its the only method at play. This same technique is used on the constant values, so they are set at the Einstein units.

When the time step is a constant across the entire simulated orbit, it can have an effect on the performance. If the time step is too low, the program takes too long at apsis on high eccentricity orbits, and on all parts of low eccentricity orbits. It is because relative uncertainty builds up as many iterations are needed, while at the same time requiring great precision for each iteration. If the time step is too high, the program struggles to be accurate at the high-velocity periapsis part of the orbit, and can shift the periapsis in an unwanted way. Therefore, the time step is calculated from each position to account for this.

How the program is ran, is written on Appendix A, and the main program is in Appendix B. For other useful programs, see Appendix D, E, and especially C.

4. Results and discussion

Results and discussion will be merged into the same chapter, as the tables and figures are spread throughout this section.

The numerical values are almost always slightly higher than the observational and analytical values 1, the exception being HD 20782 b 3, which is slightly lower. All the values for the Solar System planets follow this trend, and match closely with each other. It shows the program has successfully predicted the perihelion precession values of the Solar System planets, and is therefore excellent at predicting the values for all other planets.

Earth's observation for reference one, and especially Jupiter's observation for reference two, are off compared to the other observational, analytical, and numerical values

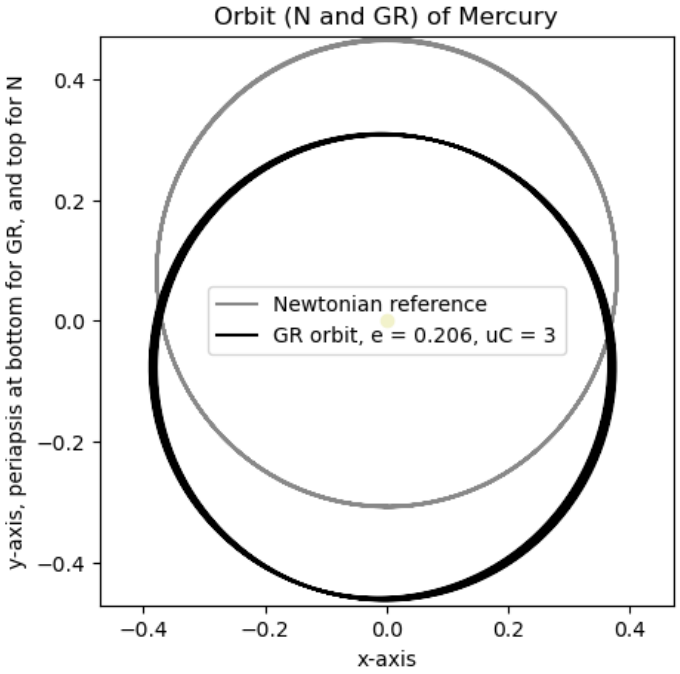

Figure 1: This image shows the orbit of Mercury. Notice how the black circle is thicker than the gray one. This is the $\beta/3$ amplified GR effect at play.

1. This is likely due to the authors of those sites [17] [18] either approximating the values too much, setting in the wrong values, or setting the first significant digit on the wrong order of magnitude. It is likely, because all the other values agree with each other.

Uranus and Neptune are missing their observational data 1. This is due to their long orbital period. Humanity has not had enough time since they gained precise enough measurements to fully measure their perihelion precession. In the future, we will have enough time and precision to measure these, and the values can be compared to the analytical and numerical values.

The analytical values for the three references [19] [17] [18] does not have an uncertainty 1. This is because the sources assume that the periapsis precession is an absolute value with no uncertainty, unlike the program, which uses the uncertainty of the planet's parameters to calculate the analytical periapsis precession unceratinty 2.

Most of the uncertainty on the analytical values for the Solar System planets are not given from a reference [10], and are instead assumed to be varying from the last significant digit 2. This will make the uncertainty much lower than what would be expected 1. The Sun's mass has a different uncertainty, since a reference for its true uncertainty was found. The same goes for Earth's semi major axis, when its assumed that its uncertainty is that value divided by the astronomical unit value minus one. Neptune's orbital period uncertainty is much higher than the other planets. This is because the reference [10] that presents Neptune's orbital period gives a value that is less

Table 1: This table shows the observational, analytical, and numerical values of the perihelion precession in arcseconds per century for the eight Solar System planets. For the observational values, The Earth-1 and Jupiter-2 values are off from the real values [17] [18]. Uranus and Neptune are missing their data, since no observational measurements have been done on them. For the analytical values, Three outside references [19] [17] [18], and the calculated analytical values with its uncertainties, are listed on the table. For the numerical values, one outside reference [17], and the calculated numerical values with its uncertainties, are listed on the table. All the references does not include an uncertainty. Digits that come after the second significant digit of the uncertainty value are not included.

Planet	Obs 1 [17]	Obs 2 [18]	Ana prog	Ana 1 [19]	Ana 2 [17]	Ana 3 [18]	Num prog	Num site [17]
Mercury	43.1(5)	43.1(5))	42.997309(13)	42.980	42.98	42.9822	43.0(31)	43.03
Venus	8(5))	8.6247(5))	8.6290407(16)	8.6247	8.648	8.6247	8.75(38)	8.655
Earth	5(1))	3.8387(4))	3.8405859(12)	3.8374	3.839	3.83881	3.85(22)	3.840
Mars	1.3624(5))	1.3565(4))	1.34247133(92)	1.3504	1.346	1.35106	1.361(99)	1.356
Jupiter	0.070(5))	0.6(3))	0.06237416(25)	0.0623	0.06300	0.0623142	0.0632(13)	0.06325
Saturn	0.014(2))	0.0105(50)	0.01366748(14)	0.0136	0.01383	0.0136394	0.0139(67)	0.01384
Uranus	*Missing*	*Missing*	0.002397742(70)	0.0024	0.002428	*Missing*	0.0024(19)	0.002427
Neptune	*Missing*	*Missing*	0.000777008(39)	0.0008	0.0007827	*Missing*	0.00079(55)	0.0007831

Table 2: This table shows the analytical uncertainties on the parameters of the eight Solar System planets. Notice how most of them is assumed to be the error from the last significant value.

Planet	T[day]	a[AU]	e[]	M[$M_\odot$]
Mercury	0.05	0.0005	0.0005	$3.6 \cdot 10^{-5}$
Venus	0.05	0.0005	0.0005	$3.6 \cdot 10^{-5}$
Earth	0.05	$1.1 \cdot 10^{-6}$	0.0005	$3.6 \cdot 10^{-5}$
Mars	0.05	0.0005	0.0005	$3.6 \cdot 10^{-5}$
Jupiter	0.5	0.0005	0.0005	$3.6 \cdot 10^{-5}$
Saturn	0.5	0.0005	0.0005	$3.6 \cdot 10^{-5}$
Uranus	0.5	0.005	0.0005	$3.6 \cdot 10^{-5}$
Neptune	50	0.005	0.0005	$3.6 \cdot 10^{-5}$

precise compared to the other planets.

The exoplanet 81 Cet b has the identical eccentricity of Mercury 3. It can be seen that the relative difference for the analytical and numerical values for both of them are exactly the same. This means that different sizes of orbits do not play a role in determining the relative difference between the analytical and numerical values when the eccentricity is the same. This makes sense, since it is only scaling up and down the numbers for the program.

