A Titanic Space Odyssey

Final Report

Project 1-2: Group 27

Group Members:

Alexander Safi	Claudia Sanchez
Abdullah Sahin	Ilian Ariesen
Jack Waterman	Giacomo Terragni

22/06/2021

To the Faculty of Data Science and Artificial Intelligent at the University of Maastricht, The Netherlands

Abstract

This study aims to bring a manned mission to land safely on Titan and return to Earth. The mission comprises three main parts. The first part is an exploratory mission in which a space probe is launched from Earth to Titan with the aim of finding a suitable landing site on Titan for the following part. Secondly, a fully-fledged multiple-stage rocket is to be launched from Earth in such a way that it enters a geostationary orbit around Titan. Finally, a safe landing is performed, opening the way for astronauts to examine the natural satellite.

Multiple external factors affect the spacecraft during the flight. In order to reduce the complexity, we assume that these only include the gravitational forces and the forces generated by the wind. Furthermore, we opted for several techniques to tackle these obstacles. Corrections to the trajectory were performed using a thrust, taking mass to be burned into account. Open and closed-loop controllers were implemented, stabilising the spacecraft's landing.

It is important to note that many assumptions were made during this mission, which explicitly simplifies the complexity of an actual space mission.

The core of the report fall into the following crucial criteria :

- Safety
- Efficiency
- Precision

To examine the robustness of the designs and algorithms constructed, multiple experiments were undertaken. These experiments played a fundamental role in understanding certain concepts.

1. Introduction

Titan is the largest moon of Saturn, discovered in 1655, is the only moon known to have a dense atmosphere. Clear evidence has shown that it is the only body in space, other than Earth, with stable bodies of surface liquid.
The mission's aim is to land safely on Titan while minimizing the resources, namely time and cost. However, the spaceship is intercepted by multiple external forces during the flight and landing.
Hence, the main studied research in this article are:

- How do we accurately reproduce the trajectory of a launched spacecraft from earth to a given target in the solar system?
- How can we safely and precisely execute a spatial landing, taking into consideration the external forces of Titan's atmosphere?

Furthermore, the report follow a specific structure :

Firstly, several assumptions that were made for this problem will be discussed. Moreover, reasons for certain assumptions will be given, as well as the effects of these assumptions on the accuracy of our model.

Secondly, in the methods and implementation section, several numerical approaches to solve physical problems are tackled. The physics include solving differentiation equations and finding the optimal trajectory using numerical integration.

In the Landing section, two main themes were approached. Starting by the creation of a stochastic wind model, followed by the establishment of controllers. Both an open-loop controller and a closed-loop controller were implemented, hence these systems will be further explored in this section.

Within the Experiments, the experiments conducted on the numerical methods to improve the performance of the methods are described and motivation is given as to why these experiments were needed to answer the research questions.

The Discussion section compares the found results to each other to explore and evaluate them.

Finally, in the Conclusion, a final verdict is drawn from all results to answer all research questions. This chapter includes recommendations as well as suggestions for further research.

2. Assumptions

The purpose of this section is to discuss the assumptions that were made in order to reduce the scope of the investigation to a plausible state. Thus, it is vital to take into consideration these assumptions in the rest of the report.

- Reduced amount of celestial bodies; in this investigation only the Sun, Mercury, Venus, Earth, Mars, Jupiter, Saturn, Titan, Neptune and Uranus were considered for the simulation. No asteroids, comets nor other planets were included in this study.

- Spaceship : The only forces acting on this spaceship are the gravitational forces during the flight, the wind force and the drag force while landing.

- Location of launch: to simplify the finding of an optimal location and time of the initial launch, it was assumed the rocket would launch from orbit of relevant planets; making the rotation of the planets irrelevant for the launching. As default, the launch would take place the 1st of April 2020.

- Landing: The inter-planet trajectories are calculated in three dimensions. However, for the landing, only two dimensions were acknowledged and inquired. Moreover, it is assumed that the landing spot has no obstacles which could unexpectedly interfere with the landing.

- Landing back on Earth: The probe will use a parachute when coming back to Earth. It is assumed the probe will land in the sea, as it is a softer and hence safer surface for the final landing.

- Number representation: Floating point numbers are only able to represent exactly dyadic numbers, i.e. they can be in the form $\frac{p}{2^q}$, where p and q are real numbers. Therefore, the result of arithmetical operations involves frequently rounding-off errors, these errors can be accumulated evolving to a significant loss of accuracy. Throughout the investigation some techniques were used to reduce these errors, and hence, we assume the results obtained have an acceptable accuracy we can rely on. Moreover, a discussion at the end of this report is contemplated in order to drastically reduce these computation errors.

3. Methods & Implementation

In space, the only factor that is affecting a planet's displacement over time is its own velocity. Nonetheless, by supposing that there were no forces affecting a planet's velocity, the planets would not be orbiting around. This means there has to be an acceleration such that it at least changes the direction of the velocity over time.

Newton's law of universal gravity implies that one object exhibits a gravitational force on another object:

$$F_g = sum \;\; G\, m_i\, m_j\, (x_j - x_i) \,/\, ||x_i - x_j||^3$$

Where m_i represents the mass of the object of interest, m_j represents the mass of the object that exhibits a gravitational force on the object of interest, x_j represents the displacement of the exhibitor, x_i represents the displacement of the object of interest, $||x_i - x_j||$ represents the Euclidean distance between x_i and x_j.

However, as this is a force and not an acceleration, it does not answer the question of what is changing a planet's velocity over time. So, in order to answer this, Newton's second law of motion comes into place. Newton's second law of motion says the involved force F equals to the mass m of the object multiplied by its acceleration a. This leads us to the following equation:

$$F = m\, a$$

When Newton's law of universal gravity is combined with Newton's second law of motion, we get a system of equations which is the answer to the question of how does a planet's velocity change over time:

$$F_g = sum \;\; G\, m_i\, m_j\, (x_j - x_i) \,/\, ||x_i - x_j||^3$$
$$a = F_g \,/\, m$$

Consequently, through this system of equations, it becomes now possible to calculate an acceleration for any planet at any time in the simulation. Since, an approach to calculate the acceleration that influences the velocity over time, we now need an approach to approximate the actual displacement and velocity of a planet evaluated at a certain time t and that is where differential equations come in hand.

3.1 Differential equation

As mentioned earlier the displacement and velocity values of a planet changes over time, therefore let us say the first evaluation of a planet's displacement and velocity is manually measured and happens at time t_0, then the next evaluation which happens at time t_1 can be approximated by finding and using the planet's rate-of-change over time at time t_0.

This implies that we are setting the planet's rate-of-change to be equal to some result we obtain through some calculations made. That is what Mathematicians would call a differential equation. So, more abstractly, a differential equation is an equation that evaluates a value or a variable over another one to be equal to some function.

So far, we have been mentioning a planet's displacement and velocity separately, but if grouped together, they form the state of a planet at a time t. Thus, leading us to the following equation where the planet's state y equals its displacement x and its velocity v:

$$y = (x, v)$$

When we apply the definition of a planet's state in defining the planet's state rate-of-change over time dy/dt to be equal to the result of some function f that takes in time t and the respective state y, we get the following differential equation:

$$dy/dt = f(t, y)$$

Furthermore, we know that the change of a planet's state y is due to a velocity v for the planet's displacement and an acceleration a for the planet's velocity, we can say the state rate-of-change over time dy/dt to be equal to a $[v, a]$, which extends the previous equation to:

$$dy/dt = f(t, y) = [v, a]$$

However, as we are working with approximations for all states of a planet after its initial measured state, then depending on how we find the approximation of the next state based on the differential equation and what evaluation time, we may end with different results.

