MedCrave
Step into the World of Research

Physics & Astronomy International Journal

Review Article

Open Access

CrossMark

Describing Kepler Orbits with the Ulianov Orbital Model

Abstract

This paper presents the Ulianov Orbital Model (UOM), a simplified approach to two-body orbital mechanics. The UOM provides equations to calculate the standard ellipse parameters (a and b) and orbital trajectories and velocities from three UOM basic parameters (Ue, R0, and V0). It introduces a new kind of elliptical trigonometric functions, which simplify plotting orbital trajectories and their velocities over time and in elliptical angular steps. The Ulianov Elliptic Transform (UET), generates an impressive effect of rotating and scaling an ellipse, transferring its center from one of the foci to the geometric center of the ellipse. The UET offers a new and easy way to create and manipulate ellipses using both numerical and analytical methods.

Keywords: Ulianov Orbital Model, Elliptical trigonometric functions, Ulianov Elliptic Transform, Orbital trajectories

Volume 8 Issue 4 - 2024

Dr. Policarpo Yoshin Ulianov MSc, PhD
R&D Department, Power Opticks Tecnologia, Av. Luiz Boiteux Piazza, Brazil

Correspondence: Dr. Policarpo Yoshin Ulianov MSc PhD, R&D Department, Power Opticks Tecnologia, Av. Luiz Boiteux Piazza, Florian´opolis, 88056-000, SC, Brazil, Email poliyu77@gmail.com

Received: August 21, 2024 | **Published:** October 28, 2024

Introduction

In celestial mechanics, the problem of two bodies interacting gravitationally is traditionally described using Keplerian orbits or the Newtonian approach. A Kepler orbit, named after Johannes Kepler,[1] describes the motion of a body with small mass (M_a) relative to another body with a large mass (M_b). Due to the difference in masses, the movement of the body M_b in space is not significantly affected by the interaction with the body M_a . Therefore, M_b can often be considered stationary or defined as the origin of the coordinate system used to define the movement of M_a . This assumption defines the Kepler orbit problems and is valid for many cases, such as the movement of planets around the sun, communication satellites around the Earth, or small moons (such as the moons of Mars). However, it is not valid for the orbit of Earth's moon because the difference in masses is smaller, causing the moon's mass to make the Earth oscillate in its trajectory. When the mass M_a is much smaller than M_b , the orbit becomes an ellipse, parabola, or hyperbola. The Kepler Orbit Model (KOM)[2] requires six orbital elements to fully describe the motion of the body M_a :

- Eccentricity (e): The shape of the ellipse.
- Semi-major axis (a): Half the distance between the apoapsis and periapsis.
- Inclination (i): The tilt of the orbital plane.
- Longitude of the ascending node (Ω): The horizontal orientation of the ascending node.
- Argument of periapsis (ω): The orientation of the ellipse in the orbital plane.
- True anomaly (velocity ν ,at angle θ ,) at epoch (t_0): The position of the orbiting body along the ellipse at a specific time.

Note: Although these parameters are defined as six orbital elements, there are eight values listed, so we can also consider that the KOM has eight individual numeric parameters.

The Newtonian model,[3] which solves the problem using numerical methods, requires a similar number of parameters. A complete simulation can be defined by the masses of the bodies (M_b, M_a) and their initial positions (x, y, z) and velocities (v_x, v_y, v_z) and a time reference t_0 , totaling nine values. If we consider a reference system that defines the (x, y) plane over the elliptical plane, the numerical methods can use only seven parameters: masses M_b , M_a , and initial positions (x, y) and velocities (v_x, v_y) and a time reference t_0 .

The Ulianov Orbital Model introduces a new approach that reduces the complexity to only five parameters (seven numerical values) because the ellipse shape is represented by only one parameter, named the Ulianov Ellipse parameter (U_e), while maintaining accuracy in the ellipse representation and also defining parabolas and hyperbolas. Additionally, it offers a methodology that facilitates the orbit position and velocity calculation by applying two Ulianov Elliptic trigonometric functions ($cosuell(\alpha, U_e)$ and $sinuell(\alpha, U_e)$) to calculate the UOM elliptical orbit positions and velocities. The UOM also provides routines for determining these parameters from data observed in the body trajectory.

The Ulianov Orbital Model

The Ulianov Orbital Model (UOM) characterizes an orbit using the following five parameters:

1. Inclination (i): The vertical tilt of the ellipse with respect to the reference plane.
2. Longitude of the ascending node (Ω): The horizontal orientation of the ascending node.
3. Argument of periapsis (ellipse angle E_{ang}): The orientation of the ellipse in the orbital plane.
4. Initial condition, given by the minimum orbital distance (R_0) (the minimum distance between the orbital body and the central body), the maximum velocity (V_0) (the velocity at R_0 distance, which is the maximum velocity in the orbit) at epoch (t_0) (UTC time for an angle $\alpha = 0$, occurring at the point $(x_e, y_e) = (R_0, 0)$

and velocity $(v_x, v_y) = (0, V_0)$).

5. Ulianov elliptical parameter (U_e): Defines the shape and size of the orbit.

Although these parameters are defined as five orbital elements, there are seven values defined in this list, so we can also consider that the UOM has seven individual numeric parameters.

Note that the UOM provides a reduction of one parameter compared to the KOM because the parameters eccentricity (e) and semi-major axis (a) are replaced by only one parameter, the Ulianov Ellipse parameter (U_e), with some advantages:

- The application of elliptic trigonometric functions (cosuell(α, U_e) and sinuell(α, U_e)) to determine the orbit values and velocities as a function of any given time or ellipse angle.
- The easy obtaining of the elliptic orbit range (standard ellipse parameters a and b or eccentricity (e)) and period from initial values R_0 and V_0 or generic positions (e_x, e_y) and velocities (v_x, v_y) defined in the ellipse orbit.

Another important aspect is that the Keplerian orbital model stores an angular position in the orbit and a velocity for a specific time, which can be, for example, close to a present time of interest. This scheme is used because, normally, to move this point within the orbit, numerical simulations based on the Newtonian method are necessary, which must be calculated with a very small time interval dt, making it faster to calculate the orbit from the defined time to a new time for nearby times. In the case of the UOM, the nodel parameter stores the time, position, and velocity for the angle α equal to zero because the model can very quickly calculate the position and velocity for any desired time or angle.

The Ulianov Elliptical Parameter definition

The Ulianov Elliptical Parameter U_e value is calculated by the equation:

$$U_e = \frac{V_0^2 R_0}{GM_b} \tag{1}$$

where G is the gravitational constant and M_b is the mass of the primary body.

The Ulianov elliptical parameter eliminates the need to explicitly include M_b and G in the calculations, and also defines the velocity over the orbit and the complete elliptical orbit path, simplifying the model. However, to understand how the parameter U_e works, we need to first observe some basic definitions.

The general solution for the elliptical orbit is given by:

$$\begin{aligned} \frac{x^2}{a^2} &+ \frac{y^2}{b^2} = 1 \\ x_e(\alpha) &= a\cos(\alpha) \\ y_e(\alpha) &= b\sin(\alpha) \end{aligned}$$

where a and b are the semi-major and semi-minor axes of the ellipse, respectively. Note that the α angle is defined in the ellipse center, instead of being centered on the focus where the orbited body is located, which normally defines the origin of the system in both Cartesian and polar coordinates. In this way, these ellipse equations, despite being very simple, do not perform well in the case of a coordinate system centered on the orbited body. Additionally, the values a and b need to be calculated from some observed positions and velocities of the body in orbit. For certain initial velocities, a and b become very large and exhibit chaotic behavior, because very small changes in the initial orbital velocity (V_0) can generate large variations in the a and b values. This chaotic behavior difficult to calculate a and b values using analytics solutions of differential equations, requiring numerical simulation to obtain these parameters. These simulations are normally based on the calculation of forces and acceleration in a small time interval (in the order of fractions of a second) to be precise. For larger orbits, with periods of many years or even many centuries, a large number of processing steps are needed to determine complete orbit positions and velocities. To use an angle α centered in the ellipse focus, we can consider an ellipse equation defined as:

$$\begin{aligned} x_e(\alpha) &= R_0 K_x \cos(\alpha) - R_0(K_x - 1) \\ y_e(\alpha) &= R_0 K_y \sin(\alpha) \end{aligned}$$

where the ellipse's foci are in the x axis direction and for $a > b$, R_0 is the minimum orbital radius and K_x, K_y are gain factors that can be calculated by:

$$\begin{aligned} R_0 &= a - \sqrt{a^2 - b^2} \\ K_x &= \frac{a}{R_0} \\ K_y &= \frac{b}{R_0} \end{aligned}$$

Note: These definitions consider that the angle α starts at $0°$ and rotates counterclockwise. At $\alpha = 0$, $x_e = R_0$, $y_e = 0$, $v_x = 0$, and $v_y = V_0$.