A similar result can be seen from Kepler-83 c 3. It has identical eccentricity as Venus, and similar to the case of 81 Cet b and Mercury, the relative difference between the numerical and analytical is the same. This confirms that the β value is independent of the result regardless of eccentricity when the dimensional analysis method is used.

From the results, there is no difference occurring from low or high stellar mass or orbital period, as can be seen from b Cen AB, PSR J1719-1438 b, HAT-P-35 b, and VHS J125601.92-125723.9 b 3.

For LP 791-18 c, it also has a good agreement for the analytical and the numerical values 3 2. For HD 20782 b, the same can be said, but for other high eccentricity values,

Table 3: This table shows the numerical values of the periapsis precession in arcseconds per century for eight selected exoplanets, including their uncertainties. It is compared to the analytical calculations. Digits that come after the second significant digit of the uncertainty value are normally not included. The last name is VHS J125601.92-125723.9 b. It is written here to compress the table.

Planet	Analytical	Numerical
HD 20782 b	31.206(33)	31.1(26))
LP 791-18 c	1328.6167(36))	1346(54))
81 Cet b	1.47554(19))	1.49(55))
Kepler-83 c	94.28145(15))	95.5(38)
b Cen AB b	$9.3 \cdot 10^{-6}(18)$	$9.3 \cdot 10^{-6}(13892)$
PSR J1719-1438 b	4936362(0)	$4.97 \cdot 10^{6}(20)$
HAT-P-35 b	68206.21(49)	$6.94 \cdot 10^{4}(27)$
(name in desc)	$2.2 \cdot 10^{-7}(258)$	$2.2 \cdot 10^{-7}(2832)$

the difference is elevated to be slightly higher into a deviation of up to 4% 3 5. The increased error is due to major differences in the orbital speed and radius at apsis and periapsis, as explained in the numerical theory. It explains why the accuracy at high eccentricity values is lower than at low eccentricity values. The effect can be seen at the very right of the eccentricity image 3. The compensation value of "dtBalance" in the program makes this issue less profound, as it decreases the time step for very high eccentricities. If it's not added in, the difference between the analytical and numerical values will be slightly elevated into a deviation of up to 6% **??**. However, this compensation value makes very high eccentricity simulations take much longer to complete.

HD 20782 b show that the N periapsis is not at the vertical line, but slightly clockwise to it when the eccentricity of a planet is high 3. This shift in the N periapsis makes the program count the first orbit very early on. This is compensated for in the program. However, this slight shift of the periapsis will make the orbit become slightly

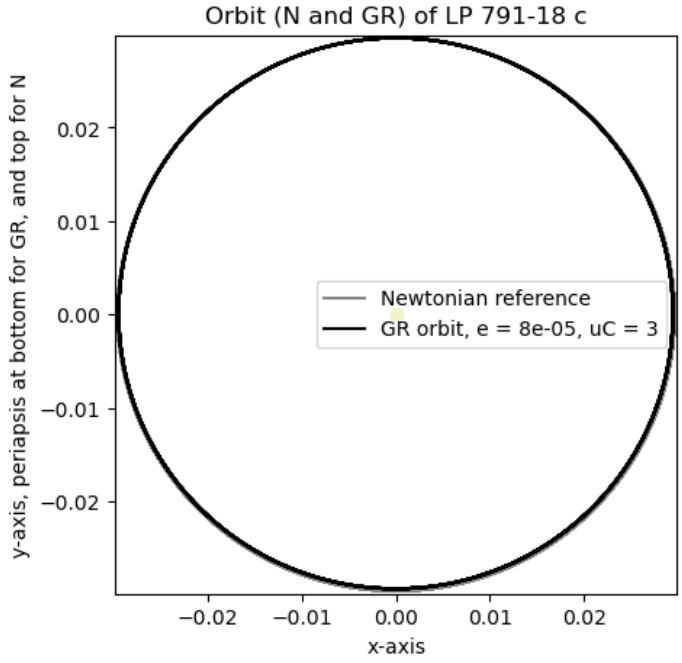

Figure 2: This image shows a low eccentricity planet called LP 791-18 c. The N and GR orbits are practically the same.

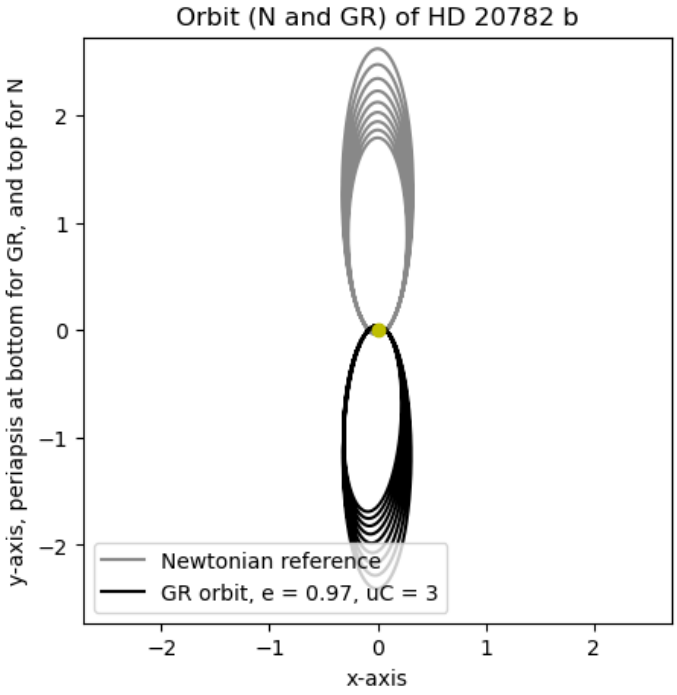

Figure 3: This image shows a high eccentricity planet called HD 20782 b. Notice that both N and GR orbits are shrinking. This effect is due to the repeated time step calculations.

shorter, which could explain why the numerical value is usually higher than the analytical, as shorter orbits give higher periapsis precessions.

From the image of HD 20782 b, its apsis is clearly shrinking with each orbit 3. This is an effect that comes from constantly recalculating the time step. The reason can likely be that the time step calculations to perform varying sizes of positional change can interfere with the way the Euler-Cromer method stores the orbital energy. It is likely, because it only happens for high eccentricity values, where the variation of the time steps are the largest. Nonetheless, it still gives a very good prediction of the periapsis precession, with a maximum deviation of around 4% .

The eccentricity image displays effects for short and middle sized eccentricities 5. For low eccentricity orbits, the line is jumping back and forth. This could be due to the program taking one more or one less iteration in finding the periapsis. the same cause is happening for the middle sized eccentricity values. But here, its not due to random fluctuations. This is due to the program taking longer to perform the simulation for a higher eccentricity, therefore taking more time steps. This can be seen from the line being bumped up when it goes too far down.

b Cen AB and HAT-P-35 b, with very large orbits, have uncertainties that exceed the numerically predicted values 3. This is due to the low periapsis precession values they have, meaning the program requires excellent precision if it should get an uncertainty that solves this issue. The program however, doesn't have this precision, since when the uncertainty is low enough, the absolute uncertainty becomes the dominant uncertainty, which can't be lowered. It means that the program is unable to achieve an uncertainty value that does not exceed the numerically predicted values when the planet's orbit is very large. However, the analytical and numerical values still agree with each other, like they do on all the other planets.