Furthermore, if we were to come up with our own approximation method, then we would first have to prove the integrity of the method which is a tenacious process and that is why using differential equation solvers for our problem is truly convenient as there is no need to prove the method integrity.

3.1.1 Differential equation solvers

So, in order for our simulation to be capable of calculating the different planetary states for each planet with the equations we have shown so far, we have implemented two differential equation solvers, namely Euler's method and Runge-Kutta's method of the fourth order.

3.1.2 Euler's method

Our first approach in approximating the next planetary states started with Euler's method as the method is a first order method which makes it relatively easy to implement and use it as a starting point in approximating solutions.

As Euler's method is a method that numerically solves differential equations, there has to be some form of an iterative step, and there is. The method itself is a straightforward iterative step which says the next planetary state y_{i+1} equals its current state y_i plus the state's rate-of-change over time *dy/dt* multiplied with the difference between next evaluation time t_{i+1} and the current evaluation time t_i.

Substituting the state rate-of-change over time *dy/dt* for the function $f(t_i, y_i)$ that determines the rate-of-change, we get the following system of equations for Euler's method:

$$y_{i+1} = y_i + h\,f(t_i, y_i)$$

$$h = t_{i+1} - t_i$$

Furthermore, as this is an iterative approximation method, there probably should be some kind of truncation error and there is. Its global truncation error is $O(h)$ and local truncation error is $O(h^2)$, these are fairly big numbers regardless of the actual value of the step size.

3.1.3 Fourth order Runge-Kutta method

Thus, to have more accurate approximations, the classical Runge-Kutta method of the fourth order has been implemented. Unlike Euler's method, this method uses four different evaluations of the state's rate-of-change over time *dy/dt* to calculate the approximation of the next state $y_{i+1,}$

The first evaluation k_1 is the same state's rate-of-change over time *dy/dt* evaluation used in Euler's method, which results into

$$k_1 = h\,f(t_i, y_i)$$

whereas the second evaluation k_2 is done through evaluating the current planet's state y_i plus half the first evaluation k_1 at the current evaluation time t_i plus half the step size h, leading to the equation:

$$k_2 = h\,f(t_i + h/2, y_i + k_1/2)$$

As for the third evaluation k_3 is obtained by evaluating the current planet's state y_i plus half the second evaluation k_2 at the current evaluation time t_i plus half the step size h, resulting to

$$k_3 = h\,f(t_i + h/2, y_i + k_2/2)$$

while the last evaluation k_4 is a little bit different as it evaluates the current planet's state y_i plus the third evaluation k_3 at the current evaluation time t_i plus the step size h, forming the equation:

$$k_4 = h\,f(t_i + h, w_i + k_3)$$

Once the four evaluations are found, they are summed up to a single value, whereas the second evaluation k_2 and the third evaluations k_3 are counted twice. This summation is then followed by a division of a sixth which leads to the final state rate-of-change over time *dy/dt* that will be added to the current state y_i to get the next state y_{i+1}.

By summarizing everything up to this point for Runge-Kutta, we get the following system of equations representing the entire method:

$$y_{i+1} = y_i + 1/6\,(k_1 + 2k_2 + 2k_3 + k_4)$$
$$h = t_{i+1} - t_i$$
$$k_1 = h\,f\,(t_i, y_i)$$
$$k_2 = h\,f\,(t_i + h/2, y_i + k_1/2)$$
$$k_3 = h\,f\,(t_i + h/2, y_i + k_2/2)$$
$$k_4 = h\,f\,(t_i + h, y_i + k_3)$$

which has a global truncation error of $O(h^4)$ and a local truncation error of $O(h^5)$.

Both the global and local truncation errors are a double-edged sword. For small step sizes such as step sizes below h = 1 the errors become very small due to the order, however for larger step sizes, these errors can quickly escalate to very large numbers.

3.3 Numerical integration

Besides the implementation of differential equation and differential equation solvers, two different numerical integrations have taken place; Verlet integration & Newton-Raphson. The Verlet integration has been integrated due to it being a numerical method that is good at simulating systems with energy conservation, while Newton's method has been integrated to assist us with helping find a trajectory to Titan from Earth and back.

3.3.1 Verlet integration

Verlet integration was developed to calculate solutions to kinematic equations. As it is a numerical integration and not a differential equation solver it does not use a differential equation at all. The only requirement for the next planetary state y_{i+1} besides a measured or known initial state y_0 would be the acceleration a_i evaluated at time t_i.

The displacement integration itself is pretty straight forward, it says the planet's next displacement x_{i+1} equals to double its current displacement x_i subtracted by its previous displacement x_{i-1} and finally plus the involved acceleration a_i multiplied with the step size h squared.

$$x_{i+1} = 2\,x_i - x_{i-1} + a\,h^2$$

Moreover, the velocity integration is less complex than the displacement integration as it says, the next velocity v_{i+1} equals to its current velocity v_i plus the involved acceleration a multiplied with the step size h.

$$v_{i+1} = v_i + a\,h$$

However, there is one catch to the displacement integration at the very beginning of the simulation with the initial displacement x_0, because this requires displacement x_{-1} to be known. Therefore, only at the very first evaluation step, we use Euler's method to find the next state y_{i+1}.

Contrary to what you would expect, the local truncation error of the position integration is $O(h^4)$, while its global truncation error is $O(h^2)$. As for the velocity integration it performs a bit better than position integration as both its global and local truncation error are $O(h^2)$

3.3.2 Newton-Raphson

One of our mission milestones is to enter Titan's orbit. This implies that we would have to keep track of the distance between Titan and the rocket and find velocity v such that the rocket could intersect with Titan's orbit.

This is where Newton's method comes in. This numerical method finds the root(s) of a function, where a root represents a function value x_{root} such that:

$$f(x_{root}) = 0 \text{ or } f(x_{root}) \leq accuracy$$

where *accuracy* represents a number close to 0 that is significant enough for the implementer to stop iterating.

As the method solves equations numerically, it has an iterative step. Its iterative step consists of finding a x which is closer to a possible x_{root} than the previously calculated x by the algorithm. Therefore, the formula to find a next attempt for x_{root} is:

$$x_{n+1} = x_n + \frac{f(x)}{f'(x)}$$

Where x_{n+1} is the next attempt for x_{root}, x_n is the previous attempt for x_{root}, $f(x)$ is the function and $f'(x)$ is the derivative of the function.

However, our problem is based on 3-dimensional vectors instead of a single variable, this implies that we have to use the Newton-Raphson method for multiple variables. Thus, this is the formula:

$$x_{n+1} = x_n - Df' * f(x_n)$$

Where x_{n+1} is the next 3-dimensional velocity vector, x_n is the previously computed velocity vector, $Df'(x_n)^{-1}$ is the Jacobian derivative matrix of $f'(x)$and$f(x)$ represents the function.

More specifically, $f(x)$ represents the distance between the rocket and Titan

$$f(x) = p_f - p_r$$

Where p_f is the position of Titan and p_r the endpoint of the rocket's trajectory. This endpoint is computed using our implemented solvers, and it depends on the current initial velocity x_n.

Moreover, the Jacobian derivative matrix $Df'(x_n)^{-1}$ is a matrix composed of all the first-order partial derivatives of the vector x_n. To calculate each entry of the matrix, we used the three-point centred difference formula:

$$f'(x) = \frac{(f(x+h) - f(x-h))}{2h}$$

Which adapted to the data it becomes

$$f'_a(x) = \frac{(f'_a(x_b+h) - f'_a(x_b-h)}{2h}$$

Where *a* is the axis for the function calculation and *b* the axis in the vector addition and/or subtraction

3.4 Trajectory

One of the mission's main milestones is to reach Titan from Earth. The duration of the travel is dependent on the rocket velocity during its trajectory to Titan. Furthermore, depending on when and how the rocket thrusts the remaining parts of the trajectory gets affected. Therefore, we approached this milestone as a three-staged problem to solve:

1. An approach to find a velocity such that the rocket can enter Titan's orbit
2. An approach to correct the trajectory if the remaining trajectory won't be able to intersect with Titan's orbit.
3. Optimize fuel usage by reducing the velocity while being able to enter Titan's orbit within a reasonable amount of time.