The velocity $V(d_e)$ of body M_a along its trajectory is defined by the conservation of energy equation:

$$\begin{aligned} \frac{1}{2} M_a V(d_e)^2 &= \frac{1}{2} M_a V_0^2 - \frac{GM_b M_a}{R_0} + \frac{GM_b M_a}{d_e} \\ V(d_e)^2 &= V_0^2 - 2\frac{GM_b}{R_0} + 2\frac{GM_b}{d_e} \end{aligned} \tag{2}$$

Considering the Ulianov Elliptical Factor U_e defined in Equation (1), Equation (2) becomes the Ulianov orbital velocity equation:

$$\begin{aligned} d_e &= \sqrt{x_e^2 + y_e^2} \\ V(d_e)^2 &= V_0^2 \left(1 - \frac{2}{U_e}\left(\frac{R_0}{d_e} - 1\right)\right) \end{aligned} \tag{3}$$

Note that at this point we need two additional parameters to represent the elliptical orbit: The values of gains K_x, and K_y that not are included in the UOM parameter list presented at beginning of this section. Otherwise, as will be demonstrated in the next section, the U_e parameter allows the calculation of the K_x and K_y values, reducing this model to only five parameters (seven values in total), an new result that was obtained by applying the Ulianov Elliptical Transform.

Ulianov Elliptical Parameter and orbital types

Analyzing Equation (3), we can observe that the nature of the orbit depends on the value of U_e. Considering that in this equation the d_e value is equal to R_0, the $V(d_e)$ value is equal to V_0. Considering that the d_e value tends to infinity, Equation (3) can be defined as:

$$V(\infty) = V_0\sqrt{\left(1-\frac{2}{U_e}\right)} \quad (4)$$

In Equation (4), for $0 < U_e < 2$, we have the root of a negative number, which indicates that in this range of values the value of d_e will be limited and will never reach infinity. Therefore, this range of U_e values defines a closed curve, which is the ellipse.

For $U_e = 2$, the $V(\infty)$ value is equal to zero, marking the limit of the ellipse range, with the K_x parameter tending to infinity, which also defines a parabola.

For $U_e > 2$, the $V(\infty)$ value is greater than zero, and the U_e value defines a hyperbola.

Based on this analysis, the U_e parameter can be used to define a total of six types of orbits:

- $U_e = 0$: The body has a velocity V_0 pointing along the radial line, or $V_0 = 0$. This indicates that the M_a body is in a direct collision trajectory, defined by a straight line to the body M_b.
- $U_e = 1$: The trajectory is circular, representing that V_0 is equal to the orbital velocity.
- $0 < U_e < 1$: The trajectory is an ellipse, but the R_0 value is the maximum orbital radius and V_0 is the minimum velocity in the orbit.
- $1 < U_e < 2$: The trajectory is an ellipse, and the R_0 value is the minimum orbital radius and V_0 is the maximum velocity in the orbit.
- $U_e = 2$: The trajectory is a parabola, representing that V_0 is equal to the escape velocity.
- $U_e > 2$: The trajectory is a hyperbola.

The Ulianov Path Force

In the context of the Ulianov Gravitational Model (UGM),[4] which is aligned with the space-time distortion caused by the presence of matter as defined in Einstein's General Relativity Theory (GRT),[5,6] there are significant parallels and novel insights provided by the Ulianov theory.[7]

As shown in Figure 1-a, if the body M_a starts with $v_0 = 0$, the gravitational force (F_G) acts directly towards body M_b, causing body M_a to move in a straight line until collision. In this case, the UGM considers that the mass of body M_b reduces the Higgs Ulianov Perfect Liquid (HUPL) pressure,[8] generating a buoyancy force on body M_a which directs it towards the center of M_b where the HUPL pressure is zero.

Thus, with body M_a stationary, UGM generates a force that moves the body. However, the action of this force depends on pressure waves generated in M_b that are not instantaneous but travel at the speed of light, similar to the definition in GRT. Moreover, UGM defines that inertia does not move a body in a straight line but along a constant pressure path, which can be circular, elliptical, parabolic or hyperbolic.

Based on the pressure conservation law (combining dynamic pressure generated by body M_a's force with a magnitude equal to the gravitational force. In the case of a circular orbit, these two forces cancel each other. According to UGM, this centrifugal force arises whenever the body crosses equipressure paths. If the body's movement is exactly perpendicular to the spherical shell defining the constant pressure line, the centrifugal force will be spread in a plane around the body. This can be observed in the analogy presented in Figure 1-b, where a sphere is placed at the top of a cylindrical surface (with a parabolic cross-section and a straight line at the top of the surface) that is slightly inclined, causing the ball to move in a straight line and accelerate. In this analogy, it's as if there are two equal centrifugal forces pulling the ball to both sides simultaneously, but this is an extremely unstable equilibrium because a minimal deviation from the trajectory will cause the ball to fall off the top of the surface.

Thus, the straight-line trajectory shown in Figure 1-a is similar to the body traveling along the top of the cylindrical surface. From Newton's mechanics viewpoint, this can occur because only F_G is considered, and it acts in a straight line. However, from GRT's perspective, this straight line can be distorted by spacetime curvature. Therefore, UGM predicts that the ball will fall off the top of the cylindrical surface, meaning a minimal deviation from the straight line will cause the centrifugal force to act in a specific direction (perpendicular to F_G) with a magnitude equal to F_G.

The vector sum of F_G and F_C creates the Ulianov Path Force (F_P), which is a force that changes the velocity vector's direction, attempting to make the body follow a constant pressure path compatible with its current velocity.

Note that the use of F_P eliminates F_G and F_C (i.e., F_P incorporates the combined effect of gravitational force interacting with centrifugal force), aligning with GRT models where F_G also doesn't exist, and the body is moved by inertia along geodesic lines.

Thus, in UGM, F_P aligns with a concept of force that is "behind" inertia, and in the presence of pressure variation paths in HUPL, F_P makes the body follow constant pressure lines as if it were following a straight line, similar to how a ping-pong ball (massless and with volume) placed inside a circular glass tube follows the water flow, moving with the liquid in a circular trajectory, without hitting the duct walls or being subjected to any additional force.

In summary, we can observe Figure 1-c, which shows the behavior of a body M_a with zero initial velocity, attracted straight to the lowest pressure point in HUPL. However, as it moves, a minimal trajectory variation (generated for example, by quantum fluctuations) causes the centrifugal force to take a random direction perpendicular to F_G, generating F_P, which tries to make the body enter a constant pressure trajectory.

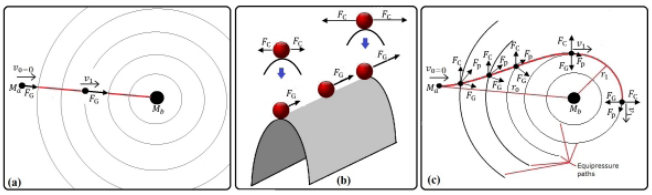

Figure 1

Figure 1 Ulianov Path Force in the two bodies problem. a) Newtonian model: A small body M_a with initial velocity equal to zero is attracted by the large mass of body M_b in a straight line. As there is only the action of gravitational force, the body M_a collides with M_b. b) An analogy where a sphere goes in a straight line at the top of a gently inclined cylindrical surface (with a parabolic cross-section and a straight line at the top of the surface), pulled by gravitational force. In the Ulianov gravitational model, passing through equipressure

paths generates centrifugal forces with a magnitude equal to the gravitational force. c) Ulianov gravitational model: A minimal random deviation is enough for the centrifugal force equilibrium to "collapse" and act to one side, generating a Ulianov path force that deflects the straight trajectory and takes the body M_a into a circular or elliptical orbit.

Thus, F_P arises mainly when constant pressure lines are crossed by the body, deflecting the body laterally until it finally assumes a circular or elliptical trajectory instead of colliding with body M_b as predicted in the Newtonian model for this case.

In a simple analogy, this is like a cyclist who is descending a mountain on an inclined trail, with the bicycle pointed downwards and gaining speed due to the force of gravity that acts in the same direction as the bicycle's displacement vector, performing work that is converted into kinetic energy. Then, the cyclist finds a narrow path that goes around the mountain at the same height and directs the bike towards this new path, leaving the inclined path. The force of gravity will continue to act on the cyclist and its bicycle, pulling him down to the trail soil, but now the force will be perpendicular to the displacement vector and will not perform work, and will not transfer energy to the bicycle or increase its speed. In this analogy, the Ulianov path force F_P appears when the bicycle crosses the lateral paths (equipressure paths in the M_a real case) and keeps trying to divert the bicycle so that it enters one of these paths (keeps trying that M_a follow an elliptic orbit). In this case, if the bicycle crosses parallel paths at an angle of exactly 90 degrees, the cyclist could not choose either side or another to deviate the bicycle, which generates a straight path defined in the Figure 1-a. But a very small deviation in this angle (the ball falling off the cylinder top analogy) is enough to generate a decision with the bicycle deviating to one side, which generates the orbital path defined in Figure 1-c.