PSR J1719-1438 has zero analytical uncertainty 3. This is because all the three uncertainty values that have been gathered from NASA Exoplanet archive used in the program have been given zero uncertainty, and that's because the research team who got those values have not measured its uncertainty, or that it's too large to be measured accurately. This uncertainty is not realistic. The real uncertainty would be a non-zero value.

Some other contributions of uncertainties are also labeled as zero 4. These uncertainties lead to lower overall uncertainties of the periapsis precession values than what they should normally be, and might misrepresent their true values.

The program is only using the Δt to acquire the numerical uncertainty value, it does not include the uncertainties of the planet parameters. Therefore, the uncertainties of the program are lower than what a truly accurate uncertainty would be.

The program is using Euler-Cromer as the numerical method, which is only of order one. Other methods of numerically calculating the periapsis precession exist, for example an integration method [17], or using an orbital RK4 method[20]. These methods can lessen the variation

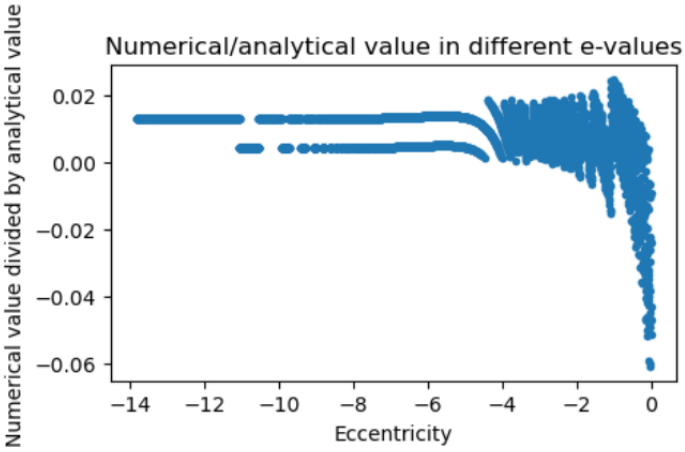

Figure 4: This image shows the logarithmic relation between the numerical value divided by the analytical value compared to different eccentricity values, when the time step does not vary with eccentricity. Notice how the small eccentricity jumps from one line to another, and the destabilization of the high eccentricity values. The max deviation is around 6%. The data was made from the program in Appendix D and E.

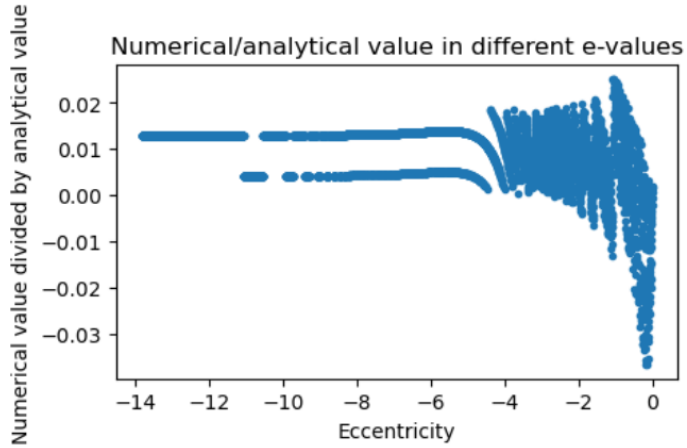

Figure 5: This image shows the logarithmic relation between the numerical value divided by the analytical value compared to different eccentricity values, when the time step does vary with eccentricity. Notice how it deviates less here, since the largest eccentricity values are given a much bigger time step. The max deviation is around 4%. The data was made from the program in Appendix D and E.

on the results, and give even more accurate predictions.

The results in the table takes in ten orbits, and an accuracy of 1000 as an input for the program **??** 3. Increasing more orbits per simulation and accuracy can increase the accuracy. However, the time required for the program to complete the orbits, and with the new accuracy are, from experience, approximately proportional to T^2. This means doubling the amount of orbits quadruples the time required, which at some point would take too long to complete.

There are other small assumptions included in the program. Firstly, the program assumes that the three parameters of orbital period, semi major axis, and solar mass, agree with each other in a way that putting two of the values in an equation will give the third value correctly. Secondly, the star is assumed to be stationary, as its mass far outweighs the mass of all other planets. However, if that was the case, all planets with radial velocity detection would not have been detected, which is false. However, the approximation works well when the star has a much larger mass than the planet, as then, the star's movement compared to the planet's orbit would be insignificant. Therefore, the interference of the approximation of the star being still is insignificant for the overall uncertainty of the periapsis precession value. Thirdly, the GR equation is based off of the Schwarzschild solution (1), another approximation. However, as mentioned in the mathematical theory, this is a good approximation, as stars with orbiting planets will have the approximate behaviour of what the Schwarzschild solution says their behaviour would be: being a massive static spherically symmetric object [7]. Lastly, the program uses an approximation of the GR acceleration equation. It doesn't have a major impact on the results, since the β value is divided by a high value, so not to make the better approximation of the GR acceleration equation become significant enough to interfere with the calculations.

Table 4: This table shows the analytical uncertainties on the parameters of the eight exoplanets. If it's labeled zero, there is no contributions to the uncertainty. It includes the reason the exoplanet is included in the analysis for study.

Reason	T[day]	a[AU]	e[]	M[$M_\odot$]
High e	2.8	0.2	0.01	0.07
Low e	$7.4 \cdot 10^{-6}$	$3.6 \cdot 10^{-4}$	$4 \cdot 10^{-5}$	0.01
Mercury e	8.8	0	0.029	0.03
Venus e	0.94	0.0032	0.0018	0.03
High a	$8.3 \cdot 10^{5}$	17.0	0	0.5
Low a, low T	$2 \cdot 10^{-9}$	0	0	0
High M	$2.1 \cdot 10^{-5}$	0	0.02	97.1
Low M, high T	$4 \cdot 10^{6}$	150	0.11	0

5. Future outlook

The most important future outlook is to find a way to observe the periapsis precession on the other planets; Uranus, Neptune, and the exoplanets. When that is done, the observational values can be compared to the analytical and numerical values, to give a test on GR for the periapsis precession.

The program is the first step of this field of study. It can be improved by other people, and there are many improvements to be made.

The Euler-Cromer method is of order one. An improvement of the program would be to implement a higher order method that conserves energy. Another improvement

would be to alter the numerical values slightly, in a way so it match better with the analytical values. This might be done by increasing or decreasing the time step. For the deviations between the analytical and numerical values caused by the high eccentricity, it might be solved by modifying the "dtBalance" compensation.

The approximations used everywhere can be modified to give even better approximations, or exact solutions. Those approximations are for example all the derivations to acquire the GR acceleration, Having the star stand still, using a finite time step, etc.

Periapsis precession with zero eccentricity can still happen. However, the program does not include these planets. One future step is to implement zero eccentricity periapsis precession simulations, to be able to study more planets. Note that no real planet has exactly zero eccentricity. However, NASA Exoplanet Archive label many planets to have this value, so this expansion is still useful.

Another outlook is to implement uncertainty of the planet parameters into the numerical simulation. This improves the true accuracy of the numerical uncertainty value of the periapsis precession.

Other developments not mentioned can be done. The program is open for anyone to expand upon, and to be used by anyone for any reason; examplewise for education, comparison, archiving, program improving, etc.

