A Titanic Space Odyssey!

Project 1-2

Report

DACS - Maastricht University

Academic Year 2022/2023

Group 17:
Adel Asaad,
Raul Boekhoudt,
Miriam Espinosa Solana,
Vasile Mereuta,
Arthur Oganyan,
Jakub Suszwedyk,
Horia Ionescu.

Abstract

This report presents a comprehensive comparison of various numerical methods used in the computer simulation of the solar system and the planning of a space mission from Earth to Titan. A detailed understanding of the dynamical behaviour of celestial bodies is fundamental for the study and prediction of astronomical events. Accurate and efficient simulation requires the implementation of appropriate numerical methods to solve the differential equations in charge of the motion of planets, asteroids, and other objects in the solar system.

The report focuses on the comparison of three widely used numerical methods: the Euler method, the Adam-Bashford method and the Runge-Kutta method. Results obtained by applying the algorithms to first-degree differential equations and to the simulation of the motion of celestial bodies are analysed. In addition, variations of these methods, such as the Ralston method or higher-order Runge-Kutta methods, are explored. The advantages and disadvantages of each method in terms of implementation complexity and run time are also analysed. The results obtained show that the higher-order Runge-Kutta method offers the highest accuracy and stability, although it has a longer run time.

In addition, numerical optimization methods applied to space mission planning are investigated, such as the Hill Climbing algorithm, which allows finding optimal solutions for trajectory planning, fuel saving in a mission, and the use of the Hill Climbing algorithm in space mission planning. We have also implemented a wind module based on the different air flows in Titan's atmosphere. We aimed to study how this module affects the different controller types and we concluded that, due to the random nature of the wind flow, an Open-Loop controller is not able to land at a pre-set point after being affected by it. So a FeedBack controller would be a better approach for this purpose.

Table of Contents

1. Introduction

Exploration of the outer planets and their moons has long been an ambition of space agencies around the world. Titan, one of Saturn's largest moons, has emerged as a fascinating target for scientific research in recent years due to its unique characteristics and potential for hosting life (Carter (2022)). As a result, preparing an expedition to Titan has piqued the interest of the space exploration community. This report goes over the most important aspects of planning a mission to Titan, such as mission objectives or trajectory planning. Overall, this project aims to create a simulation of the universe, based on solving the differential equations that define the motion of celestial bodies, and to design a trajectory that will take a rocket from Earth to Titan while optimising the distance and fuel consumption. Both aspects have been widely studied in the field of computer science, giving us a reliable foundation for our project. As a result, the methods that will be discussed throughout this report are our implementation of these numerical methods or optimisation algorithms

1.1 Problem Statement and Research Question

A mission to Titan, one of Saturn's largest moons, is a complex and difficult undertaking that necessitates sophisticated software systems and excellent engineering capabilities. Furthermore, the spacecraft's limited computational power and memory capacity pose additional challenges in designing and implementing software systems that can operate efficiently in space environments (Gianluca, Meoni & Moloney (2020)). Therefore, the issue this project attempts to solve is the creation of a software system using the Java programming language that can facilitate effective mission planning to Titan using different equation solvers. Throughout this report we will try to answer the following research question:

1. To what extent are some numerical methods more efficient than others in solving an Initial Value Problem?
2. How does the wind in Titan's atmosphere affect the landing process of a space probe?

1.3 Our Methodology

In order to simulate the universe, we have used Euler's method to solve the differential equations that define the interaction between the bodies in the solar system. We have compared the results with different time steps for the following methods : Runge-Kutta, LeapFrog and Adam- Bashford.

On the other hand, to trace the rocket trajectory we have implemented a genetic algorithm (Zhou, Gao, Fang, & Lan (2021)), but due to its random nature, the algorithm becomes unpredictable especially at high velocities. Therefore, we opted for hill climbing (Chalup & Maire (1999)), a local search algorithm that works iteratively by, starting from an arbitrary solution, tries to improve by introducing gradual changes until the most optimal solution is found. We have also developed a simple greedy algorithm.

Landing on Titan required controllers on the ship. In this project we have opted for the Open-Loop and Feedback controller. We also simulated the Titan atmosphere with a random component that models the wind. After doing this we discovered that the Open-Loop controller is not able to support this as it needs a plan beforehand, so we decided to focus on the Feedback controller.

Following the Abstract, the Introduction, and this current section, the rest of the report is organised as follows: Theoretical Foundation provides the theoretical basis on which we have based the project; Numerical Solvers and Mission to Titan sections describe the different approaches to discover an answer to the research questions; Section 4 provides a description of all experiments, analyses, and simulations that have been performed in order to test and compare the algorithms; Section 5 provides the presentation of findings from the previous sections; the discussion section, which is focusing largely on the results' interpretation and the study's limitations and, finally, in the conclusion, a review about the project's key findings and potential future study directions or enhancements.

2. Theoretical Foundation

The aim of our project is to bring a manned mission to Titan and back. In order to do so, it is necessary to simulate the various forces of attraction between the different celestial bodies that constitute our galaxy, as well as a spacecraft with an engine that allows it to reach Titan. Throughout the project we will ignore relativistic considerations that affect the behaviour of the celestial bodies, simplifying our model of the universe to one that follows Newton's laws.