3.4.1 Velocity

As explained in the section of Newton's method, the method is a fairly powerful tool to solve trajectory problems like these, due to it trying to find a velocity *v* such that the final distance between Titan and the rocket at the end of the simulation of a time interval $[t_0, t_{final}]$ would be close to zero if not already being zero.

Furthermore, the goal is not to collide with Titan but to enter its orbit, meaning the final distance does not require to be very close to zero but to the specified orbit distance interval [100.000, 350.000], choosing Newton's method over other approaches was the best decision as the interval favors the method's iterative step.

3.4.2 Correction

Due to choosing Newton's method over other options, it was not required to perform trajectory corrections if the same simulation setup from Newton's method was used for the actual rocket simulation.

However, there were a few ideas that were explored in case other options were chosen. The three main ideas were mostly centered around when the rocket should correct its trajectory. These ideas were:

1. Straight-line trajectory
2. Cone-deviations
3. Box-deviations

The straight-line trajectory should be self-explanatory as a straight line from one point to another point represents the shortest distance between the two points. Therefore, the straight-line trajectory represents the fastest trajectory the rocket could have out of all trajectories.

However, this approach is totally not fuel friendly, as Titan's displacement is changing over time and the acceleration due to the universal gravity is changing the rocket's velocity over time. Hence, it is only recommended to use such an approach when the rocket is relatively close to Titan to reduce the remaining trajectory.

In order to understand what a cone-deviation does for the trajectory correction, the forming of a cone will be provided first. For this idea to work, the height of the cone will be defined as the Euclidean distance between the rocket and Titan, the rocket will function as the tip of the cone, while the Titan will be the middle point of the circular sector.

As this cone is shaped, the goal becomes to determine whether the rocket's relative velocity is contained inside the code. If that is the case, then this implies that the rocket is moving within the allowed velocity deviation towards Titan. Else, it would imply that the rocket is outside desired deviation and its velocity would have to be corrected such that it would fall back inside the shape where the optimal correction would fall on the line representing the height of the cone.

Moreover, due to the shape of the cone, this type of correction is more fuel friendly as the rocket nears Titan, while early on it may be less fuel friendly but compared to straight-line trajectory it is way better-off.

Just like cone-deviations, the box-deviations use the same principle to decide on when to correct the rocket's velocity. The box is formed through the line representing the Euclidean distance between the rocket and Titan whereas the height, width and depth of the box is represented by the respective differences in the axes.

Therefore, the interval representing the maximum allowed velocity deviations is a big one, meaning this correction type would be most fuel friendliest out of the three, but on the cost of requiring more time for the rocket to approach Titan closely.

3.4.3 Fuel optimization

To find an initial velocity for the rocket to reach during take-off from Earth, a probing mission with a probe was launched with the same initial universe state as the rocket simulation. The probing mission was limited to a year of simulation time. Therefore, the best case result would be the probe entering Titan's orbit within a year.
Through Newton's method, the probe was easily capable of entering Titan's orbit at the last day of simulation around 100.000 meters with an initial velocity of around 45.000 meters per second. Even though this is a positive sign for the mission itself, it was not for the rocket's mission.

For the rocket to reach an initial velocity of around 45.000 meters per second, it would take a relatively long burn time to reach this velocity while also requiring huge amounts of fuel. Therefore, this was not a feasible solution and a different simulation in Newton's method had to be run.

After a few attempts and reasoning, the conclusion was made that the rocket should only have a fuel mass of around 700.000 kg and enter Titan's orbit within 12 years as both together represented a more realistic mission. Running Newton's method in combination with Euler's method for a final evaluation time of 12 years, resulted in a velocity of around 6000 meters per second which allows the rocket to arrive at Titan's orbit after 12 years.

3.5 Wind Model

3.5.1 Linear Regression Model

Titan has a specific climate. Considering that the wind is an important factor which can cause serious deviation to the lander, in order to simulate the main forces acting on the lander, a stochastic wind model is created.
The European Space Agency (ESA) Huygens Probe landing mission on Titan served as a basis for establishing a wind model for this mission. Once reached the atmosphere of Titan, the ESA's probe started its descent through Titan's hazy cloud layers.
A dataset was extracted from the Huygens Doppler Wind Experiment obtained during the Huygens Mission at Titan. The following dataset contains the wind speed at different altitudes during the landing of the space probe. The observations show that the wind speed decreases

as the altitude decreases. Furthermore, using Linear Regression for the previously obtained data, a function that provides the wind speed of a given altitude was written.

To maintain the randomness of the wind, and possible change of direction, as it happened twice during the Huygens mission, the wind model is made stochastic. Hence, the wind speed is multiplied by a random number between 0.8 and 1.2. In other words, it can deviate 20 percent from the calculated wind speed. Moreover, If the number is positive, the direction of the wind points eastward. Another random factor is added, which has a 1/100 chance of changing direction, thus the sign can be changed.

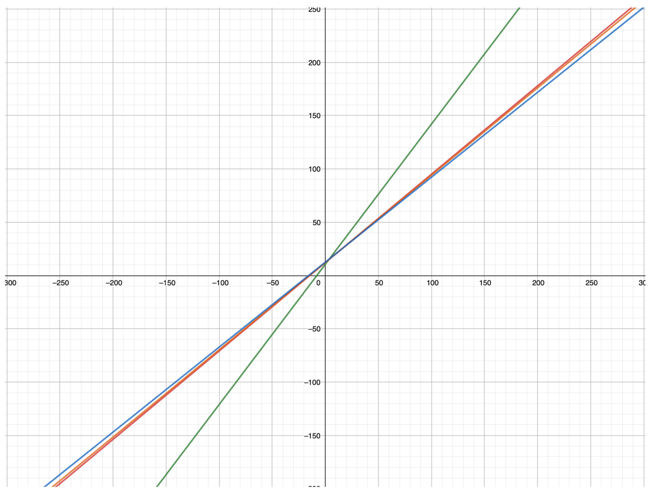


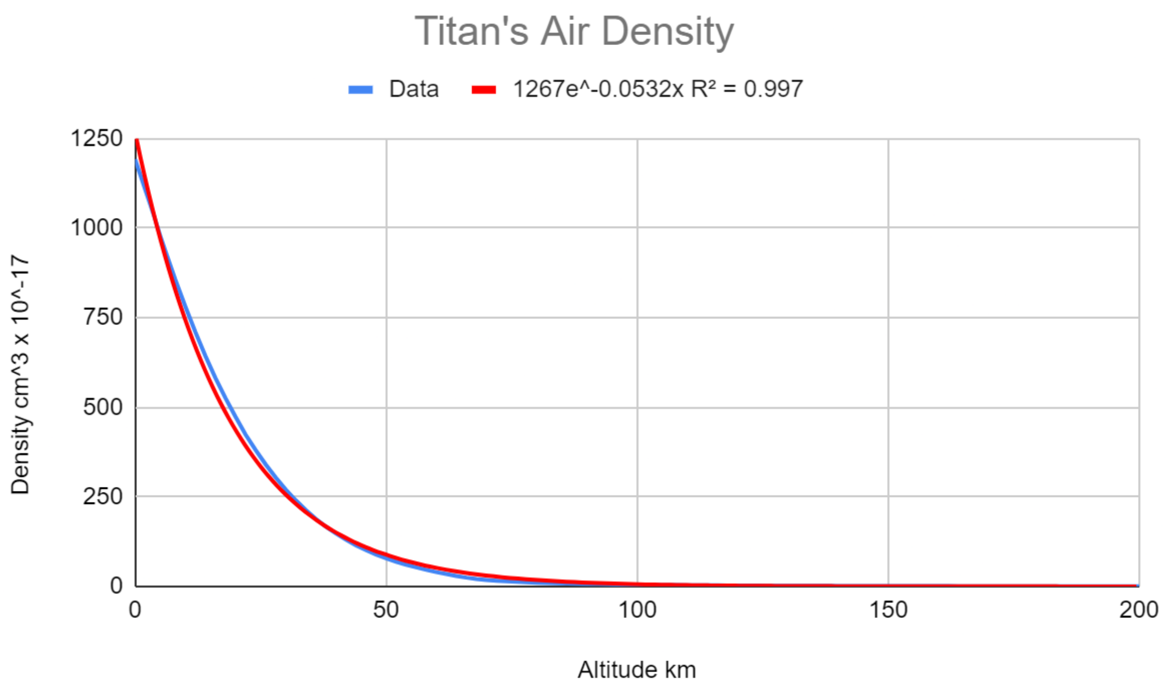
Figure 1 : Linear functions representing the relationship between the wind speed and altitude with the added randomness factors.