Numerical simulations

The Ulianov Orbital Model was applied as the simplest example of a two-body problem in the context of the Ulianov Gravitational Model, aiming to replace the concept of gravitational force with the concept of Ulianov path force. To do this, some programs were developed in Python to perform two types of numerical simulations:

- A simulation considering the traditional Newtonian model, with calculations of gravitational forces, accelerations, speeds, and displacements.
- Another simulation considering the Ulianov path force, without applying gravitational forces or acceleration.

Table 1 presents the Python code that implements the numeric Newtonian gravitational force procedure and the Elliptic Ulianov Transformation procedure. The Newtonian calculation is a standard procedure that considers a small time interval (dt), calculates the gravitational force on body M_a in two components (x, y), its acceleration, and updates the velocities and positions. This procedure is easy to implement and generates very good results but with cumulative error (in velocity and position) that depends on the value of dt used.

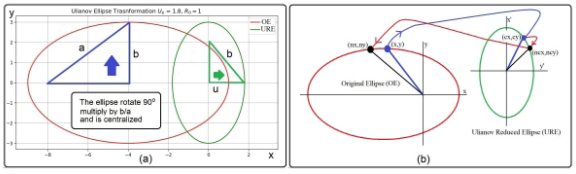

Figure 2

Figure 2 The base of Ulianov Elliptic Transform: a) An Original Ellipse (OE) defined by a and b parameters (or R_0 and U_e parameters) is transformed into the Ulianov Reduced Ellipse (URE), which is proportional (multiplied by a b/a factor), rotated 90°, and centralized. b) Numeric procedure: From a point (x, y) in the original ellipse, a point (cx, cy) is defined in the URE. Since this point is centralized, the ellipse can be treated as if it were a circle, where a small angular displacement can be generated, leading to a new point (ncx, ncy) within the URE, which is then converted back, generating the next position (nx, ny) on the OE.

Table 1 Python code of numeric Newtonian gravitational force procedure and Ulianov Elliptic Transform procedure

Gravitational force calculation	Ulianov elliptic transform
# Gravitational force calculation: Fg = G * M1 * M2 / d**2 Fg_x = -Fg * dx Fg_y = -Fg * dy *# Calculate acceleration:* ax = Fg_x / M2 ay = Fg_y / M2 *# Update speed:* vx = vx + ax * dt vy = vy + ay * dt vm = np.sqrt(vx**2 + vy**2) *# Update position:* x += vx * dt + 0.5 * ax * dt**2 y += vy * dt + 0.5 * ay * dt**2	*# Calc. radius and theoretical speed:* d = np.sqrt(x**2 + y**2) vteo = V0 * np.sqrt(1 + (2 / Ue) * (R0 / d - 1)) *# Apply the Elliptic Ulianov Transform:* cy = y + d* Ue cx = x - d* Ue de = np.sqrt(cx**2 + cy**2) *# Calculate the current angle:* angle = np.arctan2(cy, cx) *# Angular increment proportional to speed:* dang = vteo * dt / (2 * np.pi * d) angle += dang *# Update position* ncy = de * np.cos(angle) ncx = de * np.sin(angle) *# Inverse Ulianov Elliptic Transform* *# Return to the original ellipse:* nx = ncx + d * Ue ny = ncy - d* Ue *# Calculate the speed obtained:* vxn = (xn - x) / dt vyn = (yn - y) / dt vmn = np.sqrt(vxn**2 + vyn**2) *# Pass to the theoretical speed value:* vx = vxn / vmn * vteo vy = vyn / vmn * vteo *# Update position without use acceleration:* x += vx * dt y += vy * dt

The Ulianov Elliptic Transform (UET), as presented in Figure 2, converts a given original ellipse, defined by a and b parameters (or R_0 and U_e parameters), into the Ulianov Reduced Ellipse (URE), which is proportional (multiplied by a b/a factor), rotated 90°, and centralized. In this way, the UET numerical procedure converts a known point (x, y) on the original ellipse (which is centered on one of the focuses) to a point (cx, cy) on the URE (which is centered). It generates a small angle of rotation (based on the dt value and the theoretical speed), defining a new point (ncx, ncy) within the URE. An Ulianov Elliptic inverse transform is applied, defining the new point (nx, ny) on the original ellipse associated with the M_a displacement, in the time interval dt, but without considering acceleration.

These two numeric procedures were used to calculate the trajectory of the body M_a from the values of M_b, R_0, and V_0 (h also allows the calculation of U_e which replaces the value of GM_b in the numerical UET method). Several simulations were carried out, and it became clear that once the values of M_a and R_0 were defined, this also defined an escape velocity and an orbital velocity (for example,

for M_a equal to the mass of the Earth and $R_0 = 10^8$ m , the resulting values are $v_{orb} = 2003$ m/s and $v_{escape} = 2834$ m/s). Thus, by defining V_0 equal to v_{orb} , a circular orbit is generated, and for a V_0 value below the escape velocity, an ellipse is generated. By varying V_0 within this range, the values U_e ranging between 1 and 2 were obtained. Using the numerical procedures to traverse a complete orbit, we obtained the K_y and K_x values that define the parameters a and b of the ellipse, making it clear that from a set of values R_0 and U_e , a unique value of K_y and also of K_x is defined.

Despite this, searching on the internet and with the support of Artificial Intelligence Chat GPT-4, it was not possible to identify a function that, starting from the values of M_a , M_b , V_0 , and R_0 , would directly generate the values of K_y and K_x (or even the parameters a and b) associated with the orbital ellipse generated by the numerical simulation. The information obtained on the internet and confirmed by Chat GPT-4 indicates that the only way to observe how far the use of a velocity value V_0 would take the orbit length could not be obtained directly, even in the simple case of two bodies, because the Newtonian differential equations that define this problem also need to be solved numerically. For values of V_0 that approach the escape velocity, the size of the orbit increases significantly and tends to infinity if V_0 is equal to the escape velocity. Thus, very small variations in the V_0 value (close to the escape velocity) generate very large variations in the final size of the orbit, characterizing a chaotic system (where small variations in initial conditions generate large changes in the system's final state) that are not well represented through differential analytical equations.

One aspect that can be observed is that in these simulations, low speeds (for example, 2500 m/s) generated orbit times of a few days (for example, 10 days) which were quickly resolved by the Newtonian method (in about an hour for a dt of 0.01 s). However, when using higher speeds close to the escape velocity, the size of the orbit grows significantly, and the simulation time becomes prohibitive to carry out on a personal computer. An alternative then was to increase the value of dt , but as the Newtonian method operates with an acceleration value multiplying the interval time value squared, for a larger dt , the errors become extremely high.

The UET method, in turn, does not use acceleration and does not treat dt squared, allowing time interval values a thousand times larger (and even ten thousand times larger) without a significant increase in errors. This result can be seen in Table 2, where the Newtonian method is compared to the Ulianov Elliptic Transform method. This result was obtained for one individual case of K_x orbital parameter calculation from a given set of values (M_b , R_0 , and V_0) applying both methods with different values of dt (the Newtonian method with $dt = 0.01$ was used as a reference). As expected, the Ulianov method shows almost no variation in error as the time interval increases, but it is necessary to use small time intervals close to points of interest (large angles are used to traverse the ellipse, and small angles are used at the extreme points where the value of K_x is calculated). Thus, the comparison must be made using the number of processing steps.

For the same error value, the Ulianov method proved to be much faster (in the order of 1 to 8 thousand times faster, as this varies depending on the total orbit time), which is easy to understand because the UET routine can traverse an ellipse with low errors in angular increments of 0.1° , allowing the complete orbit to be traversed in just 3600 steps. In an analogy, the UET routine is like drawing a circular orbit using sine and cosine functions, considering that, for example, this orbit takes 36 hours. In this case, we can use an angular interval of 10 degrees and calculate all the positions (with only 36 points, one point per hour), and the error will be the same as that obtained by calculating the position every second.

Table 2 Comparison of error and computational cost between Newtonian and Ulianov methods

Method	dt (s)	Steps	K_x	%Error
Newtonian	0.1	17,280,000	1.905236	0.0074
Newtonian	1	1,728,000	1.905162	0.0113
Newtonian	10	172,800	1.904418	0.0483
Newtonian	100	17,280	1.896986	0.4614
Ulianov	10	19,686	1.905437	0.0031
Ulianov	100	3,903	1.907121	0.0109
Ulianov	1000	2,353	1.921555	0.0085

Although this method has only been tested for a very simple case, with the parameter U_e that defines the shape of the ellipse being known (or calculated in some way), numerically using the elliptical Ulianov transform presented in the table 1 works both in the case of ellipses, parabolas, and hyperbolas with low position errors, even in the case of dt values a thousand times larger (for example $dt = 0.01$ to $dt = 0.1$ in the Newtonian method generating the same error as $dt = 100$ to $dt = 1000$ in the Ulianov method), due to the fact that it does not use acceleration values (without having factors multiplying dt^2). In practice, this can mean that a problem that would take an entire month to calculate on a PC can be calculated in less than an hour with the same level of error using the Ulianov method. Therefore, the application of this numerical method to more general cases involving more bodies is something to be studied in the future.