2.1 Law of Universal Gravity

The Law of Universal Gravity is a physical law that states that the force of attraction between two bodies with mass is directly proportional to the product of their masses and inversely proportional to the square of the distance separating them. This law was first formulated by Sir Isaac Newton in the 17th century and is one of the fundamental laws of physics. According to the law of universal gravitation, any object with mass exerts a gravitational force on any other object with mass in its environment.

This law allows us to calculate the force acting on the i^{th} mass m_i by the formula

$$F_i^G = -\sum_{j \neq i} G\, m_i m_j \frac{x_i - x_j}{||x_i - x_j||^3} \tag{1}$$

where G is the universal constant of gravity.

2.2 Fundamental Principle of Dynamics

Newton's Second law states that when a force acts on a body it produces a change in its momentum, which is proportional to that force.

$$\sum_{i=1}^{n} \vec{F} = \frac{d\vec{p}}{dt} \tag{2}$$

For the specific case where the mass of the body is invariant, we have

$$\sum_{i=1}^{n} \vec{F} = \frac{d\vec{p}}{dt} = m\,\frac{d\vec{v}}{dt} = m \cdot \vec{a} \tag{3}$$

The acceleration is the time-derivative of velocity, so is the second-derivative of position,

$$\vec{a} = \overrightarrow{v'} = \overrightarrow{x''} \tag{4}$$

where prime denotes derivation.

The equation of motion for the body can be written as a second-order system,

$$\overline{F}_i^{\,G}(t) = m_i \overline{x}_i''(t) \tag{5}$$

or as a system of coupled first-order equations

$$y_i'(t) = \frac{d}{dt}\begin{pmatrix} x_i(t) \\ v_i(t) \end{pmatrix} = \begin{pmatrix} v_i(t) \\ \dfrac{F_i}{m_i} \end{pmatrix} \tag{6}$$

2.3 Spaceship Engine

In addition to universal gravitation, our spaceship has an engine that provides an additional force $F_s^{\,T}$, giving a total force of

$$\overline{F}_s = \overline{F_s^{\,G}} + \overline{F_s^{\,T}} \tag{7}$$

In this project we will assume that the impulse *I* provided by the engine occurs in a brief interval of time, so we could approximate it to the velocity

$$\overline{v}(t + \delta t) = \overline{v}(t) + \frac{\bar{I}}{m} \quad \text{where } \bar{I} = \int_t^{t+\delta t} \overline{F}(\tau)\,d\tau \tag{8}$$

This impulse will be applied at certain times of the journey to correct the trajectory of the spaceship, also taking into account the fuel consumed while ignoring the loss of mass in the spaceship.

2.4 Landing Engine

Due to the complexity of the landing, we will consider the spacecraft moving only in two directions. In order land, we will rely on the following differential equations describing the motion

$$x'' = u\,sin\,(\theta) \tag{9}$$

$$y'' = u\,cos\,(\theta) - g \tag{10}$$

$$\theta'' = v \tag{11}$$

where prime denotes derivation, x is the horizontal position, y the vertical position, θ the angle of rotation, u the acceleration provided by the main thruster, v the total torque provided by the side thrusters, and g the acceleration due to gravity on Titan.

3. Numerical Solvers

Numerical methods are very useful for evaluating dynamical systems and predicting their behaviour over time. The approach is based on the idea that by segmenting a continuous function into small discrete intervals, the function can be approximated, thus providing a numerical approximation of the ODE solutions. A first-order differential equation is an initial value problem of the form,

$$y'(t) = f(t, y(t)), \text{ with } y(t_0) = y_0 \tag{12}$$

We will assume that this has a unique solution $y(t)$. Throughout this section we will use this form in the description of the different numerical methods implemented and h as a step size,

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

3.1 Forward Euler's method

In mathematics and computational science, the Euler method (also called the forward Euler method) is a first-order numerical procedure for solving ordinary differential equations (ODEs) with a given initial value. It is the most basic explicit method for numerical integration of ordinary differential equations and is the simplest Runge–Kutta method.

The Euler method is a **first-order method**, which means that the local error (error per step) is proportional to the square of the step size $O(h^2)$, and the global error (error at a given time) is proportional to the step size $O(h)$.

Consider the initial value problem (12) . The differential equation is approximated by

$$\frac{dy}{dt} \approx f(t, y), \text{ where } dt = h \tag{13}$$

Then

$$y(t + h) - y(t) = dy \approx hf(t, y) \tag{14}$$

Therefore

$$y(t + h) \approx y(t) + hf(t, y) \tag{15}$$

Over Euler's approach, the curve of the solution may be roughly approximated at steps of h by the tangent at each interval (i.e., by a series of brief line segments).

3.2 Runge Kutta methods

The Runge-Kutta approach is a reliable and popular approach for resolving differential equation initial-value issues. Without the necessity for high order derivatives of functions, the Runge-Kutta technique may be utilised to build high order accurate numerical methods alone.