The slope of the obtained linear function, the constant of the function, and the Drag Equation were employed to convert the wind speed at any altitude to a force that will be applied on the rocket's X-axis during the landing. We assume that the wind direction is only horizontal.

The constructed wind model plays a crucial role for the closed-loop controller which will consider the force generated by the wind on the landing spaceship, as this will be further discovered in the next section.

3.5.2 Titan's Air Density

The drag coefficient was taken from a 2007 article by Striepe et al. which analysed the trajectory of the Huygens Titan Probe. In the article a drag coefficient of 0.8 was used. For the density values data was taken from a 1983 paper by Lindal et al. The paper examined radio occultation measurements from the Voyager 1 probe. This data was mapped to an exponential function giving a R^2 value of 0.997. Using this equation it was possible to calculate a density value for Titan's atmosphere at any height between 200 - 0km.

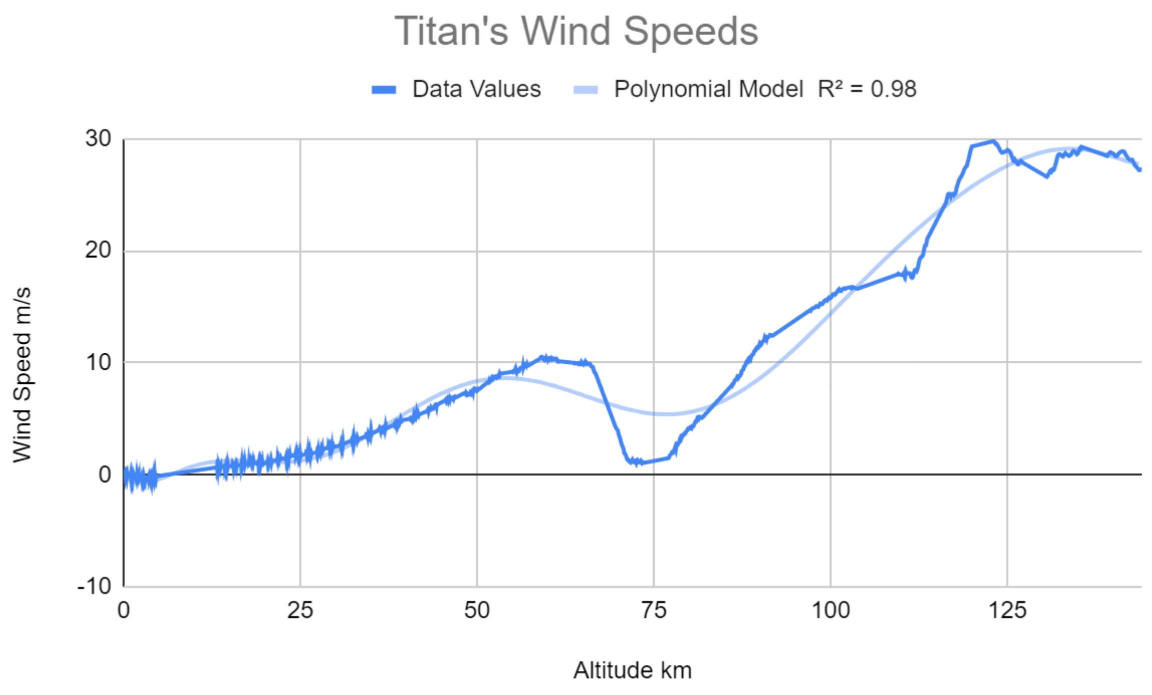


Equation for air density above Titan:
ρ= Air Density (kg/m^3)
x = Altitude (m)

$$\rho = 1267000e^{-0.0532xE^{-17}}$$

3.5.3 Polynomial Model

To improve the wind model the data taken from Huygens Doppler Wind Experiment **REF** was fitted to a degree 10 polynomial model. This improved the R^2 value from 0.875 to 0.98.



Note: the data ranges from 0-144 km above Titan. As such, wind speed values were not taken at altitudes above this range.

The wind speed polynomial is given by the following equation:

V_w = Wind speed (m/s)

x= Altitude above Titan (km)

$$V_w = 0.0215 + -0.67x + 0.186x^2 + -0.0174x^3 + 7.97E-04x^4 + -2.04E-05x^5 + 3.09E-07x^6 + -2.87E-09x^7 + 1.6E-11x^8 - 4.91E-14x^9 + 6.42E-17x^{10}$$

3.5.4 Wind Force Equation

For both wind models the following equation was used to calculate wind force generated:

F_w = Wind force (N)

ρ = Air density (kg/m^3)

A = Cross Sectional Area (m^2)

V_w = Wind speed (m/s)

$$F_w = \frac{1}{2} \times \rho \times A \times V_w^2$$

3.6 Feedback Controller

In order to perform a safe landing, an open-loop controller and a feedback controller were implemented, taking into consideration multiple external forces acting on the lander.
This section discusses the closed-loop controller known as the feedback controller.
The principle behind a feedback controller is to measure the current state of a system. The difference between the current state and the desired state is then calculated, this is called the error. With the error, a response is generated to be applied correcting the state of the system. Once the response has been applied (in this case through the lander's propulsion system) the state of the system is measured and is compared again to the desired state, starting the next loop of the control system. (Woolf, n.d.)

The inputs which describe the landing system's state are as follows:

1. X-position.
2. Y-position.
3. θ the angle of the lander.
4. X-velocity.
5. Y-velocity.
6. θ' angular velocity of the lander.
7. Wind force acting on the x-direction.
8. Drag force acting of the y-direction.

3.6.1 Lander Specifications

The output of the lander control system is the thrust produced by the landers' engine. For this the max thrust of the engine was based on the Apollo lander descent engine (LM Descent Engine), which produced a max thrust of 42923 N (Scott, 2007). This can only be applied in the opposite angle to the direction the lander is pointing towards. Additionally, the lander was designed with two RCS (Reaction Control System) thrusters. These are placed on opposite sides of the lander in line with the center of mass at a distance equal to the radius of the lander. Each thruster was assumed to be capable of producing a thrust of 500 N in both the Y+ and Y- direction. The main purpose of the RCS thrusters is to control the inclination of the lander by applying torque perpendicular to the center of mass of the lander. ****.

3.6.2 Representation

For the landing controller, the center of the system (0,0) was Taken to be the center of Titan. Additionally, the center of mass of Titan was placed at this point. This is important as if the lander's position is -X and -Y the resultant gravitational force will be applied with a positive magnitude in both the X vector and Y vector.