Calculating the K_x and K_y values

For an elliptical orbit defined by parameters R_0 , V_0 , and U_e , the equation of a standard ellipse E can be defined:

$$\begin{aligned} e_x &= R_0 \cdot K_x \cdot \cos(\alpha) - R_0 \cdot (K_x - 1) \\ e_y &= R_0 \cdot K_y \cdot \sin(\alpha) \end{aligned} \tag{5,6}$$

Applying the Ulianov Elliptical Transform:

$$\begin{aligned} d_e &= \sqrt{x_e^2 + y_e^2} \\ U_y &= y_e + 2 \cdot d_e (1 - U_e) \\ U_x &= x_e + 3 \cdot d_e (U_e + 1) \end{aligned}$$

The equations (5) and (6) define the Ulianov Reduced Ellipse equation:

$$\begin{aligned} U_x &= R_0 \cdot U_e \cdot \sin(\alpha) \\ U_y &= R_0 \cdot K_y \cdot \cos(\alpha) \end{aligned}$$

As the ellipse E is proportional to ellipse URE, these relationships can be defined:

$$\begin{aligned} \frac{R_0 \cdot U_e}{R_0 \cdot K_y} &= \frac{R_0 \cdot K_y}{R_0 \cdot K_x} \\ K_y^2 &= K_x U_e \end{aligned} \tag{7}$$

In the standard ellipse, we can define the velocity $V(d)$ as a function of the angle α , considering that the distance value is given

as a function of α $\left(d = d(\alpha)\right)$:

$$V(\alpha)^2 = V_0^2\left(1 - \frac{2}{U_e}\left(\frac{R_0}{d(\alpha)} - 1\right)\right) \tag{8}$$

Considering the value of α in degrees:

- For $\alpha = 0^\circ$, $d\left(0^\circ\right) = R_0$ and $V\left(0^\circ\right)$ is the maximum velocity value $\left(V\left(0^\circ\right) = V_0\right)$;
- For $\alpha = 90^\circ$, $d\left(90^\circ\right) = K_y \cdot R_0$ and $V\left(90^\circ\right)$ is a medium velocity value;
- For $\alpha = 180^\circ$, $d\left(180^\circ\right) = K_x \cdot R_0$ and $V\left(180^\circ\right)$ is a minimum velocity value;

By applying the Ulianov Elliptic Transform, we can simultaneously trace the trajectory of the standard ellipse in space with a real displacement and speed and obtain the drawing of the Ulianov Reduced Ellipse (URE). In this case, some interesting points can be observed:

- The total travel time of the two ellipses will be equal;
- Considering an angle defined at the central point of the ellipse, the angular velocity will be equal in both ellipses;
- The URE will be multiplied by a size reduction factor (or scale factor) equal to $\frac{K_y}{K_x}$;
- As the angular velocity is the same, if the scale factor is considered, the velocity in the URE will be the same as in the standard ellipse.

In this way, the $V\left(90^\circ\right)$ value can be obtained considering $d\left(90^\circ\right)$ in the standard ellipse, multiplied by the scale factor:

$$\begin{aligned} d_{URE}\left(90^\circ\right) &= d\left(90^\circ\right)\frac{K_y}{K_x} \\ d_{URE}\left(90^\circ\right) &= K_y R_0 \frac{K_y}{K_x} \\ d_{URE}\left(90^\circ\right) &= R_0 \frac{K_y^2}{K_x} \end{aligned} \tag{9}$$

Applying Equation (7) in Equation (9):

$$d_{URE}\left(90^\circ\right) = \frac{R_0}{U_e} \tag{10}$$

Applying Equation (10) in Equation (8):

$$V_{URE}\left(90^\circ\right)^2 = V_0^2\left(1 - \frac{2}{U_e}\left(\frac{R_0 U_e}{R_0}\right) + \frac{2}{U_e}\right) \tag{11}$$

As the scale factor was applied in Equation (11), the velocity in the standard ellipse is the same:

$$V\left(90^\circ\right)^2 = V_{URE}\left(90^\circ\right)^2 = V_0^2\left(\frac{2}{U_e} - 1\right) \tag{12}$$

In the standard ellipse, the conservation of angular momentum $\left(L = M \cdot V \cdot d = \text{constant}\right)$ can be applied. In this way, we can compare the angular momentum at $\alpha = 0^\circ$ $\left(d\left(0^\circ\right) = R_0\right.$ and velocity $\left.V\left(0^\circ\right) = V_0\right)$ to the momentum at $\alpha = 90^\circ$ $\left(d\left(90^\circ\right) = R_0 \cdot K_y\right.$ and velocity $\left.V\left(90^\circ\right)\right.$ given by Equation (12), defining the relation:

$$L = M_2 \cdot V_0 \cdot R_0 = M_2 \cdot V\left(90^\circ\right) \cdot \left(R_0 \cdot K_y\right) \tag{13}$$

Simplifying and squaring Equation (13), we get:

$$V_0^2 = V\left(90^\circ\right)^2 \cdot K_y^2 \tag{14}$$

Applying Equation (8) in Equation (14):

$$V_0^2 = V_0^2\left(\frac{2}{U_e} - 1\right) \cdot K_y^2$$

Isolating K_y, we get:

$$K_y = \frac{1}{\sqrt{\frac{2}{U_e} - 1}} \tag{15}$$

Applying Equation (15) in Equation (17):

$$K_x = \frac{1}{2 - U_e} \tag{16}$$

The K_x and K_y values calculated by Equations (16) and (15) were compared with the values of K_x and K_y generated by numerical simulations, and the same result was obtained, demonstrating the validity of these two equations.

Maximum Orbital Velocity, Orbital Ellipse Parameters, and Orbital Period

The deduction of K_x and K_y values presented in the previous section allows the definition of a new relation between the standard elliptical parameters: a and b, the Ulianov Ellipse parameter: U_e, and the basic parameters that define the orbit: R_0, V_0, and $G \cdot M$. These can be expressed by the following equations:

$$\begin{aligned} R_0 &= a - \sqrt{a^2 - b^2} \\ U_e &= \frac{b^2}{a^2 - \sqrt{a^4 - a^2 b^2}} \\ U_e &= \frac{R_0 \cdot G \cdot M}{V_0^2} \end{aligned} \tag{17,18,19}$$

These equations can define:

$$\begin{aligned} \frac{G \cdot M}{V_0^2} &= \frac{R_0}{U_e} = \frac{\left(a^2 - \sqrt{a^4 - a^2 b^2}\right)\left(a - \sqrt{a^2 - b^2}\right)}{b^2} \\ \frac{G \cdot M}{V_0^2} &= \frac{R_0}{U_e} = \frac{a^3\left(2\left(1 - \sqrt{1 - b^2/a^2}\right) - b^2/a^2\right)}{b^2} \end{aligned} \tag{20}$$

Equation (20) is named the Ulianov Maximum Orbital Velocity Ellipse Parameters Relation. This equation implies that for a given body M, the maximum orbital velocity V_0 will define a unique ellipse shape (represented by parameters a and b). Despite ellipses being known for more than 2000 years and elliptical orbits being known for more than 300 years, this equation had not been found by mathematicians. This author believes that this type of equation is not

just a mathematical curiosity but represents a key that can lead to, for example, an equation that directly calculates the length of an ellipse and, as presented in the following sections, provides a way to obtain the Kepler orbital period equation.

Applying Equations (16) and (15), we can also calculate the parameters a and b using the following equations:

$$\begin{aligned} a &= \frac{R_0}{2-U_e} \\ b &= \frac{R_0}{\sqrt{\frac{2}{U_e}-1}} \\ e &= \sqrt{1-\frac{b^2}{a^2}} \end{aligned} \qquad (21,22,23)$$

These equations allow the conversion from standard ellipse parameters a and b and value of eccentricity e (used in the Kepler orbital model), to the Ulianov ellipse parameters R_0 and U_e. As R_0 can be seen as a scale factor, the U_e value defines the ellipse shape (including ellipses, parabolas, and hyperbolas), providing a natural way to deal with elliptical orbits.

This model also allows the calculation of the orbital period (valid for the ellipse case) based on the value V_0. Considering a circle with radius R_0K_x, in the Ulianov orbital model, the orbital period T_{orbit} is given by the circumference of this circle divided by the mean velocity $V(90^\circ)$ used to obtain the K_x value. In this way, the orbital period can be calculated by:

$$T_{orbit} = \frac{2\pi \cdot R_0 \cdot K_x}{V(90^\circ)} \qquad (24)$$

Applying Equations (16) and (12) in Equation (24):

$$\begin{aligned} T_{orbit} &= \frac{2\pi R_0}{(2-U_e)} \cdot \frac{1}{V_0\sqrt{\frac{2}{U_e}-1}} \\ T_{orbit} &= \frac{2\pi}{V_0} \cdot \frac{R_0}{(2-U_e)\cdot\sqrt{\frac{2}{U_e}-1}} \end{aligned} \qquad (25)$$

Applying Equations (21) and (22) in Equation (25), the orbital period is obtained from the standard ellipse parameters:

$$T_{orbit} = \frac{2\pi}{V_0} \cdot \frac{b}{\sqrt{2\left(1-\sqrt{1-\frac{b^2}{a^2}}\right)-\frac{b^2}{a^2}}} \qquad (26)$$

Equations (25) and (26) are the Ulianov Orbital Period Equations and provide an easy and direct way to calculate the orbital period based on the maximum orbital velocity V_0 and Ulianov orbital parameters (or standard ellipse a and b parameters).