To derive the Runge–Kutta method, we divide the interval $[a, b]$ where the function is defined into N subintervals as $[t_n, t_{n+1}]$ for $(t = 0, 1,\ldots N-1)$, integrating $y' = f(t, y)$ over $[t_n, t_{n+1}]$ and apply the mean value theorem to integrals, which states that if $f(t)$ is continuous over an interval $[a, b]$, then there is at least one point $c \in [a, b]$ such that

$$f(c) = \frac{1}{b-a}\int_a^b f(t)dt \tag{16}$$

The Runge-Kutta technique comes in a variety of orders, each of which offers a distinct level of accuracy and computing efficiency. Consider the initial value problem (12), the explicit Runge-Kutta method of order s is given by

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i k_{i,i}, \tag{17}$$

where

$$\begin{aligned} k_{i,1} &= f(t_i, y_i) \\ k_{i,2} &= f(t_i + c_2 h, y_i + h(a_{21}k_{i,1}) \\ k_{i,3} &= f(t_i + c_3 h, y_i + h(a_{31}k_{i,1} + a_{32}k_{i,2}) \\ &\vdots \\ k_{i,s} &= f(t_i + c_s h, y_i + h\sum_{j=1}^{\iota-1} a_{ij}k_j) \end{aligned} \tag{18}$$

Here the coefficients a_{ij}, b_i, c_i are provided by *Butcher tableau* [1]

Through this report we will concentrate on the **fourth-order Runge-Kutta technique**, often known as RK4. In the Runge-Kutta family, this approach is the most often employed because it strikes a compromise between precision and computing expense. The RK4 approach surpasses the preceding orders in accuracy since it uses four assessments.

Consider the initial value problem (12). The fourth-order Runge-Kutta method's main technique entails computing intermediate values based on derivatives at various times during the time step $(k_{i,1}, k_{i,2}, k_{i,3}, k_{i,4})$. The ultimate answer for the following time step is then calculated using these intermediate values (y_{i+1}) and according to the general formula (17)

[1] For further information (Sanz-Serna, Verwer & Hundsdorfer(1986))

$$k_{i,1} = hf(t_i, y_i)$$

$$k_{i,2} = hf(t_i + \tfrac{1}{2}h, y_i + \tfrac{1}{2}k_{i,1})$$

$$k_{i,3} = hf(t_i + \tfrac{1}{2}h, y_i + \tfrac{1}{2}k_{i,2})$$

$$k_{i,4} = hf(t_i + \tfrac{1}{2}h, y_i + \tfrac{1}{2}k_{i,3})$$

$$y_{i+1} = y_i + \tfrac{1}{6}(k_{i,1} + 2k_{i,2} + 2k_{i,3} + k_{i,4}) \quad (19)$$

The global truncation error of the RK4 approach is $O(h^4)$. This implies that decreasing the step size results in exponentially reduced mistakes, improving the approximation's accuracy.

To conclude this part, a useful numerical integration approach for solving ordinary differential equations is the Runge-Kutta method. The method's second, third, and fourth orders, for example, offer various degrees of precision and computing efficiency.

3.3 Fourth order Ralston's

This method is a member of the Runge-Kutta technique family. Because it finds a compromise between accuracy and computing economy, Ralston's technique is particularly well-liked. As a two-stage approach, it computes the approximation by evaluating the derivative function twice at each time step.

A variation of the fundamental Ralston's approach that provides better accuracy is the fourth-order Ralston's method. It accomplishes this by employing a framework that is comparable to the fourth-order Runge-Kutta (RK4) approach but uses different coefficient values. This is how the fourth-order Ralston's technique works:

$$a_2 = 0.4;\ a_3 = \frac{14-3\sqrt{5}}{16};\ a_4 = 0$$

$$b_{21} = a_2$$

$$b_{31} = \frac{-28891+1428\sqrt{5}}{1024};\ b_{22} = \frac{3785-1620\sqrt{5}}{1024}$$

$$b_{41} = \frac{-3365+2094\sqrt{5}}{6040};\ b_{42} = \frac{-975+3046\sqrt{5}}{2552};\ b_{43} = \frac{467040+203968\sqrt{5}}{240845}$$

$$g_1 = \frac{263+24\sqrt{5}}{1812};\ g_2 = \frac{125+1000\sqrt{5}}{3828};\ g_3 = \frac{3346+1623\sqrt{5}}{5924787};\ g_4 = \frac{30+4\sqrt{5}}{123}$$

$$k_1 = hf(x_i, y_i)$$

$$k_2 = hf(x_i + a_2h, y_i + b_{21}k_1)$$

$$k_3 = hf(x_i + a_3h, y_i + b_{22}k_2 + b_{21}k_1)$$

$$k_4 = hf(x_i + a_4h, y_i + b_{41}k_1 + b_{42}k_2 + b_{43}k_3)$$

$$y_{i+1} = y_i + g_1k_1 + g_2k_2 + g_3k_3 + g_4k_4 \quad (20)$$

The formula approximates next value y_{i+1} using current y_i plus weighted average of four increments. By adding an extra assessment point inside each time step, the fourth-order Ralston's approach enhances the original Ralston's method. It is crucial to keep in mind, nevertheless, that the fourth-order Ralston's approach still performs with less precision than the fourth-order Runge-Kutta method (Banu, Raju & Mondal (2021)).