The orientation of the lander was represented in radians and was taken as an absolute relative value. Meaning, if the orientation value was greater than **τ** the value was subtracted by **τ**. Similarly, if the value was less than 0 the **τ** was subtracted by the value to find the through angle.

3.6.3 Gravitational & Wind Force

The principal environmental force acting on the X and Y position and velocity is acceleration due to gravity. To find this first, the force of gravity acting on the lander needs to be calculated. This is done using Newton's law of gravitation.

F = Force N

G = Gravitation constant

M_1= Mass of Titan kg

M_2= Mass of the lander kg

R= Distance between the two bodies m

$$F = \frac{GM_1M_2}{R^2}$$

The R value is calculated by taking the norm of the X,Y position of the lander. After finding the gravitational force, the next step is to find the X and Y component of this force. In order to execute this first, the angle of action of this force needs to be calculated using the equation:

Θ = Angle of action (rad)

X = X position of the lander (m)

Y = Y position of the lander (m)

$$\theta = tan^{-1}(\frac{X}{Y})$$

Subsequently, the X component and Y component of this force can where found using the following equations:

F_g = Gravitational force (N)

F_x = X component of the force (N)

F_y = Y component of the force (N)

Θ = Angle of action (rad)

$$F_x = F_g \times cos\,(\theta)$$

$$F_y = F_g \times sin\,(\theta)$$

The F_x and F_y values are input as either negative or positive depending on the position of the lander's X and Y position. This is to ensure that the force is always acting towards Titan.

```
if (posX > 0 && posY > 0) {
   xForce += phy.xComponent(-gravity, gravity_Angle);
   yForce += phy.yComponent(-gravity, gravity_Angle);
}
else if (posX < 0 && posY > 0) {
   xForce += phy.xComponent(gravity, gravity_Angle);
   yForce += phy.yComponent(-gravity, gravity_Angle);
}
else if (posX < 0 && posY < 0) {
   xForce += phy.xComponent(gravity, gravity_Angle);
   yForce += phy.yComponent(gravity, gravity_Angle);
}
else if (posX > 0 && posY < 0) {
   xForce += phy.xComponent(gravity, gravity_Angle);
   yForce += phy.yComponent(gravity, gravity_Angle);
}
```

Next, the wind component forces need to be added to the gravitational components. The wind function discussed in the wind section is used to generate a wind force and angle of action. The component equation used for the gravity is used again to find the X and Y components of the wind force. Though there is a change in the implementation, as the wind angle produces a force in the opposite orientation. Therefore the inputs were generated as so:

```
if (windAngle > Math.PI && windAngle < 1.5*Math.PI) {
   xForce += phy.xComponent(windForce, windAngle - Math.PI);
   yForce += phy.yComponent(windForce, windAngle - Math.PI);
}
else if (windAngle < 2*Math.PI && windAngle > 1.5*Math.PI) {
   xForce += phy.xComponent(-windForce, 2*Math.PI - windAngle);
   yForce += phy.yComponent(-windForce, 2*Math.PI - windAngle);
}
else if (windAngle < Math.PI && windAngle > 0.5*Math.PI) {
   xForce += phy.xComponent(windForce, Math.PI - windAngle);
   yForce += phy.yComponent(-windForce, Math.PI - windAngle);
}
else if (windAngle < 0.5*Math.PI && windAngle > 0) {
   xForce += phy.xComponent(-windForce, windAngle);
   yForce += phy.yComponent(-windForce, windAngle);
}
```

3.6.5 Drag Force

Finally the drag forces need to be added to component X and Y forces. To find the drag force the following equation was used:

F= Drag force N

C_d= Drag coefficient

A= Reference area m

ρ= Density of the air kg/m^3

V= Velocity of the lander m/s

$$F = C_d \times A \times \frac{\rho V^2}{2}$$

As the drag force can be calculated in component form by setting the Vvalue equal to the X or Y velocity to find the respective drag force. The A value was additionally unique for the X and Y component. As the drag force acts as a reaction to velocity of the lander the force generated acts in the opposite direction to the landers velocity.

Following the summation of the environmental forces acting on the lander the resultant force and angle of action can be calculated using the following equations:

F_r= Resultant force acting on the lander (N)

F_x= X component force (N)

F_y= Y component force (N)

θ = Angle of action (rad)

$$F_r = \sqrt{F_x^2 + F_y^2}$$

$$\theta = tan^{-1}\left(\frac{F_y}{F_x}\right)$$

The resultant force is used to find the required thrust force to counteract the environmental forces acting on the lander. While the θ is used to find the appropriate orientation for the thrust to be applied.

3.6.6 Lander Thrust & Acceleration

Finally, the acceleration components can be calculated using Newton's second law:

m= mass of the lander kg

a= acceleration m/s^2

$$F = ma$$

If a corrective thrust is applied the resultant acceleration can be found using the following equation:
m= mass of the lander kg
a= acceleration m/s^2
F_e= Environmental force (N)
F_t= Thrust force (N)

$$a = \frac{F_e - F_t}{m}$$

3.6.7 Lander Mass Balance

The mass flow rate for the RCS thruster was taken from a 1994 article by Borchers et al. which examined the YAV-8B reaction control system. From this, the mass flow rate of the RCS thruster was assumed to be a constant of 5.352856 kg/s. This was deemed appropriate as in this implementation the RCS thrusters are designed as impulse thrusters. Meaning they don't throttle but exert 100% of their force when used. So by multiplying the mass flow rate by the activation time, the change in mass of RCS fuel can be calculated.

For the main thruster, the mass flow rate was taken from a 2014 article by Smirnov et al. which simulated hydrogen fuelled rockets. Again, even though it is unrealistic, the main thruster too was treated like an impulse engine in this implementation. Therefore, the constant mass flow rates were taken as follows:
Δm_{O2}= Oxygen mass change (kg)
Δm_{H2}= Hydrogen mass change (kg)
Δm_L= Lander mass change (kg)
h = Burn time (s)

$$\Delta m_{O2} = 0.0904 \times h$$
$$\Delta m_{H2} = 0.0331 \times h$$
$$\Delta m_L = \Delta m_{O2} + \Delta m_{H2}$$

3.6.8 Adjusting Inclination

To adjust the inclination of the lander. First the Δangle is found by subtracting the current angle from the desired. Next this interval is divided by 2. This is because for the first interval the lander is accelerating. Then for the next interval the lander is decelerating, in order to stop at the desired inclination.

To find the acceleration first the torque is found:

$$\tau = F_t \times r$$

Next the inertia moment is found using:

$$I = m \times r^2$$

Finally the acceleration is found using:

$$\alpha = \frac{\tau}{m}$$

The final velocity is found using:

$$V_f = V_i^2 + 2\alpha \times S$$

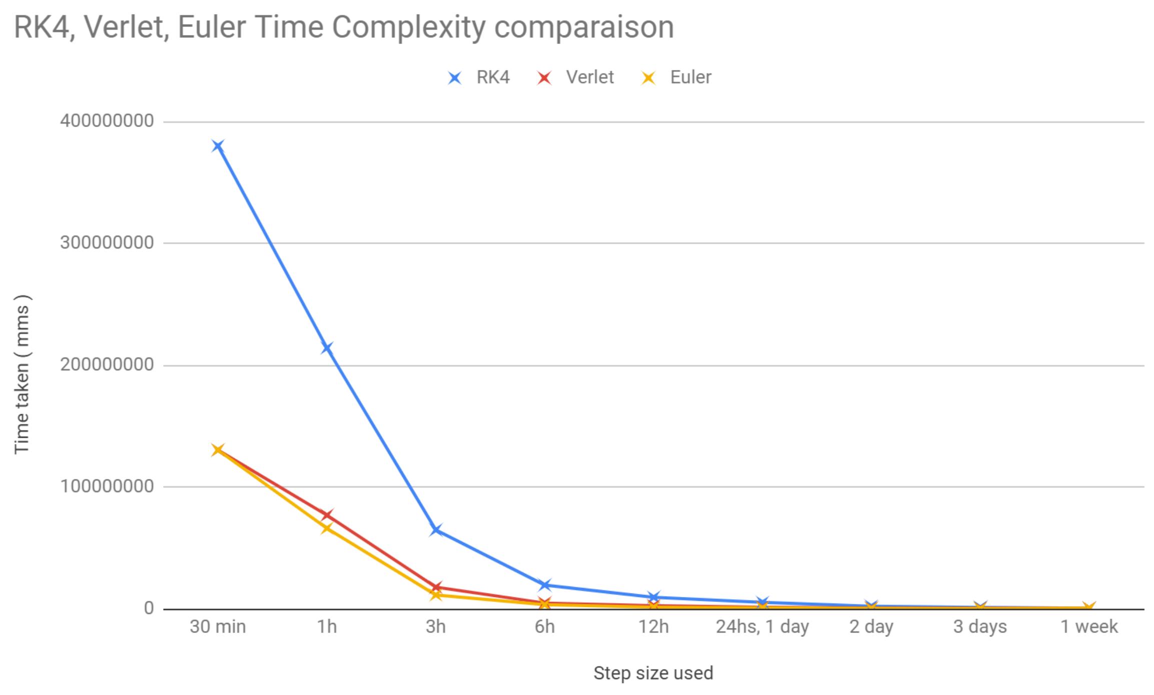
Displacement is found using:

$$S = \theta + (V \times h) + (0.5 \times \alpha \times h^2)$$

5. Experiments

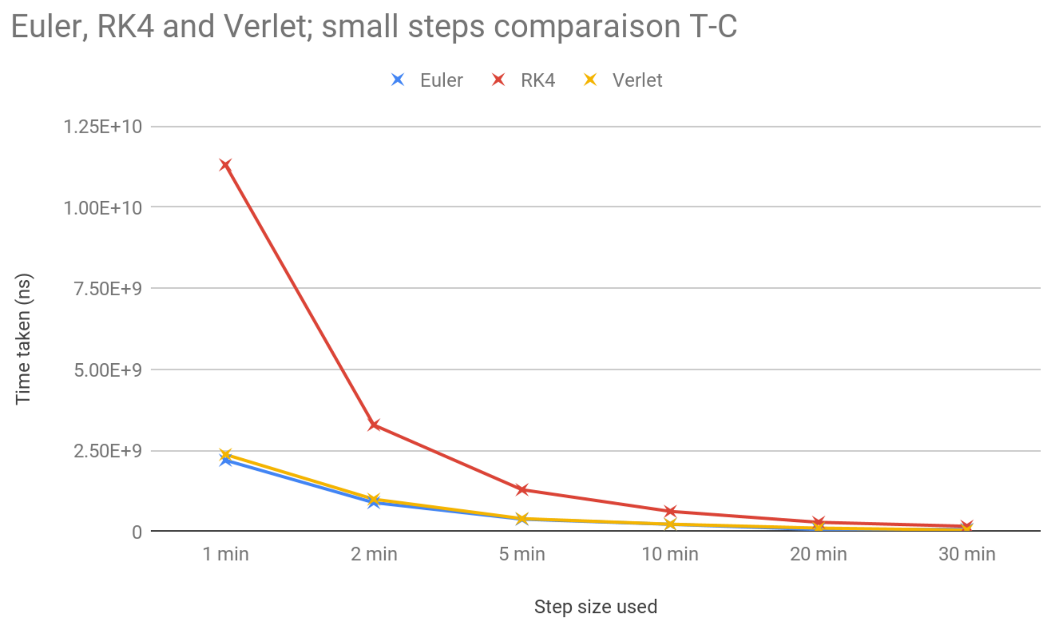
5.1 Time Complexity

The speed of computation is a key factor in our performance. It can also indicate the computational effort or cost that the algorithm requires, we must then consider the cost of the solvers we would implement for our spatial journey.
All solvers were independently executed to simulate our solar system for a year. Step sizes of 30 mins, 1 hour, 3 hours, 6 horse, 12 hours, 1 day, 2 days, 3 days and 1 week were tested. The time taken for every solver at different step sizes was recorded in milliseconds, they were plotted in the following graph;



The method of fourth order of Runge-Kutta (RK4) demands a higher computational time relative to the others, this could be explained since RK4 involves four different function evaluations per step. Furthermore, Euler's method being of first order requires little computation time compared to RK4, and Verlet's integration curve is pleasantly close to Euler's.

The same trial was made but with smaller step sizes such as 1 minute, 2, 5, 10 and 20 minutes. In order to achieve a higher precision, nanoseconds were used as the standard unit for this experiment. As we can see, both of the graphs have similar findings, RK4 involves higher computational effort and Euler and Verlet have similar and much smaller marks.

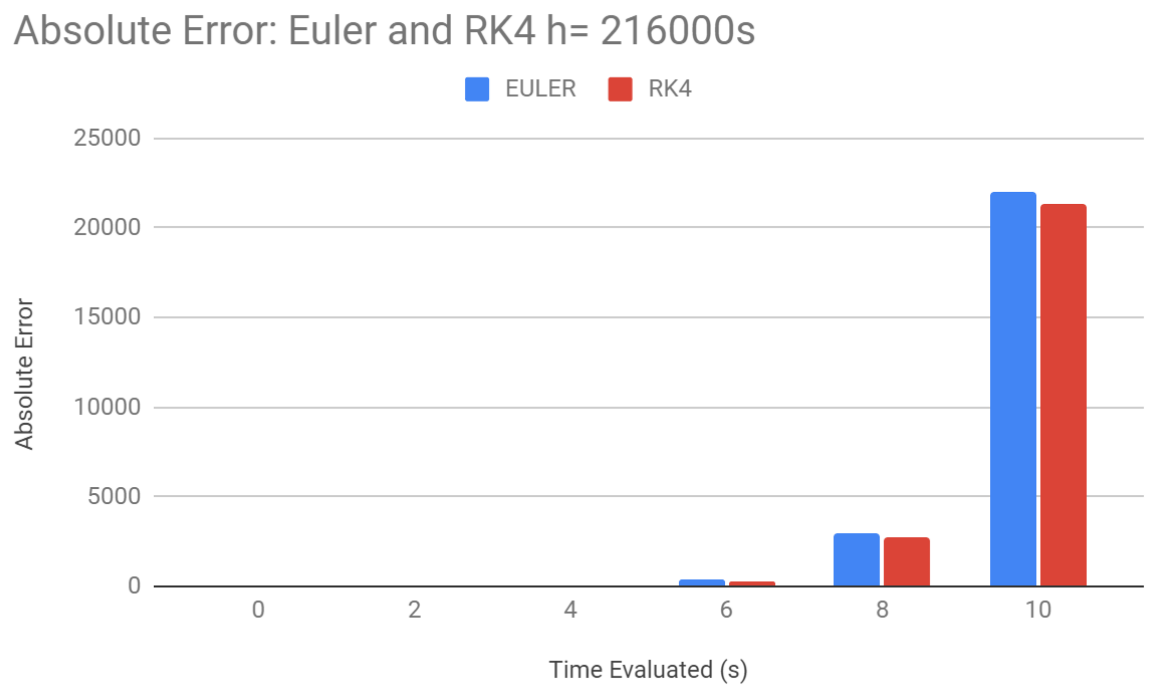


Moreover, it can be distinguished in both of the graphs, exponential looking-like curves for all experiments tested. This suggests the extra computational effort required by reducing through a small magnitude the current step size. For RK4 this "extra effort" is much bigger, indicated by the slope of the curve; its gradient is much higher and the gradient of the curve of Euler's or Verlet . As the magnitude of the step size increases the computational time needed reduces and the gradient of the slope to the next step size also decreases.