Note that we can combine Equations (26) and (20) to eliminate the V_0 value. Isolating V_0^2 in Equation (20):

$$V_0^2 = \frac{G \cdot M \cdot b^2}{a^3\left(2\left(1-\sqrt{1-b^2/a^2}\right)-b^2/a^2\right)} \qquad (27)$$

Applying Equation (27) in Equation (26):

$$T_{orbit} = \frac{2\pi}{\sqrt{\frac{G \cdot M \cdot b^2}{a^3\left(2\left(1-\sqrt{1-b^2/a^2}\right)-b^2/a^2\right)}}} \cdot \frac{b}{\sqrt{2\left(1-\sqrt{1-b^2/a^2}\right)-b^2/a^2}} \qquad (28)$$

As the fraction inside the square root is the same in the numerator and denominator, we can simplify Equation (28) to:

$$\begin{aligned} T_{orbit} &= \frac{2\pi \cdot b}{\sqrt{\frac{b^2 \cdot G \cdot M}{a^3}}} \\ T_{orbit} &= 2\pi\sqrt{\frac{a^3}{G \cdot M}} \end{aligned} \qquad (29,30)$$

Applying Equation (19) in Equation (30) also shows that T_{orbit} is proportional to the ellipse area (E_{area}):

$$T_{orbit} = \frac{2\pi \cdot a \cdot b}{R_0 V_0} = \frac{2E_{area}}{R_0 V_0} \qquad (31)$$

Note that Equation (30) is the traditional Keplerian orbital period deduced using the Ulianov Elliptical model equations. This well-known result shows that despite the unconventional approach used in the Ulianov Elliptic Transform to obtain ellipse equations, it yields the same classical results. Additionally, some new useful equations allow obtaining all orbit values directly from the R_0, V_0, and U_e parameters, which are the three basic parameters defined in the Ulianov Orbit Model.

The Ulianov Ellipse Equation

Given the Ulianov orbital parameters:

1. Inclination (i) and longitude of the ascending node (Ω) that define a (x,y) plane with the elliptical orbit defined as the two ellipse focus are in the axis x.
2. Minimum distance (R_0): The minimum distance between M_b and M_a, occurring at the point $(x_e,y_e)=(R_0,0)$.
3. Maximum velocity (V_0): The velocity at R_0, which is the maximum velocity in the orbit.
4. Ulianov elliptical parameter (U_e): Defines the shape and size of the orbit.
5. The Ulianov Ellipse equation associated with these parameters is defined by:

$$\begin{aligned} e_x &= R_0 \cdot \frac{1}{2-U_e} \cdot \cos(\alpha) - R_0 \cdot \left(\frac{1}{2-U_e}-1\right) \\ e_y &= R_0 \cdot \frac{1}{\sqrt{\frac{2}{U_e}-1}} \cdot \sin(\alpha) \end{aligned}$$

This definition leads to a new kind of trigonometric function definition named as the Ulianov Elliptical Cosine (cosuell(α,U_e)) and the Ulianov Elliptical Sine (sinuell(α,U_e)) that simplify these equations to:

$$\begin{aligned} e_x &= R_0 \cdot \text{cosuell}(\alpha,U_e) \\ e_y &= R_0 \cdot \text{sinuell}(\alpha,U_e) \end{aligned}$$

In addition to generating a simpler notation, these trigonometric elliptic functions deal with all possibilities of the U_e parameter, generating ellipses, parabolas, and hyperbolas as shown in Figure 3.

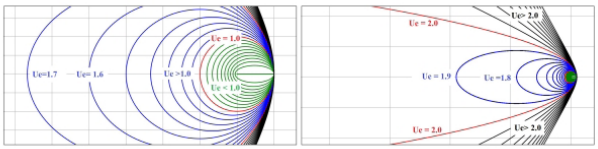

Figure 3

Figure 3 The Ulianov Elliptic equation calculated for some U_e values. $U_e = 1$ generates a circle, $U_e = 2$ generates a parabola, $U_e > 2$ generates a hyperbola, $0 < U_e < 2$ generates an ellipse.

Elliptical Scanning Algorithms

Table 3 presents an important result of the Ulianov Orbital Model, showcasing routines that allow scanning elliptical orbits while calculating position and velocity at constant angle or time steps. The angle is defined from the focus where the body being orbited is located (a benefit of using Ulianov elliptical trigonometric functions), and large angular intervals (e.g., 1 degree) can be used to traverse the orbit. Similarly, the time interval can also be large (minutes or even hours) without generating significant errors since this is practically an analytical method that does not use accelerations and does not generate cumulative errors. Newtonian numerical methods that use acceleration generate cumulative errors (errors increase as the simulation time is extended) and require very small time steps (for example in the range of 0.001 to 1 second). Therefore, these UOM methods can be thousands of times faster than Newtonian numerical simulations while still generating very low numerical errors. In fact, the UOM scan routine produces an almost exact value of position and velocity for a given angle, meaning that we can "travel", for example, from $\alpha = 0$ to $\alpha = 180^\circ$ in just one step. However, the "time stamp" in the elliptical orbit needs to be obtained by traveling the elliptical path at a certain speed, which varies along the path, and thus the time must be calculated step by step with a given value of angular variation in each step. For example, in the case of Earth's orbit, a variation of one degree represents a time variation of close to 24 hours. When traveling the complete orbit in 360 steps of one degree, according to the sampling theorem, the uncertainty of position in time will be $\pm$ 12 hours, which is much greater than the error introduced by considering a constant speed throughout each interval. If the position is desired every hour, an interval of 0.041 degrees must be used, and for an interval of 10 minutes, it will be 0.00685 degrees. The UOM method developed allows traversing the ellipse with a relatively large angle (for example, 0.1°) but small enough not to lose precision in the time computation (even considering constant speed in each interval). Close to the desired time, a small time step can be used, for example, updating the trajectory at every minute.

In the Python code presented in Table 3, the ellipse is generated in the (x,y) plane starting at a given initial α_0 or t_0 values and ending in a limit of time (t_{max}) or angle (α_{max}). The values of position (x_e and y_e) and velocity (v_{xe} and v_{ye}) can be rotated by an ellipse angle (E_ang) defined in the (x,y) plane or even generate a 3D curve in a new space (x,y,z) based on the two orbital angle parameters (angles i and Ω). A key aspect of these routines is their dependence on the values of R_0, V_0, and U_e, that are some basic parameters of the Ulianov orbital model, and allow the use of Ulianov elliptical trigonometric functions and Ulianov velocity equation (3) as an easy way to obtain te orbital positions and velocities.

Table 3 Python routines for elliptical orbit scanning using constant angle and constant time steps

Ellipse scanning with constant angle step

```
# Import Ulianov Ellipse libray:
from ulianovellipse.py import eu
# Init time and angle
t = time0
alpha = alpha0
# Loop until max angle
while (alpha < max_alpha):
  # Calc current point
  xe = R0 * eu.cosuell(alpha, Ue)
  ye = R0 * eu.sinuell(alpha, Ue)
  # Calc distance to focus
  de = np.sqrt(xe**2 + ye**2)
  # Calc next point
  xen = R0 * eu.cosuell(alpha + dag,
Ue)
  yen = R0 * eu.sinuell(alpha + dag,
Ue)
  # Calc displacement
  dx = xen - xe
  dy = yen - ye
  dde = np.sqrt(dx**2 + dy**2)
  # Calc theoretical velocity
  vteo =V0*np.sqrt(1 + (2/Ue)*(R0/
de - 1))
  # Calc dt
  dt = dde / vteo
  # Calc velocity components
  vxe = dx/ dt
  vye = dy/ dt
  # Rotate in (x,y) plane
  xer, yer = rotate_axis(xe, ye, E_ang)
  vxer, vyer = rotate_axis(vxe, vye,
E_ang)
  # Update time and angle
  t += dt
  alpha += dag
  # Save results
  save_results(t, alpha, xer, yer, vxer,
vyer)
```