3.4 Leapfrog integration

Ordinary differential equations (ODEs) can be solved using the leapfrog integration method, sometimes referred to as the Verlet method or the Stormer-Verlet method. It works particularly effectively for systems with conservative forces, such those seen in simulations of molecular dynamics and celestial mechanics. An effective and precise method for numerically approximating the velocity of particles or celestial bodies is leapfrog integration.

Leapfrog integration's main characteristic is that it is time symmetric (meaning taking a step forward and then backwards gives you exactly the same result as the starting point), which adds to its stability and energy conservation features; The conservation of energy is done through this process: whenever a solver, without this property of time symmetry, takes a step forward it gets a positive energy error of $+\ \epsilon$. However, when that same solver takes a step backwards the error is not $-\ \epsilon$, but rather $+\ \epsilon$ again, meaning it will not end up in the exact same position it started in but rather approximately the same. Meaning that the energy error remains positive. The Leapfrog has the ability to take a step forward and another backwards ending in the exact same position, meaning that (since the energy error is not negative) the error ϵ will be equal to 0. Leapfrog integration updates the locations and velocities at alternating half steps, unlike other numerical techniques that need both position and velocity at the same time step. This alternating pattern is helpful for simulations that demand precise long-term behaviour because it enables energy conservation over lengthy integration durations.

The following equations are used in leapfrog integration to update the locations and velocities:

$$v(t + \tfrac{\Delta t}{2}) = v(t - \tfrac{\Delta t}{2}) + a(t)\Delta t \tag{21}$$

$$x(t + \Delta t) = x(t) + v(t + \tfrac{\Delta t}{2})\Delta t \tag{22}$$

where t is the time step size, $x(t)$ is the location at time t, $v(t)$ is the velocity at time t, $a(t)$ is the acceleration at time t. The velocities are updated using the acceleration at the current time step, whereas the locations are updated using the current velocity at a half-step.

Another advantage would be Leapfrog integration's ease of use and computational effectiveness. It is simple to create and simple to include the alternating half-step update strategy into simulations. Additionally, the reduced requirement to evaluate forces and derivatives at each time step, which leads to quicker calculations when compared to certain other integration approaches, contributes to the method's computational efficiency.

3.5 Second order Adam-Bashforth

Adam Bashforth is an explicit multistep approach, which predicts the answer at a future time step based on the two previous steps. Considering the initial value problem (12), Adam Bashforth's method would obtain the answer by the formula

$$y_{i+1} = y_1 + \frac{h}{2}[3f(t_i, y_i) - f(t_{i-1}, y_{t-1})] \quad (23)$$

Indeed, most of the time when we are trying to solve an ODE we are only given initial conditions, which means that y_1 is unknown to us. To approximate it we need to use another method, the choice of which should be considered both by speed and precision. In our project we calculate the first step is done using a Runge-Kutta fourth order method which was explained earlier in the report.

The main advantage of utilising Adam Bashforth is its precision. It gives a more exact approximation than first-order approaches since it is a second-order method. It includes additional data into the forecast by using information from prior time steps, resulting in improved accuracy and stability as well as speed since values are being reused instead of recalculated.

3.6 ode45

Ode45 is an adaptive numerical method, i.e. it uses a variable step size. This method is also known as Runge-Kutta 4/5 because it creates a solution combining RK4 and RK5. The Runge-Kutta method of order 4 (RK4) is used to calculate an intermediate point in the time interval considered. From this intermediate point, an estimate of lower precision of the next point is generated. Then, the Runge-Kutta method of order 5 (RK5) is used to calculate another approximation of the function value at the next point. This approximation is then compared with the approximation obtained by the RK4 method. If the difference between the two approximations is less than a set tolerance, the calculation is accurate, and the value obtained by the RK5 method is accepted. Otherwise, the time step size is decreased, and the process is repeated, meaning that a step that doesn't meet the required tolerance is wasted and repeated with a lower step size.

4. Mission to Titan

For this mission we are implementing a rocket engine that performs corrections to the trajectory . We assume that the rocket's engine runs on nuclear power and that there is no mass exhaust.

4.1 Trajectory planning

The trajectory planning gets more difficult for interplanetary journeys since the spaceship must traverse through space while taking into consideration the gravitational interactions between different celestial bodies. Gravity-assist manoeuvres, in which the spacecraft uses the gravitational pull of planets or other celestial bodies to modify its course and gain or lose velocity, are common for such missions. The trajectory of the spacecraft may curve as it ascends owing to variables such as air conditions, orbital movements, or gravitational pulls from celestial bodies.

To ensure a successful mission, creating a trajectory for a spaceship requires complicated calculations and optimization. To address this problem, multiple strategies were applied including a genetic algorithm, hill climbing and a simple accelerated chase of Titan.

4.1.1 Genetic algorithm

In order to identify optimal solutions, genetic algorithms are inspired by the process of natural selection and mimic evolution. A genetic algorithm may be used to produce a collection of probable trajectories represented as individuals in a population in the context of spacecraft trajectory planning. Each individual is associated with a unique set of trajectory parameters, such as launch angle, velocity, and orbital parameters. The genetic algorithm then applies selection, crossover, and mutation operators to iteratively evolve the population. The fittest individuals, whose trajectories exhibit promising characteristics, are selected to form the next generation. This process continues for several iterations until convergence, with each generation improving the quality of the trajectories.