5.2 Accuracy

Although the time taken for each step is important to bear in mind, the accuracy of our results is equally or even more important than the computational time needed for each of the algorithms. In order to test for the accuracy of the different methods, firstly, an analytical solution is needed. There are certain ways to obtain a comparable exact solution; either from real life data such as the NASA Horizon's database of the solar system or using an exact answer of a function calculated by hand or by a very precise program such as MATLAB. It was considered using NASA Horizon's data but the magnitude of errors owing to modelling and essential presumed assumptions was unknown to us, and hence, the tolerance authorized for our final results could not be established. For this part, an exact solution of one of the exercises of our subject Numerical Mathematics was used as our analytical solution, the exact results of the derivative of the function $y = e^{-t} - y^{2}$ at different time intervals were attained. These were then compared to the results obtained by each of the solvers. Conversely, Verlet integration is a numerical integration method and not a numerical differential equation such as Runge Kutta and Euler. Differentiation involves taking

differences between function values, and numerical integration involves the addition of the function at different times. Hence, the Verlet integration method cannot be applied to this experiment.
The following graph shows the comparison between the absolute errors of the calculations by RK4 and by Euler's method. The absolute error is the miscalculation between the exact solution, p, and the solution of our solver, p', calculate through the following $\left|p - p'\right|$.
Many different step sizes were tested for this same experiment, we show the one with a step size of an hour which was our "standard" step size.

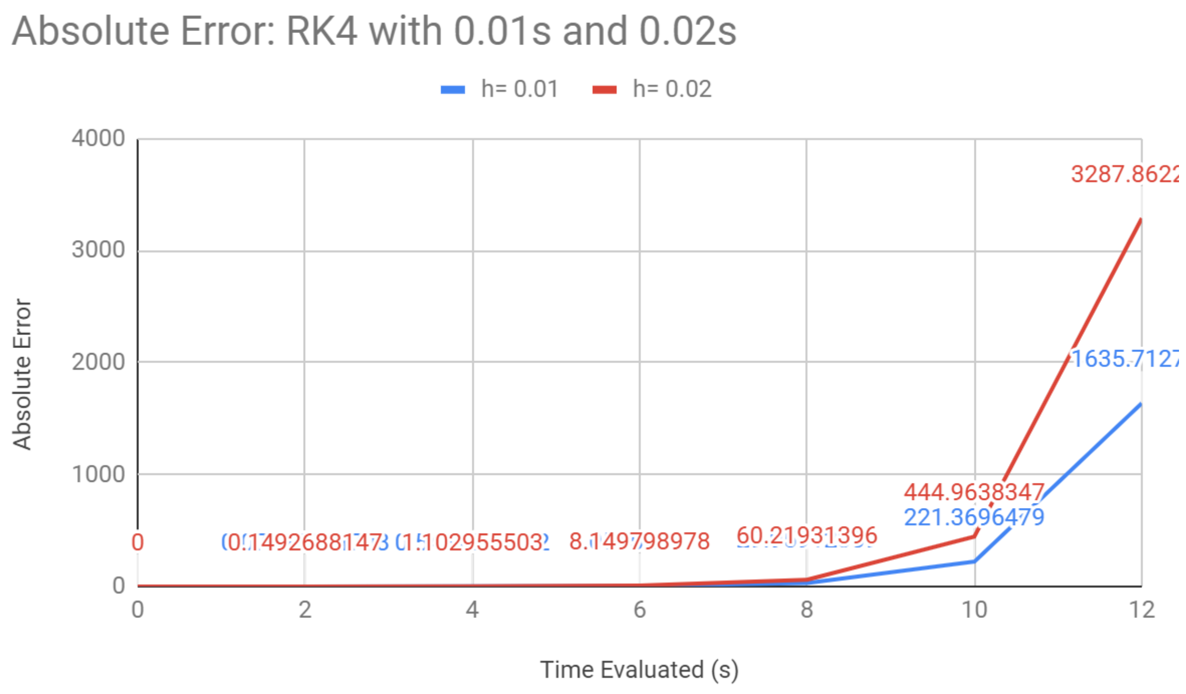


As we can see in the previous graph, the error produced by RK4 and Euler is very similar in spite of the difference in theoretical global error of accuracy of the methods.

RK4 (fourth order Runge-Kutta) is said to be a fourth order method. The definition of the global error of accuracy of a numerical method follows the following; if a numerical method is said to have a global error of accuracy p if:

$max_{n \in \{0, 1, .., T/h\}} \left|v^n - u^n\right| \leq \theta(h^p)$ as $h \to 0$, where $\left|v^n - u^n\right|$represents the absolute error of the results obtained by the numerical method (Chap 2, MIT) . Taking a step further, this implies that if the step size is per se, halved, then we would expect the error to reduce by a factor 2^p, where $p$is the method's global error of accuracy, i.e the method's order.
Taking our example of Runge Kutta that has a fourth order of global accuracy, by halving the step sizes, we would obtain an error diminish by a factor of 2^4.
However, these statements hold in the limit $h \to 0$, thus, using large step sizes we will be unlikely to observe this behavior. Subsequently, a small experiment was performed to display

the behavior of RK4's order method as the step size is halved. By using 0.01 seconds as step size and 0.02 seconds and comparing the absolute errors while evaluating the derivative of the function $y = e^{-t} - y^2$.



Even though there is a clear difference between the accuracy result of step size 0.01 and step size 0.02, the ratio between them does not correspond to RK4's order. It can be seen from the data labels that the absolute error approximately halves between the marks of the respective step sizes, where it should be divided by a factor of approximately 32 (2^4) and not 2. Meaning that our implementation of RK4 does not match its theoretical performance.

Furthermore, RK4 wasn't succeeding our JUnit test even in the early evaluations. It repetitively failed, giving the correct last integer(s) of the result. It has been considered that this problem arises because of the limited representation of the double type in Java's programming language, especially during the addition of the intermediate steps to calculate the next state. In the discussion, we will propose an alternative method to eliminate these rounding off errors.

5.3 Wind Experiments

In order to test the randomness of the wind, several experiments were conducted at different altitudes on Titan's atmosphere. By assumption, the wind can deviate 20 % from the wind force exerted on the ESA Huygens probe during the landing.
To verify the hypothesis, the following data was extracted :

15 Km

Slope	y = 1.3148863027144424x + (10.450161934162796)	Relative error
Altitude (Km)	15	NA
Original wind Force	8740,19976615786	0
Stochastic Wind Force 1	10779,3483527931	2,33306862679597
2	30833,8901723203	25,2782442018195
3	31994,1947710458	26,6057934910454
4	-7983,03903840494	19,1337031784049
5	30899,5265140631	25,3533412745397
6	25230,9777406872	18,8677357677588
7	39490,985742363	35,1831614825012
8	13863,1416195576	5,86135556447556
9	21927,2137227033	15,0877718008296
10	-25369,9046447522	39,0266874024833
11	9485,96430517803	0,853258002074252
12	34291,0116212559	29,2336703264278
Average	17244,8854342283	20,2348159265963

The following two figures represent the wind forces acting on the lander. Multiple models were applied to examine the randomness factors. The average of all the models was taken into consideration for concluding.
Firstly, the slope characterizes the relationship between the altitude and the wind speed applied on the Huygens ESA probe during its descent.

The wind speed was randomised by 20%, as the wind speed directly affects the wind force, the aim of this experiment is to examine the randomisation influence of the wind speed on the force.
At an altitude of 100 km, a relative error ranging from 1% to 55 % from the wind force executed on the Huygens probe is observed.
394 890 N is the average wind force, while the average relative error holds at 20%, which meets the hypothesis made earlier for randomising the wind speed.