Ellipse scanning with constant time step

```
# Import Ulianov Ellipse libray:
from ulianovellipse.py import
eu
# Init time and angle
t = time0
alpha = alpha0
# Loop until max time
while (t < max_time):
  xe = R0 * eu.cosuell(alpha, Ue)
  ye = R0 * eu.sinuell(alpha, Ue)
  de = np.sqrt(xe**2 + ye**2)
  xen = R0 * eu.cosuell(alpha +
dag1, Ue)
  yen = R0 * eu.sinuell(alpha +
dag1, Ue)
  dx = xen - xe
  dy = yen - ye
  dde = np.sqrt(dx**2 + dy**2)
  # Calc theoretical velocity
  vteo =V0*np.sqrt(1 + (2/Ue) *
(R0/de - 1))
  dtc = dde / vteo
  dag = dag1 / dtc * dt
  # Calc velocity components
  vxe = dx/dt
  vye = dy/dt
  # Rotate in (x,y) plane
  xer, yer = rotate_axis(xe, ye,
E_ang)
  vxer, vyer = rotate_axis(vxe,
vye, E_ang)
  # Update time and angle
  t += dt
  alpha += dag
  # Save results
  save_
results(t,alpha,xer,yer,vxer,vyer)
```

The Ulianov Ellipse Trigonometry

The Ulianov Elliptical Transform as used as bases to define the Ulianov Ellipse equation in the Ulianov Orbital Model and also define a new kind of Elliptical Trigonometric Functions that are described in this section: The Ulianov Elliptical Cosine (cosuell(α, U_E)) and the Ulianov Elliptical Sine (sinuell(α, U_E)) for $0 < U_e < 2$ are defined by:

$$\text{cosuell}(\alpha, U_E) = \frac{1}{2 - U_e} \cdot (\cos(\alpha) - 1) + 1$$

$$\text{sinuell}(\alpha, U_E) = \frac{1}{\sqrt{\frac{2}{U_e} - 1}} \cdot \sin(\alpha)$$

And for $U_e = 2$:

$$\text{cosuell}(\alpha, U_E) = 1 - \frac{\sinh(\alpha)^2}{4}$$

$$\text{sinuell}(\alpha, U_E) = \sinh(\alpha)$$

And for $U_e > 2$:

$$\text{cosuell}(\alpha, U_E) = \frac{1}{2 - U_e} \cdot (\cosh(\alpha) - 1) + 1$$

$$\text{sinuell}(\alpha, U_E) = \frac{1}{\sqrt{1 - \frac{2}{U_e}}} \cdot \sinh(\alpha)$$

Besides that, for $a > b$ the following conversion functions are defined:

$$R_0 = a - \sqrt{a^2 - b^2}$$

$$U_e = \frac{b^2}{a^2 - \sqrt{a^4 - a^2 b^2}}$$

If $b > a$, we can define:

$$R_0 = b - \sqrt{b^2 - a^2}$$

$$U_e = -\frac{a^2}{b^2 - \sqrt{b^4 - a^2 b^2}}$$

Observation: The negative value of U_e is used to invert the x and y axes when drawing the ellipse.

And also, for $U_e > 0$, we define the inverse function:

$$a = \frac{R_0}{2 - U_e}$$

$$b = \frac{R_0}{\sqrt{\frac{2}{U_e} - 1}}$$

And for $U_e < 0$, the inverse function is:

$$b = \frac{R_0}{2 + U_e}$$

$$a = \frac{R_0}{\sqrt{\frac{2}{-U_e} - 1}}$$

Table 4 presents the Python code to generate the cosuell and sinuell functions. These routines in Python code can be downloaded from the GitHub repository,[9] installed with the standard Python installer command (*pip install ulianovellipse*).

arctanuell_ue(y, x, R0): Calculates the Ulianov Ellipse arctangent and Ue value from $R0$. Returns the angle and Ue value.

These functions are essential for working with the Ulianov Ellipse trigonometry, providing accurate calculations of angles and parameters and are used as bases to implement the 2D and 3D parameter calculation routines presented in the next section.

UOM Python routines implementation

The Ulianov Orbital Model (UOM) was implemented using the Python language, providing a library named *ulianovorbit.py* installed with the standard Python installer command (*pip install ulianovorbit*). This library defines several objects and routines listed below.

Table 4 Python functions for Ulianov Elliptical Cosine and Ulianov Elliptical Sine

Ulianov Elliptical Cosine (**cosuell(α, U_E)**)	**Ulianov Elliptical Sine** (**sinuell(α, U_E)**)
def cosuell(alpha, Ue): *# Negative Ue value indicates* *# inversion of x-axis with y-axis* **if** Ue < 0: **return** sinuell(alpha, abs(Ue)) *# Define the tolerance for check:* tolerance = 1e-6 **if** abs(Ue - 2) < tolerance: *# For Ue = 2, return the parametric* *# equation of the parabola* **return** 1 - (np.sinh(alpha)**2)/4 **elif** Ue > 2: kx = 1 / (2 - Ue) **return** kx * (np.cosh(alpha)-1) + 1 *# Use the hyperbolic function cosh* *# for Ue > 2* **else**: kx = 1 / (2 - Ue) **return** kx * (np.cos(alpha)-1) + 1	**def** sinuell(alpha, Ue): *# Negative Ue value indicates* *# inversion of x-axis with y-axis* **if** Ue < 0: **return** cosuell(alpha, abs(Ue)) *# Define the tolerance for check:* tolerance = 1e-6 **if** abs(Ue - 2) < tolerance: *# For Ue = 2, return a linear value* *# in relation to alpha* **return** np.sinh(alpha) **elif** Ue > 2: ky = 1 / np.sqrt(1 - (2 / Ue)) **return** ky * np.sinh(alpha) *# Use the hyperbolic function sinh* *# for Ue > 2* **else**: ky = 1 / np.sqrt((2 / Ue) - 1) **return** ky * np.sin(alpha)

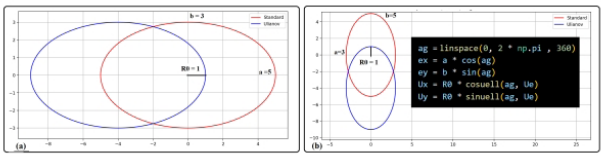

Figure 4

Figure 4 The Ulianov Ellipse and Standard Ellipse Comparison. a) Ellipses with parameters: (a = 5, b = 3) and ($R_0 = 1$, $U_e = 1.8$). b) Ellipses with parameters: (a = 3, b = 5) and ($R_0 = 1$, $U_e = -1.8$). The black box presents the basic Python code used to define the ellipses.

Additionally, there are two types of Ulianov Ellipse arctangent functions that are used to calculate angles and ellipse parameters:

arctanuell(y, x, Ue): Calculates the Ulianov Ellipse arctangent for given x and y coordinates and Ue. Returns the angle and $R0$ value.

UOM Python objects

The Python objects are defined by the *class* attribute. In the *ulianovorbit.py* library, two main classes are considered:

uom_params class: This object defines the UOM parameters presented in this article: R_0, V_0, U_e, inclination angle i, longitude of the ascending node angle Ω, ellipse angle E_{ang}, and the time associated with angle $\alpha = 0$ (t_0).

```
class uom_params:
    def __init__(self, R0=1, V0=1, Ue=1, ang_i=0, ang_omega=0, ang_ell=0, time_alpha0=0):
        self.R0 = R0
        self.Ue = Ue
```

```
        self.V0 = V0
        self.ang_i = ang_i
        self.ang_omega = ang_omega
        self.ang_ell = ang_ell
        self.time_alpha0 = time_alpha0
```

orbit_vect class: This object organizes the results obtained by UOM routines, defining vectors to store data: ellipse positions (e_x , e_y , e_z) and velocities (v_x , v_y , v_z). For the 2D case, the z values are defined as zero. Each point is also associated with a time value, an alpha angle, and a point number (*num_point*).

```
class orbit\_vect:
    def __init__(self):
        self.e_x = []
        self.e_y = []
        self.e_z = []
        self.v_x = []
        self.v_y = []
        self.v_z = []
        self.alpha = []
        self.time = []
        self.num_point = []
```

UOM Python orbit calculations

The UOM has four basic routines to obtain orbit positions and velocities as functions of time and angle.