4.1.2 Hill Climbing algorithm

The gradient descent approach may be used with the genetic algorithm to further improve the route. Gradient descent is an optimization approach that seeks the minimum of a function by altering the parameters iteratively in the direction of the steepest descent.This algorithm starts from the ground surface and tries to reach titan by minimising the use of fuel from the function

$$-d^3\, log\,(\sqrt{fuel} + 10) \tag{24}$$

Where d is the distance to target (Titan). In order to do so, the algorithm starts with a randomly generated initial velocity. It then evaluates the quality of the solution using the cost function (24). The algorithm then makes small modifications to the current solution. If the modification improves the quality of the solution, i.e. reduces the cost function, it accepts the change and continues from the new solution. If the modification worsens the quality of the solution, it is discarded and another modification is tried.

4.1.2 Greedy Algorithm

The minimum distance between two points in space corresponds to the line connecting them. In our simulation of the universe, the two points have changing coordinates, so the line must be updated every time step. The greedy algorithm makes locally optimal decisions at each step until a globally optimal solution is reached. Our implementation of the algorithm is based on this principle, accelerating each time interval towards the Titan coordinates. As the spaceship approaches Titan the interval is reduced, thus improving its accuracy but increasing fuel consumption.

4.2 Landing Module

4.2.1 Wind

Simulating the wind conditions in titan is crucial for assuring that the landing simulation will be as accurate as possible, for this purpose the wind simulation was implemented with different areas of the atmosphere which is done by generating a vector that is completely parallel to titan's surface for each area of the atmosphere where the wind has a different magnitude and/or direction, in our simulation we only have 4 vectors, 3 of which are going from east to west and the remaining one is going in the opposite direction. The magnitudes of the vectors are taken from data from a NASA's article (Lorenz, R. D. (2022, March)) , at the beginning of the simulation they are set to be the magnitude that was captured from the data, as well as them being completely parallel, each vector that changes the magnitude and the angle in relation to the surface by a small random number that is chosen from a uniform distribution, these changes cannot exceed certain parameters chosen to make it so that the wind blows with a reasonable strength and angle, the maximum angle with which it can go either up or down is 20 degrees and the magnitude cannot exceed the one given taken from the data mentioned before.

The landing module is affected by the wind on each time step, and its velocity is just increased by the velocity of the wind multiplied by the time step of the simulation and divided by the mass of the landing module.

4.2.2 Controllers

Controllers are critical in ensuring the safe and effective landing of spacecraft. These controllers are in charge of monitoring the fall trajectory, maintaining the module's location, velocity, and orientation, and taking corrective steps to navigate through numerous obstacles such as air conditions, gravity forces, and external disturbances such as the wind, resulting in accurate and controlled landings. For this part of our spacecraft's journey, we've tried two types of controllers.

4.2.2.1 Feedback controller

This type of controller is more suitable with unpredictable situations, for example wind. The feedback controller functions by computing the error of the position of the landing module at each time

step and based on that error we correct the module's position by rotating and activating the engine. It has a great chance of success due to its adaptability to unforeseen events.

The module is initially positioned at coordinates X = 0 and Y = 300 kilometres, with Y = 0 representing Titan's surface. To aid in the landing procedure, the landing module includes a landing model and a gravity model. The controller constantly adjusts the module's position and velocity throughout the fall depending on the stated goal velocities at various altitudes. The targeted speeds are: 1 km/s at 200 km, 0.5 km/s at 100 km, 0.1 km/s at 20 km, 0.01 km/s at 10 km, 0.001 km/s at 5 km, 0.0001 km/s at 0.5 km, and eventually a full halt at the surface (0 km). A stochastic wind effect is explored to account for external influences, which might impact the module's trajectory during descent.

Furthermore, the controller takes corrective steps depending on the module's current altitude. If the module is more than 200 kilometres away, the controller corrects the X location and changes the Y velocity correspondingly. At elevations over 100 km, 20 km, and 10 km, the same corrective operations are carried out. Only the Y velocity is modified when the module reaches lower heights. The landing phase begins when the module reaches an altitude of less than 0.01 kilometres. Now, it guarantees that the module maintains a constant velocity of 0.00001 km/s in order to produce a smooth touchdown. The X position is also adjusted to match the desired trajectory. All of this is done while it keeps track of the progress and determines if the landing process is complete.

4.2.2.2 Open-Loop controller

This type of controller is easier to implement and use, because it is predefined and has only a certain amount of steps. Its advantages are that it uses less computation power, thus making the code more efficient. It functions by giving instructions to the landing module at certain time steps. For example, when the time hits 300 seconds after the start we accelerate upwards by 20% and so on. This method has great use in environments where changes are close to zero.