6. Conclusion

The analytical, observational, and numerical values of the periapsis precession values match well with each other; up to 4% error for certain high eccentricities, and very small errors for other eccentricity values. Therefore, the program is excellent at doing analytical and numerical predictions for exoplanet periapsis precessions, for when they will be compared against observational periapsis precessions when a method of observing it is realized.

The main issues of the program are the destabilization and wrong modelling orbits that come from high eccentricity orbits. The destabilization comes from the major differences of velocity and position at high eccentricity orbit. The shrinkage of the orbits is likely due to time step calculations interfering with the Euler-Cromer method. It is likely, because it only happens for high eccentricity values, where the variation of the time steps are the largest.

The next step for this program would be to compare it to exoplanetary observational values when such a method is realized in the future.

7. Acknowledgements

This research has made use of the NASA Exoplanet Archive, which is operated by the California Institute of Technology, under contract with the National Aeronautics and Space Administration under the Exoplanet Exploration Program.

The next part performs a "facility" function, as that is what NASA says should be done to reference them: Exoplanet Archive Exoplanet Archive

References

[1] Yeomans, Donald, J. Rahe, and R. Freitag: *The history of Comet Halley*. The Journal of the Royal Astronomical Society of Canada. Royal Astronomical Society of Canada, May 1986.

[2] pollock, Chris: *Mercury's Perihelion*, 2003. https://www.math.toronto.edu/~colliand/426_03/Papers03/C_Pollock.pdf, visited on (Accessed: 2024-01-29).

[3] Einstein, A.: *The Foundation of the General Theory of Relativity*. 1916.

[4] Holberg, J B: *Sirius B and the measurement of the gravitational redshift*. J. Hist. Astron., 41(1):41–64, February 2010.

[5] Dyson, F W, A S Eddington, and C Davidson: *IX. A determination of the deflection of light by the sun's gravitational field, from observations made at the total eclipse of May 29, 1919*. Philos. Trans. R. Soc. Lond., 220(571-581):291–333, jan 1920.

[6] Schatzer, Laro: *The Flaws of General Relativity*, 1997. https://aether.lbl.gov/www/classes/p139/speed/fgr.html, visited on (Accessed: 2024-04-26).

[7] Simpson, David: *A Mathematical Derivation of the General Relativistic Schwarzschild Metric*. East Tennessee State University, pages 1–51, apr 2007. https://www.etsu.edu/cas/math/documents/theses/simpson-thesis.pdf.

[8] Körber, C, I Hammer, J L Wynen, J Heuer, C Müller, and C Hanhart: *A primer to numerical simulations: the perihelion motion of Mercury*. Physics Education, 53(5):055007, jun 2018. https://dx.doi.org/10.1088/1361-6552/aac487.

[9] Aeronautics, National and Space Administration: *NASA Exoplanet Archive*, 2024. https://exoplanetarchive.ipac.caltech.edu/, visited on (Accessed: 2024-02-12).

[10] Dr. David R. Williams, National Aeronautics and Space Administration: *Planetary Fact Sheet - Metric*, 2024. https://nssdc.gsfc.nasa.gov/planetary/factsheet/, visited on (Accessed: 2024-03-08).

[11] Matloff, Greg and Harold Gerrish: *Chapter 3 - The scale of the problem: Interstellar distances, time, and energy considerations*. In Johnson, Les and Kenneth Roy (editors): *Interstellar Travel*, pages 51–82. Elsevier, 2023, ISBN 978-0-323-91360-7. https://www.sciencedirect.com/science/article/pii/B978032391360700001X.

[12] Dr. David R. Williams, National Aeronautics and Space Administration: *Sun Fact Sheet*, 2022. https://nssdc.gsfc.nasa.gov/planetary/factsheet/sunfact.html, visited on (Accessed: 2024-03-09).

[13] Casagrande, L. and Don A. VandenBerg: *Synthetic stellar photometry – I. General considerations and new transformations for broad-band systems*. Monthly Notices of the Royal Astronomical Society, 444(1):392–419, August 2014, ISSN 0035-8711. https://doi.org/10.1093/mnras/stu1476.

[14] Čotar, Klemen, Tomaž Zwitter, Gregor Traven, Janez Kos, Martin Asplund, Joss Bland-Hawthorn, Sven Buder, Valentina D'Orazi, Gayandhi M De Silva, Jane Lin, Sarah L Martell, Sanjib Sharma, Jeffrey D Simpson, Daniel B Zucker, Jonathan Horner, Geraint F Lewis, Thomas Nordlander, Yuan Sen Ting, Rob A Wittenmyer, and GALAH collaboration: *The GALAH survey: unresolved triple Sun-like stars discovered by the Gaia mission*. Monthly Notices of the Royal Astronomical Society, 487(2):2474–2490, May 2019, ISSN 0035-8711. https://doi.org/10.1093/mnras/stz1397.

[15] Fundamental Astronomy, Numerical Standards for: *Astronomical Constants*, 2014. https://web.archive.org/web/20131110215339/http://asa.usno.navy.mil/static/files/2014/Astronomical_Constants_2014.pdf, visited on (Accessed: 2024-04-24).

[16] Simon, J. L., P. Bretagnon, J. Chapront, M. Chapront-Touze, G. Francou, and J. Laskar: *Numerical expressions for precession*

formulae and mean elements for the Moon and the planets. , 282:663, February 1994.
[17] Pia Appelqust, Olof Nordenstorm: *Numerical Investigation of Relaticistic Perihelion Shift*, 2022. https://www.diva-portal.org/smash/get/diva2:1678945/FULLTEXT01.pdf, visited on (Accessed: 2024-04-12).
[18] D'Addio, Anna, Roberto Casadio, Andrea Giusti, and Mariafelicia De Laurentis: *Orbits in bootstrapped Newtonian gravity.* Phys. Rev. D, 105(10):104010, 2022.
[19] Kanzi, Sara: *Perihelion Precession in the Solar System*, 2016. http://i-rep.emu.edu.tr:8080/xmlui/bitstream/handle/11129/3615/kanzisara.pdf?sequence=1, visited on (Accessed: 2024-03-25).
[20] Voesenek, C.J.: *Implementing a Fourth Order Runge-Kutta Method for Orbit Simulation*, 2008. http://spiff.rit.edu/richmond/nbody/OrbitRungeKutta4.pdf, visited on (Accessed: 2024-04-21).

8. Appendix A: How to use the programs

Before calculating the numerical periapsis precession, a dataset of exoplanets needs to be acquired. The site to use is the NASA Exoplanet Archive. This is how to perform the steps necessary to run the code:

Step 1: Go to https://exoplanetarchive.ipac.caltech.edu/, and click at the first black box called "Confirmed Planets".

Step 2: Click on "Download Table", choose CSV Format, "Download All Columns", and "Download All Rows". Then click "Download Table".

Step 3: Copy and paste the program located in Appendix C. This program will sort through the archive, and write down the exoplanets that can be used to study the periapsis precession.

Step 4: Rename the downloaded file to "ExoplanetArchive.csv", or rename "open("ExoplanetArchive.csv", "rt") to the CSV type file that was downloaded in the previous step.

Step 5: Run the Appendix C program until it prints "Finished". A new file called "ExoplanetArchiveSorted.csv" will be made. This CSV file will be used for the main program.

Step 6: Open the new file, and find a planet to be studied, and the set of values from a research group to be chosen from.