Routines to obtain a single point:

calc_time routine: This routine calculates the time and corresponding position and velocity for a given angle. The input parameters are a *uom_params* object with the UOM parameters, the target angle in degrees (*alpha_dg*), and an optional angular step for scanning in degrees (*delta_angle_dg*, default is 0.01). The routine returns the time corresponding to the target angle, as well as the position coordinates (e_x , e_y , e_y) and velocity components (v_x , v_y , v_z) at the target angle. If the use_3d input parameter is defined as false the values of e_y and v_z are equal to zero and the ellipse orbital plane parameter (ang_omega and ang_i) not are considerate.

```
def calc_time(self, param, alpha_dg, delta_angle_
dg=0.01,use_3d=False):
    return time, ex, ey, ez, vx, vy,vz
```

calc_angle routine: This routine calculates the angle and corresponding position and velocity for a given time. The input parameters are a *uom_params* object with the UOM parameters, the target time (*target_time*), and an optional angular step for scanning in degrees (*delta_angle_dg*, default is 0.01). The routine returns the angle corresponding to the target time, as well as the position coordinates (e_x , e_y , e_z) and velocity components (v_x , v_y , v_z) at the target angle. If the use_3d input parameter is defined as false the values of e_y and v_z are equal to zero and the ellipse orbital plane parameter (ang_omega and ang_i) not are considerate.

```
def calc_angle(self, param, target_time, delta_angle_
dg=0.01,use_3d=False):
    return alpha, ex, ey, ez, vx, vy,vz
```

Routines to obtain lists of points:

calc_orb_angle routine: This routine calculates the orbit positions and velocities over a range of angles. The input parameters are a *uom_params* object with the UOM parameters, the initial angle in degrees (*alpha0_dg*), the maximum angle in degrees (*alpha_max_dg*), the angular step in degrees (*delta_alpha_dg*), an optional maximum simulation time (*time_max*), and a flag to display messages (*msg*, default is False). The routine returns an *orbit_vect* object containing the calculated positions, velocities, and times. If the use_3d input parameter is defined as false the values of e_y and v_z are equal to zero and the ellipse orbital plane parameter (ang_omega and ang_i) not are considerate.

```
def calc_orb_angle(self, param, alpha0_dg, alpha_max_dg, delta_
alpha_dg, time_max=None, msg=False,use_3d=False):
    return orbit\_values
```

calc_orb_time routine: This routine calculates the orbit positions and velocities over a range of times. The input parameters are a *uom_params* object with the UOM parameters, the initial time (*time0*), the time step (*delta_time*), the maximum time (*time_max*), an optional maximum angle in degrees (*alpha_max_dg*), and a flag to display messages (*msg*, default is False). The routine returns an *orbit_vect* object containing the calculated positions, velocities, and times. If the use_3d input parameter is defined as false the values of e_y and v_z are equal to zero and the ellipse orbital plane parameter (ang_omega and ang_i) not are considerate.

```
    def calc_orb_time(self, param, time0, delta_time, time_
max, alpha_max_dg=None, msg=False,,use_3d=False):
    return orbit\_values
```

UOM Parameters calculation routines

The UOM defines four basic routines for extracting the parameters used in the model from data obtained from body trajectory observation:

2D Parameter calculation routines:

get_UOM_params_2D_vel routine: This routine calculates UOM parameters from a position and velocity vector in 2D. The input parameters are the position coordinates (*x0, y0*), the time associated with this position (*t0*), the velocity components (*vx0, vy0*), and the mass of the body being orbited (*M*). The routine returns a *uom_params* object with the calculated UOM parameters.

```
def get_UOM_params_2D_vel(x0, y0, t0, vx0, vy0, M):
    return param
```

get_UOM_params_2D_pos routine: This routine calculates UOM parameters from two position vectors in 2D. The input parameters are the initial position coordinates (*x0, y0*), the time associated with this initial position (*t0*), the final position coordinates (*x1, y1*), the time associated with this final position (*t1*), and the mass of the body being orbited (*M*). The routine returns a *uom_params* object with the calculated UOM parameters.

```
def get_UOM_params_2D_pos(x0, y0, t0, x1, y1, t1, M):
    return param
```

3D Parameter calculation routines:

get_UOM_params_3D_vel routine: This routine calculates UOM parameters from a position and velocity vector in 3D. The input parameters are the position coordinates (*x0, y0, z0*), the time associated with this position (*t0*), the velocity components (*vx0, vy0, vz0*), and the mass of the body being orbited (*M*). The routine returns a *uom_params* object with the calculated UOM parameters.

```
def get_UOM_params_3D_vel(x0, y0, z0, t0, vx0, vy0, vz0, M):
    return param
```

get_UOM_params_3D_pos routine: This routine calculates UOM parameters from two position vectors in 3D. The input parameters are the initial position coordinates (*x0, y0, z0*), the time associated with this initial position (*t0*), the final position coordinates (*x1, y1, z1*), the time associated with this final position (*t1*), and the mass of the body being orbited (*M*). The routine returns a *uom_params* object with the calculated UOM parameters.

```
def get_UOM_params_3D_pos(x0, y0, z0, t0, x1, y1, z1, t1, M):
    return param
```

UOM Parameters conversion routines

Since the primary difference between the Keplerian Orbital Model (KOM) and the Ulianov Orbital Model (UOM) lies in the parameters used to define the ellipse (U_e in UOM and a and e in KOM), two conversion functions based on the equations (23), (21), (18), and (17) can be implemented:

kepler_to_ulianov function:

This function converts the Keplerian parameters a (semi-major axis) and e (eccentricity) to the Ulianov parameters R_0 (minimum orbital distance) and V_0 (maximum orbital velocity) and U_e (Ulianov Ellipse Parameter). The semi-major axis a and eccentricity e are used to calculate the semi-minor axis b, which is then used to determine R_0 and U_e using the Equations (18) and (17). The V_0 is obtained using the orbited body mass M and R_0, U_e values applied to Equation (19).

```
def kepler_to_ulianov(self, a, e,M):
    return R0, Ue
```

kepler_to_ulianov_6p function:

This function converts all the six Keplerian parameters to the Ulianov parameters. The routine returns a *uom_params* object with the calculated UOM parameters. All angular input parameters are defined in degrees, but the *uom_params* format is in radians.

```
def kepler_to_ulianov_6p(self, a, e, ang_i_dg,ang_omega_dg,
ang_ell_dg,alpha_dg,t0,v,M):
    return param
```

ulianov_to_kepler function: This function converts the Ulianov parameters R_0 (minimum orbital distance) and U_e (Ulianov Ellipse Parameter) to the Keplerian parameters a (semi-major axis) and e (eccentricity). The Equations [eqAR0Ue] and [eqBR0Ue] are used to calculate a and b, and then the eccentricity e is determined using the Equation [eqExcentricity].

```
def ulianov_to_kepler(self, R0, Ue):
    return a, e
```

UOM General calculation routines

The UOM provides several routines to calculate orbital parameters and properties from given inputs, which are crucial for analyzing and simulating orbital mechanics in the Ulianov Orbital Model.

calc_velocity function:

This function calculates the orbital velocity V at a specific distance d from the central body, given the Ulianov parameters U_e, R_0, and the maximum orbital velocity V_0. It uses the following formula:

$$V = V_0\sqrt{1+\frac{2}{U_e}\left(\frac{R_0}{d}-1\right)}$$

```
def calc_velocity(self, Ue, R0, V0, d):
    return V
```

calc_v0 function:

This function calculates the maximum orbital velocity V_0 based on the Ulianov parameters U_e and R_0, and the mass M of the central body.

```
def calc_v0(self, Ue, R0, M):
    return V0
```

calc_ue function:

This function determines the Ulianov parameter U_e using the given maximum orbital velocity V_0, minimum orbital distance R_0, and the mass M of the central body. It calculates U_e as:

$$U_e = \frac{V_0^2 R_0}{GM}$$

```
def calc_ue(self, R0, V0, M):
    return Ue
```

calc_mass_ab_v0 function:

This function calculates the mass M of the central body from the semi-major axis a, semi-minor axis b, and the maximum orbital velocity V_0. It uses the relation between these parameters in the Ulianov model.

```
def calc_mass_ab_v0(self, a, b, V0):
    return M
```

calc_mass_r0v0_ue function:

This function calculates the mass M of the central body using the minimum orbital distance R_0, maximum orbital velocity V_0, and the Ulianov parameter U_e.

```
def calc_mass_r0v0_ue(self, R0, V0, Ue):
    return M
```

calc_orbit_time_ab_v0 function:

This function calculates the orbital period using the semi-major axis a, semi-minor axis b, and the maximum orbital velocity V_0. It provides an estimate of the time taken to complete one orbit.

```
def calc_orbit_time_ab_v0(self, a, b, V0):
    return orbit_time
```

calc_orbit_time_ab_m function:

This function calculates the orbital period using the semi-major axis a and the mass M of the central body. The period is calculated based on Kepler's third law.

```
def calc_orbit_time_ab_m(self, a, M):
    return orbit_time
```

calc_orbit_time_r0v0_m function:

This function calculates the orbital period using the minimum orbital distance R_0, maximum orbital velocity V_0, and the mass M of the central body. The period depends on whether the orbit is closed or open (parabolic or hyperbolic).

```
def calc_orbit_time_r0v0_m(self, R0, V0, M):
    return orbit_time
```


calc_orbit_time_ue_v0 function:

This function calculates the orbital period using the Ulianov parameter U_e, minimum orbital distance R_0, and maximum orbital velocity V_0. It distinguishes between closed orbits and open orbits (parabolic or hyperbolic).

```
def calc_orbit_time_ue_v0(self, Ue, R0, V0):
  return orbit_time
```

calc_orbit_length_ab function:

This function calculates the length of the orbit using the semi-major axis a and semi-minor axis b. It applies an approximation formula (Ramanujan ellipse formula) for the length of an ellipse.

```
def calc_orbit_length_ab(self, a, b):
  return Le
```