5. Experiments

5.1 Numerical Methods

We have compared the results of algorithms in order to test their accuracy and program execution time. For this purpose, we have run the different numerical methods during 30000 seconds of simulation, changing the step size, with a model test case of first degree equation, obtaining the absolute error with respect to the expected result. Furthermore, we have also measured the run time of the different numerical methods based on the time step used. We have used the following equation for our experiments:

$$y' = \frac{y}{t} - 2 \text{ ; solution } y(t) = t(3 - 2\,log(t)) \tag{25}$$

On the other hand, we have compared the results obtained after running our algorithms with the NASA Horizons database [2] to check the veracity of the simulation of the generated Solar System.

4.2 Landing Module

To test the landing module we have analysed the landing results obtained by modifying the wind. The landing is considered safe when the rocket reaches $y = 0$ with the variables satisfying

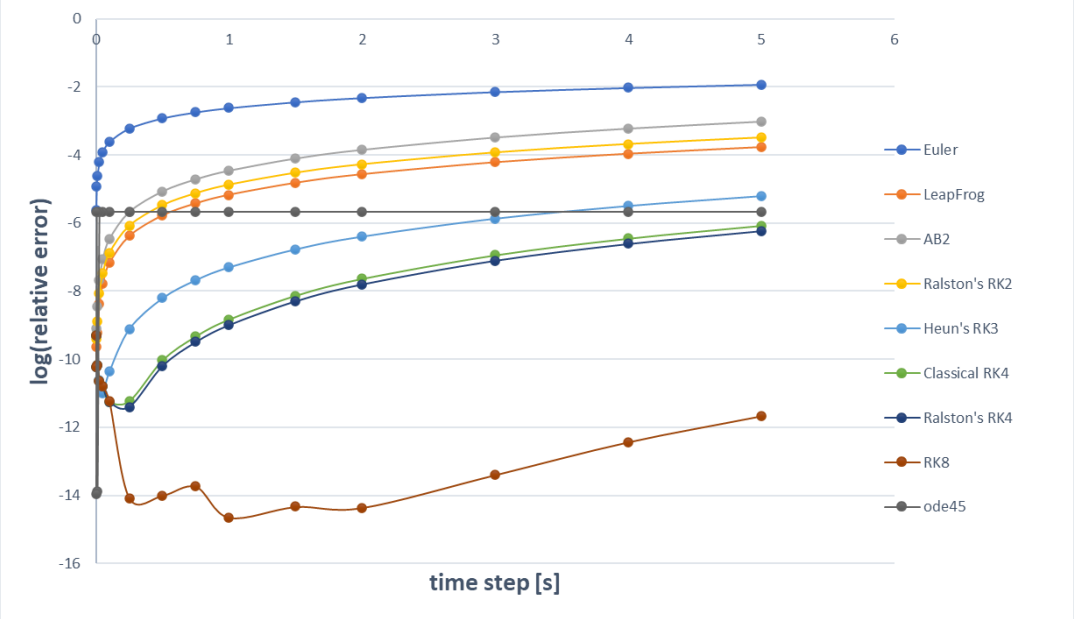
$$|x| \leq 10^{-4}\, km;\ |\theta \bmod 2\pi| \leq 0,02\, rad;\ |x'| \leq 10^{-4}\, km/s;\ |y'| \leq 10^{-4}\, km/s$$

When we activate the wind, we introduce a random variable so we have run the code 10 times to get the average data for the different wind scales.

6. Results

6.1 Numerical Methods

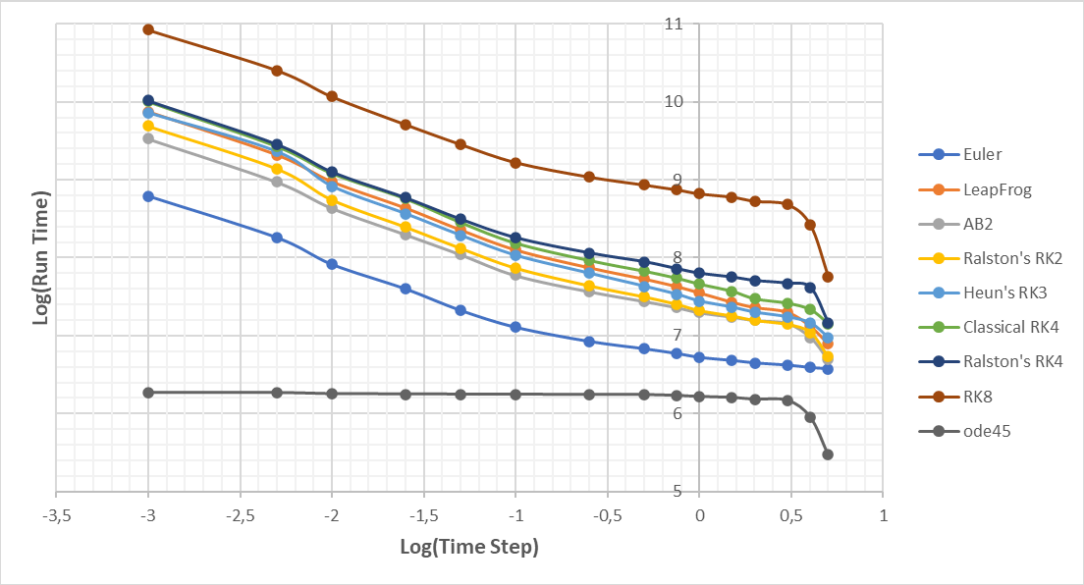
Graph 1 shows the results obtained after plotting the step size versus the logarithm of the relative error obtained with different numerical methods (they are specified in the legend) for equation 25 after 30000 s of simulation time