Step 7: Copy and paste the main program located in Appendix B, put the planet in "ExoplanetName", and which set of values from a research group to be chosen from at "ExoplanetSelection".

Step 8: Choose the two settings: how many orbital turns the numerical simulation will do, and the accuracy of the time step.

Step 9: Run the Appendix B program, and collect the data from the print and plot.

The next section Explains how to retrieve a performance test to compare against analytical and numerical values with different eccentricities.

Step 1: Copy and paste the program located in Appendix D. Change the values of the eccentricity, time step, β, and the amount of simulated runs, if desired.

```
# PSR J1719-1438 b:          Lowest s
# HAT-P-35 b (3rd):          Highest
# VHS J125601.92-125723.9 b: Lowest s

ExoplanetName = "Mercury" # The plane
ExoplanetSelection = 1 # Select the n
turns = 10 # Amount of orbits of the
accuracy = 1000 # Accuracy to make th
```

Figure 6: This image shows where "ExoplanetName", "ExoplanetSelection", and the other two settings are.

```
Analytical value: 42.99730948352939
Analytical uncertainty: 1.229371017248731e-05
Numerical value: 43.46991593912609
Numerical uncertainty: 3.0619221352904242
contributions to numerical uncertainty: 3 / 3
Time uncertainty contribution: 0.05
Semi major axis uncertainty contribution: 0.0005
Eccentricity uncertainty contribution: 0.0005
Mass uncertainty contribution: 3.6e-05
```

Figure 7: This image shows The data printed for Mercury. The plot underneath the data is found at 1

Step 2: Run the Appendix D program. This program can take up to multiple days to fully complete, or stop prematurely due to the program flinging the planet away.

Step 3: Copy the printed values onto a file, and save the file as a .csv with a name.

Step 4: Copy and paste the program located in Appendix E. Rename the downloaded file to "EccentricityData.csv", or rename open("EccentricityData.csv", "rt") to the CSV type file that was downloaded in the previous step.

Step 5: Select the scale of the dataset, for example logarithmic, and add in enough data extractors from the "if i%2 == 1:" part.

Step 6: Run the Appendix E program, and collect the data from the plot
1

[1] Compared to Körber's paper [8], the base acceleration G * M is missing. This is because it is not needed, since it is always coupled with r^{-2}. Speaking of which, Körber's paper has an error in calculating the acceleration, as it fails to include the r^{-2}