At an altitude of 15 km, a relative error ranging from 0.8% to 35 % is remarked. The average relative error remains at 20%.
A significant drop in the force acting on the lander is perceived with an average of 17 244 N. We can confirm that the previous conclusion in the wind section; the wind speed decreases as the altitude decreases; holds.

Finally, we can deduce that the hypothesis stating that there is a 20% chance of force deviation applies to our wind model.

6. Discussion

Within the discussion section, the results are evaluated based on the efficiency measure and accuracy.
The distances separating the celestial objects are astronomical, therefore large step sizes were employed, implying that the inaccuracies of the methods accumulate, getting larger and larger. In addition, the error term decreases its magnitude as the order of the method increases whenever the step size is smaller than 0. Multiplying by a value lower than 0 will decrease the result. But in this experiment we use step sizes of hours and sometimes even days! Having such large step sizes doesn't only increase the truncation error, but it also involves major rounding off errors that accumulate at every operation performed.

Taking this into account and knowing the Runge Kutta requires a high computation effort,it was decided to opt for Euler's method to perform our simulation. Throughout the previous experiments we can conclude that Euler's does not take too much computational time (neither memory, this was another big constraint implementing RK4) and the results obtained are similar to RK4. Since it was not possible to test for Verlet's integration accuracy, we decided to not use it either as the general performance of the Euler solver was satisfying.

Further improvements:

In order to reduce the rounding off errors produced by the limited representation of the numbers, we would like to aspire to reproduce a number with a mantissa such that all its bits would coincide with those of the mantissa of the exact solution. The exact solution might be a number that contains an infinite number of bits but it will be rounded off to produce a floating point number with a mantissa A (Shmuel Gal) . Hence, achieving last bit accuracy if both mantissas are equivalent. In the IBM 370 using single precision, yielding last accuracy bit is fairly easily controlled as it contains 32 bits in the mantissa. However, using double precision for extensive calculations leads to a degradation of performance that accumulates through use.

In pursuance of increasing the accuracy of our results generated from our solvers, we should firstly range reduce our function to such interval $A \leq f(x) \leq B$ for all x's, the values of these x's are also between A and B. And the difference between each point should correspond to a constant d, whose value is predefined. Secondly we should approximate our original function by a minimax polynomial; the smaller d is the smaller the minimax polynomial's degree will be, reducing the computation time and also leading to higher accuracy as all computations are small and continuous.

Nevertheless, for an optimal performance, the set of x points has to be unequally spaced; such as $Xi = Xi - 1 + i * d + Ei$; leading to an overall higher accuracy. These points Xihave a particular characteristic; whenever used with the approximating polynomial(s) their results will always contain all 0's or all 1's in the 57-65 bits of the mantissa. The latter can be achieved by adding Ei to the value. Ei is such a small number that won't alter the minimax poly in any noticeable way, but still modify the last bits of the representation, so that when evaluated, the last eight states of the bit representation are the same.

Depending on our range of our function, it can be estimated how many numbers do indeed have this characteristic; $K = \frac{d}{1024*w}$ where w is the difference of two consecutive representable numbers (this changes depending on the interval). The probability that the particular feature holds is about 2^{-8}, by computing K it would be known how many of these numbers could be chosen to approximate our function

Then, a table of values would be created with Xis and their result when evaluating the approximating polynomials, all these results are expected to have been mono-number at the end of the mantissa, as commented before. This table would be only generated once at the beginning and used throughout the simulation. The memory needed for this table is defined by the magnitude of d, a practical value for it is $\frac{1}{256}$.

7. Conclusion

Even though higher-order differential equation solver Runge-Kutta of the fourth order was implemented and the Verlet integration to assist in simulating the universe, Euler's method was in the end preferred over them. As, the computation time for Euler's method and both its global and local truncation error were smaller than the other two for large step sizes such as hours and a day.

Furthermore, both the probing mission and the rocket's mission to Titan turned into a success through the integration of Newton's method. This gave an opportunity to simulate and display three different universes, namely a universe with only the planets, the universe with the probe and the universe with the rocket.

These mission successes lead to an attempt to display the benefits of having a GUI to display the simulation which made us come across some tenacious hurdles such as having a GUI in javafx was harder to manage and expand. Moreover, the GUI was limited to a 2-dimensional display instead of 3-dimensional display which made the simulation lose its beauty more.

So, all in all, the project has been a pleasant journey of learning, studying and applying knowledge obtained throughout courses and experimentations.

8. References

Striepe, S. A., Blanchard, R. C., Kirsch, M. F., & Fowler, W. T. (2007). Huygens Titan Probe Trajectory Reconstruction Using Traditional Methods And The Program To Optimize Simulated Trajectories II (aas 07-226). *Advances in Astronautical Sciences, 127, 1853–1880.*

Lindal, G., Wood, G., Hotz, H., Sweetnam, D., Eshleman, V., & Tyler, G. (1983). The atmosphere of Titan: An analysis of the Voyager 1 radio occultation measurements. Icarus, 53(2), 348-363. doi: 10.1016/0019-1035(83)90155-0

Scott, D. (2007). Lunar Module Descent Stage Rocket Engine Thrust Chamber. Retrieved 15 June 2021, from https://www.apolloartifacts.com/2007/09/tr-201-bipropel.html

Borchers, P., Moralez III, E., Merrick, V., & W. Stortz, M. (1994). YAV-8B Reaction Control System Bleed and Control Power Usage in Hover and Transition. National Aeronautics And Space Administration, Ames Research Center. Retrieved from https://ntrs.nasa.gov/citations/19940030488

Smirnov, N. N., Betelin, V. B., Shagaliev, R. M., Nikitin, V. F., Belyakov, I. M., Deryuguin, Y. N., Aksenov, S. V., & Korchazhkin, D. A. (2014). Hydrogen fuel rocket engines simulation using logos code. International Journal of Hydrogen Energy, 39(20), 10748–10756. https://doi.org/10.1016/j.ijhydene.2014.04.150

Woolf, P. 11.1: Feedback Control. Retrieved 15 June 2021, from https://eng.libretexts.org/Bookshelves/Industrial_and_Systems_Engineering/Book%3A_Chemical_Process_Dynamics_and_Controls_(Woolf)/11%3A_Control_Architectures/11.01%3A_Feedback_control-_What_is_it%3F_When_useful%3F_When_not%3F_Common_usage.

Meyer-Vernet, N. (2007). Basics of the solar wind (Ser. Cambridge atmospheric and space science series). Cambridge University Press. Meyer-Vernet, N. (2007) Basics of the Solar Wind. University Press, Cambridge. - References - Scientific Research Publishing (scirp.org)

Tokano, T., Lorenz, R., & Lebreton, J. The way the wind blows on Titan. Retrieved 8 June 2021,https://www.esa.int/Science_Exploration/Space_Science/Cassini-Huygens/The_way_the_wind_blows_on_Titan

Drag equation, Wikipedia, Retrieved 14 June 2021, https://en.wikipedia.org/wiki/Drag_equation

Convergence of Numerical Methods, MIT Chap 2 Numerical methods, Retrieved 18 June 2021 *Chapter2.pdf (mit.edu)

Shmuel Gal, Computing elementary functions: a good approach for achieving high accuracy and good performance, Retrieved 10 June 2021, IBM Israel Scientific Center, Computing elementary functions: A new approach for achieving high accuracy and good performance (unimaas.nl)

Wind data extracted from the Huygens Doppler Wind Experiment (DWE) obtained during the Huygens Mission at Titan.
https://pds-atmospheres.nmsu.edu/data_and_services/atmospheres_data/Huygens/access_dwe.html