Graph 1: log-log plot relative Error

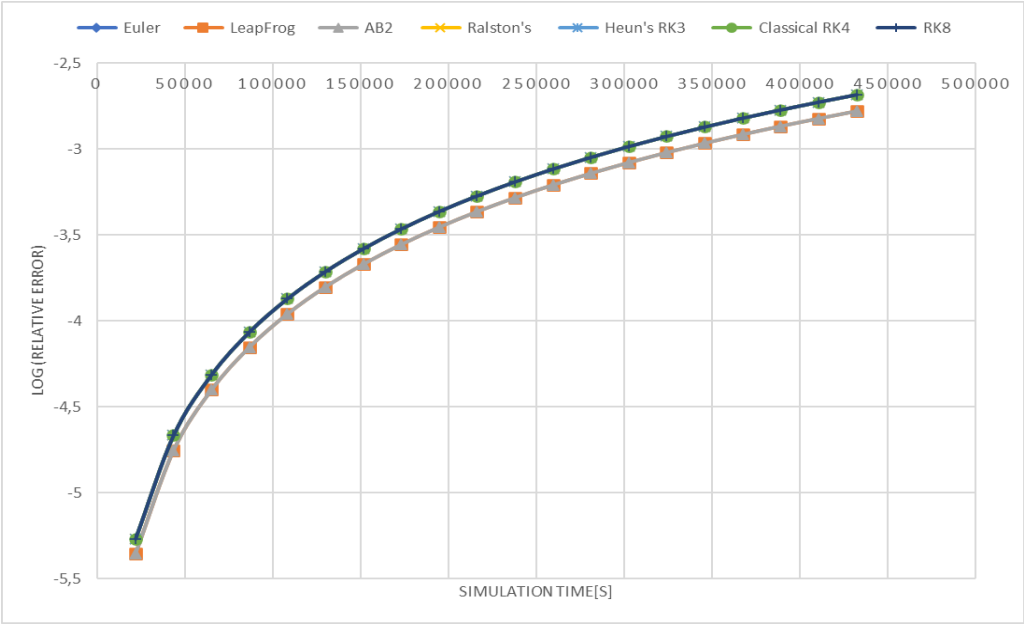
Graph 2 shows the results obtained after plotting the logarithm of the step size versus the run time of the different numerical methods (they are specified in the legend) for equation 25 until 30000 s of simulation.

[2] https://ssd.jpl.nasa.gov/horizons/app.html#/



Graph 2: Running Time

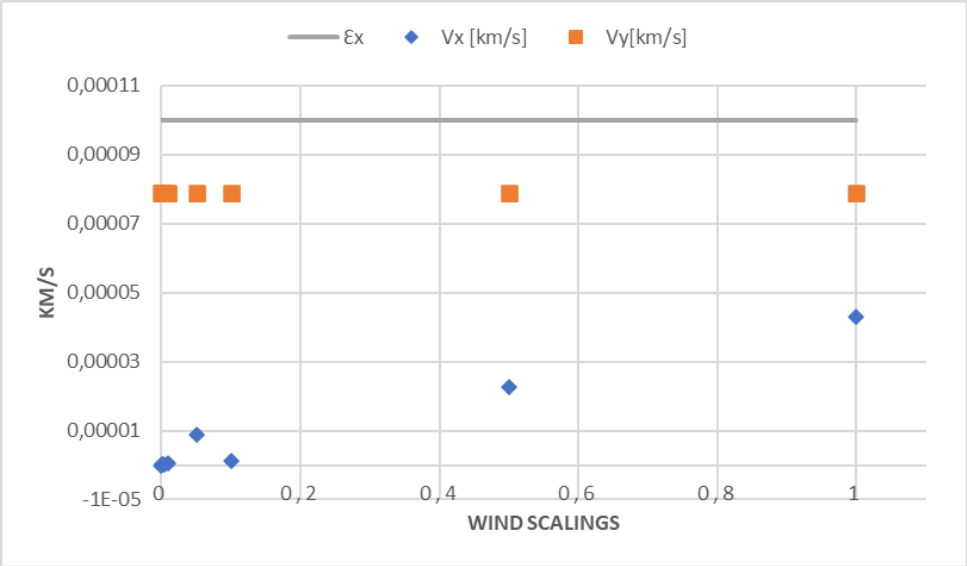
Lastly, graph 3 shows the results obtained after the comparison of the logarithm of the relative error in the Universe simulation data created versus the time elapsed in the simulation itself.



Graph 3: Relative Error compare with NASA data

6.2 Landing Module

Graph 4 shows the average value of the results obtained at the landing moment after running the landing module 10 times versus the wind scaling. In this graph the velocities at which the module impacts are analysed. On the one hand VX and VY represent the velocities of the spacecraft on the x and y axis respectively, while Ex is the maximum value that these values can have in order to ensure a safe landing.



Graph 4: Landing velocities

Finally, Table 1 shows the distance on the x-axis with respect to the expected landing point (0,0) applying the scaled-down wind.