9. Appendix B: Main program

```
# Importing numpy for data calculations, matplotlib for plotting, and csv for fetching exoplanet data
import numpy as np
import matplotlib.pyplot as plt
import csv

#Formulas from K rber et al.s paper is written with the help of (K rber et al. 2023)

# Exoplanets used for the report:
# HD 20782 b (5th):                Highest eccentricity
# LP 791-18 c:                     Lowest eccentricity
# 81 Cet b:                        Same eccentricity as Mercury
# Kepler-82 c:                     Same eccentricity as Venus
# b Cen AB b:                      Highest semi major axis
# PSR J1719-1438 b:                Lowest semi major axis, and lowest orbital period
# HAT-P-35 b (3rd):                Highest stellar mass
# VHS J125601.92-125723.9 b: Lowest stellar mass, and highest uncertainty contribution

ExoplanetName = "Mercury" # The planet/exoplanet to gain information on. Insert "Mercury" or select from the archive
ExoplanetSelection = 1 # Select the n-th result of the exoplanet with the name ExoplanetName from the archive
turns = 10 # Amount of orbits of the simulation.
accuracy = 1000 # Accuracy to make the time step smaller, and therefore more accurate.

# Mercury as standard selection
name = "Mercury"
T = 88.0 # Mercurian orbital period in day
a = 0.387 # Mercurian semi major axis in R0
e = 0.206 # Mercurian eccectricity
M = 1 # Solar mass
TDelta = 0.05 # T error
aDelta = 0.0005 # a error
eDelta = 0.0005 # e error
MDelta = 0.000036 # M error

# Initials at 0. N = Newtonian, GR = General Relativity
t = 0 # time
turnsN = 0 # N turns
turnsGR = 0 # GR turns
meanAngle = 0 # Mean perihelion precession angle
meanNvec = 0 # Mean of perihelion N
varianceN = 0 # Variance of perihelion N
varianceGR = 0 # Variance of precession GR
uncertaintyContribution = 0 # How many of the variables that contributed to the uncertainty

# Goes through the sorted exoplanet archive, searching for the specified exoplanet if it exists
print("Selecting exoplanet/planet")

try:
    with open("ExoplanetArchiveSorted.csv", "rt") as archive:
        iterate = csv.reader(archive)

        while True:
            iterating = next(iterate)
            if iterating[0] == ExoplanetName:
                break

        # Select the "ExoplanetSelection" result
        if ExoplanetSelection > 1:
            for i in range(ExoplanetSelection-1):
                iterating = next(iterate)

        # Collects the parameters of the exoplan
        name = iterating[0]
        T = float(iterating[11]) # Orbital perio
        a = float(iterating[15]) # Semi-major ax
        e = float(iterating[36]) # Eccectricity
        M = float(iterating[59]) # Stellar mass
        if not (iterating[12] == "" and iteratin
            TDelta = max(abs(float(iterating[12]
        else:
            TDelta = 0
        if not (iterating[16] == "" and iteratin
            aDelta = max(abs(float(iterating[16]
            uncertaintyContribution += 1
        else:
            aDelta = 0
        if not (iterating[37] == "" and iteratin
            eDelta = max(abs(float(iterating[37]
            uncertaintyContribution += 1
        else:
            eDelta = 0
        if not (iterating[60] == "" and iteratin
            MDelta = max(abs(float(iterating[60]
            uncertaintyContribution += 1
        else:
            MDelta = 0
except:
    pass

print("Exoplanet/planet selcted")

# Units and values for calculations.
R0 = 149597870700 # Average distance from the Ea
T0 = 86400 # Seconds in a day, often used in arc
M0 = 1.9891e30 # Mass of the Sun, often used in

# Constant conversions
uc = 299792458 # Speed of light
c = uc / R0 * T0 # c in R0**1 T0**-1 units
uG = 6.67430e-11 # Universal gravitational constant
G1 = uG / R0**3 * M0**1 * T0**2 # G in R0**3 M0**

# Orbital calculations
r = a * (1 - e) # Perihelion radius
aa = (1+e) # Semilatus rectum, used for an

# v**2 = GM(2/r-1/a). r(perihelion) = a * (1 - e
v = (G * M * (1 + e) / (1 - e) / a)**(1/2)
# Calculating the perihelion speed of the planet
rl = (r * v / c) # Specific angular momentum in

dtBalance = (1 - e**10) / (1 - 0.206**10) # Make
```

```
# (X)
dt = 2 * v * r**2 / (G * M) / accuracy * dtBalancer

dtN = dt # Timesteps for N
dtGR = dt # Timesteps for GR
beta = (r/rl)**2 / 248.939200199999405 # The effect value

#Defines the acceleration functions
# (X) without the vectors
def aN(x):
    return G * M / np.linalg.norm(x)**2 * -x / np.linalg.norm(x)
def aGR(x):
    return G * M * (1 + beta * (rl/np.linalg.norm(x))**2) / np.linalg.norm(x)

# We only have two orbital directions, since the angular momentum is always in the same direc
RvecN = np.array([[0, -r]])
VvecN = np.array([[-v, 0]])
RvecGR = np.array([[0, r]])
VvecGR = np.array([[v, 0]])

# Adds on the initial positions for N and GR, perihelionListGR be appended to perihelionListN
perihelionListN = RvecN
perihelionListGR = RvecGR

#Defines how the orbit will evolve over time, both for N and GR
def timeEvolution(RvecNOld, VvecNOld, RvecGROld, VvecGROld, dtN, dtGR):
    # Get acceleration magnitude from N and GR
    dtN = 2 * np.linalg.norm(VvecNOld) * np.linalg.norm(RvecNOld)**2 / (G * M) / accuracy * d
    dtGR = 2 * np.linalg.norm(VvecGROld) * np.linalg.norm(RvecGROld)**2 / (G * M) / accuracy *

    # (X) for whole iteration part excluding vectors
    AN = G * M / np.linalg.norm(RvecNOld)**2
    AGR = G * M * (1 + beta * (rl/np.linalg.norm(RvecGROld))**2) / np.linalg.norm(RvecGROld)**

    # Includes the acceleration directions. The minus sign is placed in front
    AvecN = -AN * RvecNOld / np.linalg.norm(RvecNOld)
    AvecGR = -AGR * RvecGROld / np.linalg.norm(RvecGROld)

    # Uses the calculated acceleration to get the new velocity and position
    VvecNNew = VvecNOld + AvecN * dtN
    VvecGRNew = VvecGROld + AvecGR * dtGR
    RvecNNew = RvecNOld + VvecNNew * dtN
    RvecGRNew = RvecGROld + VvecGRNew * dtGR

    return (RvecNNew, RvecGRNew, VvecNNew, VvecGRNew, dtN, dtGR)

# Goes through the calculations, and simulates for the planet
# (X) for non-vector part, and non N part
while turnsN < turns or turnsGR < turns:
    # Gets new values, and adds them onto the list
    RvecNNew, RvecGRNew, VvecNNew, VvecGRNew, dtN, dtGR = timeEvolution(RvecN[-1], VvecN[-1],
    RvecN = np.append(RvecN, [RvecNNew], axis=0)
    VvecN = np.append(VvecN, [VvecNNew], axis=0)
    RvecGR = np.append(RvecGR, [RvecGRNew], axis=0)
    VvecGR = np.append(VvecGR, [VvecGRNew], axis=0)

    # Check if orbital point is closer than the
    try:
        if np.linalg.norm(RvecN[-3]) > np.linalg
            turnsN += 1
            perihelionListN = np.append(perihelio
            print(f"{turnsN/turns*100}% done")
    except:
        pass
    try:
        if np.linalg.norm(RvecGR[-3]) > np.linalg
            turnsGR += 1
            perihelionListGR = np.append(perihelio
            print(f"{turnsGR/turns*100}% done")
    except:
        pass

    t = t + dt # Advance with one time step

# We add this, because the closest point accordin
perihelionListN = np.delete(perihelionListN, 0, a
perihelionListGR = np.delete(perihelionListGR, 0

# Checking if N or GR got extra turns, and remov
while len(perihelionListN) > len(perihelionListGR
    perihelionListN = np.delete(perihelionListN,
while len(perihelionListN) < len(perihelionListGR
    perihelionListGR = np.delete(perihelionListGR

# Angle of two vectors
def angle(v1, v2):
    return np.arccos(np.dot(v1, v2) / (np.linalg
# N gravitational acceleration with current posi
# Amount of points for the linear combination
length = (len(perihelionListGR) - 1) # Amount of
meanEstimation = perihelionListN[0] # Set the ini
convert = 180 / np.pi * 3600 / T * 365.24219
# N and GR list are the same lengths. Summing up
# (X) for vectors and parts
for i in range(length):
    meanAngle += angle(perihelionListGR[i], perih
    meanNvec += (perihelionListN[i+1] - meanEstim
meanN = np.linalg.norm(meanNvec)

# Calculates the variance of N and GR
# (X) for non-vector parts
for i in range(length): # for the planet
    varianceN += (meanN - np.linalg.norm(periheli
    varianceGR += (meanAngle - angle(perihelionLi

# Calculates the eccentricity and axis
aDelta = (((1-e) * aDelta)**2 + (-a * eDelta)**2
eDelta = (((1-e**2) * aDelta)**2 + (-2 * a * e *
aDelta = (((G * (1 + e) / (1 - e) / a)**(1/2) /
eDelta = (((v / c) * rDelta)**2 + ((r / c) * vD
```

```
dwDelta = ((6 * np.pi * (2 * rl / r**2) * rlDelta)**2 + (6 * np.pi * (-2 * rl**2 / r**3) * rD

# (X) for non-G and non-beta value fix
Uncertainty = (meanN**2 + varianceN + varianceGR)**(1/2) # Non-rescaled numerical periapsis p
dw = 2 * np.pi * 3 * (rl/p)**2 * angleConvert / G * G1 # Analytical periapsis precession valu
Precession = meanAngle * angleConvert / beta * 3 / G * G1 # Numerical Periapsis precession va
PrecessionDelta = Uncertainty * angleConvert / beta * 3 / G * G1 # Numerical periapsis preces

# Shows the analytical and numerical values for the precession, including the uncertainties, 
print(f"Analytical value: {dw}")
print(f"Analytical uncertainty: {dwDelta}")
print(f"Numerical value: {Precession}")
print(f"Numerical uncertainty: {PrecessionDelta}")
print(f"contributions to numerical uncertainty: {uncertaintyContribution} / 3")
print(f"Time uncertainty contribution: {TDelta}")
print(f"Semi major axis uncertainty contribution: {aDelta}")
print(f"Eccentricity uncertainty contribution: {eDelta}")
print(f"Mass uncertainty contribution: {MDelta}")

#Prints the N and GR trajectories for visual purposes. Gray is N, black is GR
aph = a * (1 + e) * 10.1 / 10 # Sets a value a bit larger than apsis, to be used for plot bou

plt.figure(figsize=(5,5))
plt.plot(RvecN[:, 0], RvecN[:, 1], color = '#888888', label = "Newtonian reference")
plt.plot(RvecGR[:, 0], RvecGR[:, 1], color = "#000000", label = f"GR orbit, e = {e}, uC = {un
plt.plot(0, 0, "yo")
plt.title(f"Orbit (N and GR) of {name}")
plt.xlabel("x-axis")
plt.ylabel("y-axis, periapsis at bottom for GR, and top for N")
plt.axis([-aph, aph, -aph, aph])
plt.legend()
plt.show()
```