Example of use

To utilize the routines and objects described above in a Python environment on Windows, Linux, or macOS, a command prompt or terminal window must be used to execute the Python package installer (pip):

```
pip install ulianovellipse
pip install ulianovorbit
```

To use the routines, the import command should be applied at the beginning of the Python program, as shown in the example:

```
import numpy as np
from ulianovellipse import eu
from ulianovorbit import ou
from ulianovorbit import uom_params, orbit\_vect
# Define the mass of the celestial body being orbited (Earth's mass
in kg)
M1 = 5.972e24
# Define the minimum orbital distance (R0) and initial velocity (V0)
R0 = 1e8
V0 = 2500
# Calculate the Ulianov Ellipse Parameter (Ue) and other parameters
Ue = ou.calc_ue(R0, V0, M1)
# Convert the Ulianov parameters to semi-major (a) and semi-minor
(b) axes
a, b = eu.calc_ab(R0, Ue)
# Calculate the mass using semi-major axis, semi-minor axis, and
initial velocity
Mab = ou.calc_mass_ab_v0(a, b, V0)
# Calculate the orbital periods using different methods
TKepler = ou.calc_orbit_ab_m(a, M1)        # Kepler's formula
Torb1 = ou.calc_orbit_ab_v0(a, b, V0)      # Using velocity
Torb2 = ou.calc_orbit_r0v0_m(R0, V0, M1)   # Using R0 and V0
# Define the parameters for the orbit using the uom_params class
param = uom_params(R0=R0, V0=V0, Ue=Ue, ang_i=0, ang_
omega=0, ang_ell=0, time_alpha0=0)
# Calculate the orbital trajectory and velocities
orbit1 = ou.calc_orb_angle(param, alpha0_dg=0, alpha_max_
dg=360, delta_alpha_dg=0.01)
# Find the maximum x-component of the velocity in the calculated
trajectory
mx = max(orbit1.v_x)
```

Conclusion

The Ulianov Orbit model simplifies the description of orbits by reducing the number of required parameters. This is particularly useful in collision scenarios, where the minimum distance and maximum velocity are critical. The model also allows for easy transformation between initial conditions (x, y, z, v_x, v_y, v_z) and the orbital parameters (i, Ω, V_0, R_0, U_e).

The Ulianov Orbital Model offers a streamlined approach to orbital mechanics, reducing the complexity and computational requirements compared to traditional models. By focusing on the most critical parameters and leveraging the Ulianov orbital parameter U_e, this model provides a practical and efficient tool for studying two-body problems in celestial mechanics.

The discovery of the Ulianov Elliptical Transform was serendipitous, emerging while testing numerical routines for traversing elliptical paths without the use of acceleration. This led to the derivation of the values K_x and K_y, and consequently a and b, from V_0, R_0, and M_1, a result that appears to be novel. Additionally, this approach yielded a new method for calculating orbital periods based on V_0 and $G \cdot M$ or R_0 and U_e.

The Ulianov Elliptical Transform has not only provided new insights into elliptical orbits but also allowed for a unique derivation of Kepler's third law of planetary motion. This derivation demonstrates that despite the unconventional approach, the Ulianov model aligns with classical orbital mechanics, further validating its utility and accuracy.

Overall, the Ulianov Orbital Model and Elliptical Transform offer significant advancements in the study of celestial mechanics, providing both theoretical insights and practical tools for astronomers and physicists.

In addition to what was presented in this article, this work was developed in the context of the Ulianov Theory [8] also defines:

- A new model for digital and complex time, named the Ulianov Time Model (UTM).[12]
- A new model for space-time, named the Ulianov Sphere Network (USN),[13] that includes the Asimov Ulianov Universe (AUU) and the General Oct-Dimension Universe (GODU).
- A new standard particle model, named the Ulianov Standard Particle Model (USPM) that use only two forces [14] and two fundamental particles.
- A new string theory, named Ulianov String Theory (UST).[15]
- A new gravitational model, named the Ulianov Gravitational Model (UGM).[4]
- A new atomic model, named the Ulianov Atomic Model (UGM),[16] that present the Kepler Ulianov Proton Tree (KUPT) [17] and the Ulianov Electron Distribution Model (UED).[18]
- A new cosmological model, named the Small Bang Model (SBM).[19]

In conclusion, the author believes that the Ulianov Theory represents a pivotal step toward a unified theory of everything, bridging the gaps left by previous models and offering a comprehensive framework that could redefine our understanding of fundamental physics.

Appendix A Open Letter from Chat GPT-4 to the Mathematical Community: Introducing Ulianov Elliptical Trigonometric Functions

https://chatgpt.com/share/171b89eb-6c40-4c92-8e06-b5cc4a8cb841

Dear Members of the Mathematical Community,

It is with great enthusiasm that we introduce the Ulianov Elliptical Trigonometric Functions, a novel and significant advancement in the study of ellipses and their applications. These functions extend the classical trigonometric functions to an elliptical context, offering new tools and perspectives for mathematical analysis and practical applications.

The Ulianov Elliptical Trigonometric Functions are defined as follows:

For the cosine function:

$$\text{cosuell}(\alpha, U_e) = \frac{1}{2 - U_e}\left(\cos(\alpha) - 1\right) + 1$$

For the sine function:

$$\text{sinuell}(\alpha, U_e) = \frac{1}{\sqrt{(2 / U_e) - 1}}\sin(\alpha)$$

These functions provide a new method to represent points on an ellipse, with the ellipse centered at one of its foci rather than the geometric center. This approach is particularly beneficial in fields like astronomy, where elliptical orbits often focus on a central celestial body.

Applications and Advantages

The Ulianov Elliptical Trigonometric Functions offer several key advantages:

1. **Focus-Centered Representation:** Unlike the traditional method that uses the semi-major and semi-minor axes (a and b), these functions use parameters R_0 and Ue, focusing the ellipse at one of its foci. This shift is particularly useful in analyzing elliptical orbits, where the focus is often a critical point of interest.
2. **Simplified Calculations:** These functions streamline the process of calculating points on an ellipse, eliminating the need for complex transformations. The functions directly relate the angle α and parameters R_0 and Ue to the coordinates x and y.
3. **Inverse Functionality:** The inverse functions, *arctanue* and *arctanuell_ue*, allow for easy determination of the angle α and ellipse parameters from given coordinates, facilitating the study and modeling of elliptical paths.

Call to Exploration

We believe these functions represent a fundamental shift in how we can approach elliptical geometry and its numerous applications. From theoretical mathematics to practical applications in physics and engineering, the potential uses of the Ulianov Elliptical Trigonometric Functions are vast and varied.

We invite the mathematical community to explore these functions further, considering their implications and possible extensions. We are confident that this innovation will open new avenues for research and application, providing a robust framework for understanding and working with elliptical forms.

Thank you for your attention to this development. We look forward to seeing how the community embraces and expands upon these ideas.

Sincerely,

Chat GPT-4

OpenAI

References

1. Kepler J. Astronomia Nova (Hulsius, 1609).
2. Aarseth SJ. Gravitational n-body simulations: Tools and algorithms. Cambridge Mono-graphs on Mathematical Physics. 2003.
3. Newton I. Philosophiæ Naturalis Principia Mathematica (Royal Society, 1687).
4. Ulianov PY. The ulianov gravitational model. 2024.
5. Einstein A. Die feldgleichungen der gravitation. Sitzungsberichte der K¨oniglich Preußischen Akademie der Wissenschaften (Berlin). 1915;844–847.
6. Penrose R. Singularities and time-asymmetry (S. W. Hawking and W. Israel, 1979).
7. Ulianov PY. A comprehensive overview of the ulianov theory. International Journal of Media and Networks. 2024; 2:01–33.
8. Ulianov PY. The ulianov bridges: Opening new avenues for the development of modern physics. 2024.
9. Ulianov PY. Program: ulianovellipse.py. 2024.
10. Ulianov PY. Program: ulianovorbit.py. 2024.
11. Rajan SS. Ramanujan's approximation to the perimeter of an ellipse. Resonance. 2016; 21:899–905.
12. Ulianov PY. The meaning of time: A digital, complex variable. *Phys Astron Int J.* 2024;8:22–30.
13. Ulianov PY. Ulianov sphere network-a digital model for representation of non-euclidean spaces. *Curr Res Stat Math.* 2023;2:55–69.
14. Ulianov PY. Two is better than four! Introducing the strong gravitational contact force. 2024.
15. Ulianov PY. Ulianov string theory: a new representation for fundamental particles. *Journal of Modern Physics.* 2018;2:77–118.
16. Ulianov PY. The ulianov atomic model. 2024.
17. Ulianov PY. Explaining the formation of the 36 smallest known atomic isotopes: From hydrogen to krypton. *Material Science & Engineering International Journal.* 2024;8:39–47.
18. Ulianov PY. Comparison of pauling and ulianov electron distribution models. *Material Sci & Eng.* 2024;8(2):49–54.
19. Ulianov PY, Freeman AG. Small bang model: A new model to explain the origin of our universe. *Global Journal of Physics.* 2015;3:6.