10. Appendix C: Exoplanet data filter

```
# Importing numpy for data calculations, and csv for file manageme
import csv
import numpy as np

#Units for calculations
R0 = 149597870700 # Average distance from the Earth to the Sun (Astronomical Unit, AU), often
T0 = 86400 # Seconds in a day, often used in archives
M0 = 1.9891e30 # Mass of the Sun, often used in archives

uc = 299792458 # Speed of light
c = uc / R0 * T0 # c in R0**1 T0**-1 units
uG = 6.6743e-11 # Universal gravitational constant
G = uG / R0**3 * M0**1 * T0**2 # G in R0**3 M0**1 T0**-2 units

# Exoplanetary values:                    Source 9
# Values for the Solar System planets:    Source 10
# Planetary flux values:                  Source 11
# Values for the Sun:                     Source 12
# Solar magnitude value for 2MASS:        Source 13
# Solar magnitude value for Gaia:         Source 14
# Solar mass uncertainty:                 Source 15
# Earth semi major axis uncertainty:      Source 16

# Open the unedited file, and write on a new file where the requirements of the exoplanet are
with open("ExoplanetArchive.csv", "rt") as archive, open("ExoplanetArchiveSorted.csv", "w", n
    reader = csv.reader(archive)
    writer = csv.writer(sort)

    #Throws away the non-data part
    for i in range(97):
        throwAwayNonData = next(reader)

    #Adds on the eight planets of our Solar System:
    writer.writerow(['Mercury','Sun','1','1','8','Eye','Ancient times','Eye','Published Confir
    writer.writerow(['Venus','Sun','1','1','8','Eye','Ancient times','Eye','Published Confirme
    writer.writerow(['Earth','Sun','1','1','8','Eye','Ancient times','Eye','Published Confirme
    writer.writerow(['Mars','Sun','1','1','8','Eye','Ancient times','Eye','Published Confirmed
    writer.writerow(['Jupiter','Sun','1','1','8','Eye','Ancient times','Eye','Published Confir
    writer.writerow(['Saturn','Sun','1','1','8','Eye','Ancient times','Eye','Published Confirm
    writer.writerow(['Uranus','Sun','1','1','8','Eye and telescope','1781','A 6.2-inch reflect
    writer.writerow(['Neptune','Sun','1','1','8','Eye and telescope','1846','A 9-inch refracto

    #Iterates through each row; checks if each row has the requirements, and writes down the
    try:
        while True:
            # Checks requirements. 11 = orbital period, 15 = semi major axis, 36 = eccentricit

            row = next(reader)

            if row[11] == "" and row[15] == "" or row[11] == "" and row[59] == "" or row[15] =
                pass

            else:
                #Calculates missing data if needed, and writes all the data, calculated or no
                if row[11] == "":
                    row[11] = (2 * np.pi * (float
                if row[15] == "":
                    row[15] = (((float(row[11]))
                if row[59] == "":
                    row[59] = ((2 * np.pi * (floa
                writer.writerow(row)
        except:
            # Ends up here when the whole list is ite
            pass

        print("Finished")
```

11. Appendix D: Performance test

```
# Importing numpy for data calculations, matplotlib for plotting, and csv for reading the archive
import numpy as np
import matplotlib.pyplot as plt
import csv

#Formulas from K rber et al.s paper is written with the (X) symbol

# Exoplanets used for the report:
# HD 20782 b (5th):          Highest eccentricity
# LP 791-18 c:               Lowest eccentricity
# 81 Cet b:                  Same eccentricity as Mercury
# Kepler-82 c:               Same eccentricity as Venus
# b Cen AB b:                Highest semi major axis
# PSR J1719-1438 b:          Lowest semi major axis, and lowest orbital period
# HAT-P-35 b (3rd):          Highest stellar mass
# VHS J125601.92-125723.9 b: Lowest stellar mass, and highest orbital period

ExoplanetName = "HD 20782 b" # The planet/exoplanet to gain information on. Insert "Mercury" to use standard
ExoplanetSelection = 5 # Select the n-th result of the selected exoplanet from the archive
turns = 10 # Amount of orbits of the simulation.
accuracy = 1000 # Accuracy to make the time step smaller, and therefore more accurate.
runtimeAccuracy = False # Enable a * (1 - e) and the accuracy will automatically compensate for

# Mercury as standard selection
name = "Mercury"
T = 88.0 # Mercurian orbital period in day
a = 0.387 # Mercurian semi major axis in R0
e = 0.206 # Mercurian eccectricity
M = 1 # Solar mass
TDelta = 0.05 # T error
aDelta = 0.0005 # a error
eDelta = 0.0005 # e error
MDelta = 0.000036 # M error

# Goes through the sorted exoplanet archive, searching for the specified exoplanet if it exis
try:
    with open("ExoplanetArchiveSorted.csv", "rt") as archive:
        iterate = csv.reader(archive)

        while True:
            iterating = next(iterate)
            if iterating[0] == ExoplanetName:
                break

        # Select the "ExoplanetSelection" result of the selected exoplanet
        if ExoplanetSelection > 1:
            for i in range(ExoplanetSelection-1):
                iterating = next(iterate)

        # Collects the parameters of the exoplanet if it's found. The uncertainties are chose
        name = iterating[0]
        T = float(iterating[11]) # Orbital period in days
        a = float(iterating[15]) # Semi-major axis in R0
        e = float(iterating[36]) # Eccectricity
        M = float(iterating[59]) # Stellar mass
        if not (iterating[12] == "" and iteratin
            TDelta = max(abs(float(iterating[12]
        else:
            TDelta = 0
        if not (iterating[16] == "" and iteratin
            aDelta = max(abs(float(iterating[16]
            uncertaintyContribution += 1
        else:
            aDelta = 0
        if not (iterating[37] == "" and iteratin
            eDelta = max(abs(float(iterating[37]
            uncertaintyContribution += 1
        else:
            eDelta = 0
        if not (iterating[60] == "" and iteratin
            MDelta = max(abs(float(iterating[60]
            uncertaintyContribution += 1
        else:
            MDelta = 0
except:
    pass
accuracySave = accuracy # Saves the accuracy input to automatically compensate for

for calcy in range(5999): # Can be changed. Modifies how many times the progr

    # Initials at 0. N = Newtonian, GR = General
    t = 0 # Time
    turnsN = 0 # N turns
    turnsGR = 0 # GR turns
    meanAngle = 0 # Mean perihelion precession a
    meanNvec = 0 # Mean of perihelion N
    varianceN = 0 # Variance of perihelion N
    varianceGR = 0 # Variance of precession GR
    uncertaintyContribution = 0 # How many of the

    # Can be changed. Modifies the eccentricity
    e = 10**((calcy+1)/1000 - 6)

    accuracy = accuracySave
```

```
    # Adds on runtime accuract, if set at true
    if runtimeAccuracy == True:
        accuracy *= (1 - e)

    # Our units and values for calculations.
    R0 = 149597870700 # Average distance from the Earth to the Sun (Astronomical Unit, AU), o
    T0 = 86400 # Seconds in a day, often used in archives
    M0 = 1.9891e30 # Mass of the Sun, often used in archives

    # Constant conversions
    uc = 299792458 # Speed of light
    c = uc / R0 * T0 # c in R0**1 T0**-1 units
    uG = 6.6743e-11 # Universal gravitational constant
    G1 = uG / R0**3 * M0**1 * T0**2 # G in R0**3 M0**-1 T0**-2 units
    G = 1 # Einstein units

    # Orbital calculations
    r = a * (1 - e) # Perihelion radius
    p = a * (1-e**2) # Semilatus rectum, used for angular momentum

    # v**2 = GM(2/r-1/a). r(perihelion) = a * (1 - e)
    v = (G * M * (1 + e) / (1 - e) / a)**(1/2)
# Calculating the perihelion speed of the planet
    rl = (r * v / c) # Specific angular momentum in reduced units

    # Can be changed. Modifies the time step contribution from acceleration
    dtBalance = (1 - e**10) / (1 - 0.206**10) # Makes the new values of dt
    dtBalance = 1

    # (X)
    dt = 2 * v * r**2 / (G * M) / accuracy * dtBalance # Time step in reduced units

    dtN = dt # Timesteps for N
    dtGR = dt # Timesteps for GR

    # Can be changed. Modifies the beta value
    beta = (r/rl)**2 / 248.939200199999405 # The effect value of GR

    #Defines the acceleration functions
    # (X) without the vectors
    def aN(x):
        return G * M / np.linalg.norm(x)**2 * -x / np.linalg.norm(x) # Newtonian acceleration
    def aGR(x):
        return G * M * (1 + beta * (rl/np.linalg.norm(x))**2) / np.linalg.norm(x)**2 * -x / np

    # We only have two orbital directions, since
    RvecN = np.array([[0, -r]])
    VvecN = np.array([[-v, 0]])
    RvecGR = np.array([[0, r]])
    VvecGR = np.array([[v, 0]])
    # Adds in the initial positions for N and GR
    perihelionListN = RvecN
    perihelionListGR = RvecGR

    #Defines how the orbit will evolve over time
    def timeEvolution(RvecNOld, VvecNOld, RvecGR
        # Gravitation magnitude from N and G

        dtN = 2 * np.linalg.norm(VvecNOld) * np.
        dtGR = 2 * np.linalg.norm(VvecGROld) * np

        # (X) for calculations excluding
        AN = G * M / np.linalg.norm(RvecNOld)**2
        AGR = -beta * AN * 2 / (rl * np.linalg.n

        # Includes the acceleration directions.
        AvecN = AN * RvecNOld / np.linalg.norm(R
        AvecGR = -AGR * RvecGROld / np.linalg.nor

        # Uses the calculated acceleration to ge
        VvecNNew = VvecNOld + AvecN * dtN
        VvecGRNew = VvecGROld + AvecGR * dtGR
        RvecNNew = RvecNOld + VvecNNew * dtN
        RvecGRNew = RvecGROld + VvecGRNew * dtGR

        return (RvecNNew, RvecGRNew, VvecNNew, V

    # Goes through the calculations, and simulat
    # (X) for non-vector part, and non N part
    while turnsN < turns or turnsGR < turns: # Makes sure GR a
        # Gets new values, and adds them onto th
        RvecNNew, RvecGRNew, VvecNNew, VvecGRNew
        RvecN = np.append(RvecN, [RvecNNew], axis
        VvecN = np.append(VvecN, [VvecNNew], axis
        RvecGR = np.append(RvecGR, [RvecGRNew],
        VvecGR = np.append(VvecGR, [VvecGRNew],

        # Check if orbital point is closer than
        try: # value of GR. It's normally set at thre
            if np.linalg.norm(RvecN[-3]) > np.lin
                turnsN += 1
                perihelionListN = np.append(perih
        except:
            pass
        try:
            if np.linalg.norm(RvecGR[-3]) > np.li
                turnsGR += 1
                perihelionListGR = np.append(peri
        except:
            pass
```

```
    t = t + dt # Advance with one time step

# We add this, because the closest point according to the program is not the starting poi
perihelionListN = np.delete(perihelionListN, 0, axis = 0)
perihelionListGR = np.delete(perihelionListGR, 0, axis = 0)

# Checking if N or GR got extra turns, and removing the extras:
while len(perihelionListN) > len(perihelionListGR):
    perihelionListN = np.delete(perihelionListN, -1, axis = 0)

while len(perihelionListN) < len(perihelionListGR):
    perihelionListGR = np.delete(perihelionListGR, -1, axis = 0)

# Angle of two vectors
def angle(v1, v2):
    return np.arccos(np.dot(v1, v2) / (np.linalg.norm(v1) * np.linalg.norm(v2)))

# Initials and units for error calculations
length = (len(perihelionListGR) - 1) # Amount of orbits used for error calcuations
meanEstimate = (perihelionListN[0]) # Sets the periapsis point of N. It shifted, as explai
angleConvert = 180 / np.pi * 3600 / T * 365.242199 * 100 # Radians per orbit to arcsecond

# N and GR list are the same lengths. Summing up the angles, and summing up the N differer
# (X) for non-vector parts
for i in range(length):
    meanAngle += angle(perihelionListGR[i], perihelionListGR[i+1]) / (length)
    meanNvec += (perihelionListN[i+1] - meanEstimate) / (length)
meanN = np.linalg.norm(meanNvec)

# Gets the variance of N and GR
# (X) for non-vector parts
for i in range(length):
    varianceN += (meanN - np.linalg.norm(perihelionListN[i+1] - meanEstimate))**2 / (lengt
    varianceGR += (meanAngle - angle(perihelionListGR[i], perihelionListGR[i+1]))**2 / (le

# Calculates the numerical and analytical values, with uncertainty
rDelta = (((1-e) * aDelta)**2 + (-a * eDelta)**2)**(1/2) # Periapsis radius uncertainty
pDelta = (((1-e**2) * aDelta)**2 + (-2 * a * e * eDelta)**2)**(1/2) # Semilatus rectum un
vDelta = (((G * (1 + e) / (1 - e) / a)**(1/2) / (2 * M**(1/2))* MDelta)**2 + ((G * M * (1
rlDelta = (((v / c) * rDelta)**2 + ((r / c) * vDelta)**2)**(1/2)# Relativistic specific a
dwDelta = ((6 * np.pi * (2 * rl / r**2) * rlDelta)**2 + (6 * np.pi * (-2 * rl**2 / r**3) *

# (X) for non-G and non-beta value fix
Uncertainty = (meanN**2 + varianceN + varianceGR)**(1/2) # Non-rescaled numerical periaps
dw = 2 * np.pi * 3 * (rl/p)**2 * angleConvert / G * G1 # Analytical periapsis precession
Precession = meanAngle * angleConvert / beta * 3 / G * G1 # Numerical Periapsis precessior
PrecessionDelta = Uncertainty * angleConvert / beta * 3 / G * G1 # Numerical periapsis pr




# Can be changed. Modifies which desired data to priont out to be collected in the second
print(e)
print(Precession / dw)
```

12. Appendix E: Data plotter

```
# Importing numpy for data calculations, matplotlib for plotting, and csv for file management
import numpy as np
import matplotlib.pyplot as plt
import csv

Input = np.array([0])
Output = np.array([0])



# Can be changed. Insert the file to read the data from
try:
    with open("EccentricityData.csv", "rt") as archive:



        reader = csv.reader(archive)

        i = 0

        while True:

            i += 1
            data = next(reader)



            # Can be changed. Insert which data to be extracted. Don't forget to add more of
            if i%2 == 1:  #Input
                Input = np.append(Input, [np.log(float(data[0]))]) # Can be changed. Modifies
            if i%2 == 0:  #Output
                Output = np.append(Output, [np.log(float(data[0]))]) # Can be changed. Modifi

            # Some of the printed results from the previous code gives holes in the data. rem



except:
    pass

Input = np.delete(Input, 0)
Output = np.delete(Output, 0)

plt.figure(figsize=(5,5*9/16))
plt.plot(Input, Output, linestyle = "", marker = ".")
plt.title(f"Numerical/analytical value in different e-values")
plt.xlabel("Eccentricity")
plt.ylabel("Numerical value divided by analytica
plt.show()
```