Wind scaling	0	0.0001	0.0005	0.001	0.005	0.01	0.05	0.1	0.5	1
Distance [m]	0	0.160	0.085	0.310	3.100	6.100	4.600	2.700	3.000	10.000

Table 1 : Landing distance x-axis

7. Discussion

As shown in section 5 (graph 1), as the time step increases, the results obtained lose precision despite maintaining a similar distribution for all time steps. Furthermore, comparing the different numerical methods, Euler is the method with the lowest accuracy. AB2 is a linear multistep method, i.e unlike Euler, this method tries to gain efficiency by using the information obtained in previous steps and, therefore, the results obtained are more accurate than those obtained in Euler. On the other hand, the difference between Classical RK4 and Ralston's RK4 is barely visible, but both methods differ from RK3 and Ralston's RK2 by the values that the logarithm of the absolute error reaches, indicating that, as expected, as the Runge-Kutta order increases, more accurate values are obtained. In the case of RK8 we can see that when the error is always way lower than all the other solvers, as well as it hitting round off errors earlier than the other methods, This is due to its high precision which leads to these kinds of errors with small step sizes. We have also tested the results with an adaptive method ode45 with expected precision 1/100. The result obtained is constant since this method starts from a time step that increases or decreases looking for the maximum time step that remains within the established accuracy, so this algorithm is less accurate than the others when the initial time step is small (since the time step increases),

but it is considerably more accurate than the others when the initial time step is larger (the time step size decreases to match required precision).

Graph 2 shows the execution time of the different numerical methods. We can see that as the step size decreases the execution time increases exponentially. Furthermore, we can notice that the Runge-Kutta methods increase the run time as the order of the Runge-Kutta method increases. Based on this and the results of graphs 1, we observe that the run time differs considerably, so depending on the application that is going to be given, it would be necessary to weigh up to what extent it is profitable to use a higher-order method. In our case, the simulation time is 30,000 seconds, so the difference between one method and the other is hardly perceptible. However, for longer simulations the results may differ.

The errors obtained when we use the methods in our simulation of the universe are considerably larger than those obtained by applying the same methods to a first-degree differential equation. As shown in Graph 3, AB2 and LeapFrog remain slightly smaller than Euler and RK. This may be because Leapfrog and Adams-Bashforth methods are both examples of symplectic integration methods so they have certain mathematical properties related to conservation of energy and momentum (Press, Teukolsky, Vetterling & Flannery (2007)). On the other hand, looking at the trend line we can see how both errors grow linearly. A considerable error when the simulation covers long periods of time.This may be due to the comparison of our data with those of NASA Horizons, since we have created a simplified version of the Universe where only celestial bodies with large masses are represented, ignoring others such as the moons of Saturn, and relativistic issues that may vary the orbits of the bodies are ignored, while NASA takes into account these aspects, among others, to develop its database.

On the other hand, analysing the results obtained in the landing module experiments (graph 4 and table 1), we deduce that the landing is safe with respect to the velocity regardless of the wind, so that the rocket would not hit the surface. However, the results obtained for the landing position show that, by incorporating the random component of the wind, the landing position varies, so that the landing does not meet the established criteria. Despite the gradual increase in wind scaling, no stable pattern of behaviour is observed, although the furthest value is obtained when the wind acts in its entirety.

8. Conclusion

This project was devoted to organising and carrying out a human trip to Titan, one of Saturn's moons. The plans called for the launch of a spacecraft, orbital insertion near Titan, a safe surface landing, and crew return to Earth. In order to do this, a mathematical model of spaceship motion and the orbits of the sun, planets, and the moon was created. The trajectories of heavenly objects were computed using an explicit Runge-Kutta 4 solver in a physics engine. The numerical solver's various step sizes were tested to assure accuracy and prevent implausible results, among other things. Log-log graphs were used for analysis and presentation of the data. To help with mission planning and comprehension, the routes taken

by the spacecraft and astronomical objects were depicted. By employing the same coordinate system as NASA Horizons and taking quantities like the mass, distances, and velocities into consideration, the project's needs were satisfied. In seconds after launch, the launch position and velocity were stated in relation to the position and velocity of Earth. All aspects including the spacecraft's separation from Earth, probe's closest approach to Titan and the process of landing a module were documented and displayed.

Implementing several differential equation solvers, including the fourth-order Runge-Kutta solver and the Euler solver, and evaluating their accuracy for various step sizes were additional jobs. Using a hill climb or gradient descent method, the fuel consumption of the ideal mission—which included takeoff and landing on Earth—was estimated and reduced to the absolute minimum. To assure code functioning and get 100% code coverage statistics, straightforward JUnit test cases were built.

Therefore, after successfully planning and executing all tasks including an exploration trip to Titan, a safe landing process inside Titans orbit and an elaborate return mission to Earth. The research displayed a thorough grasp of celestial mechanics, numerical solvers, optimization methods, and graphical visualisation. By completing the objectives and achieving the project criteria, the crew has created a functional software capable of facilitating and executing a space mission.

9. References

Banu, M. S., Raju, I., & Mondal, S. (2021). A comparative study on classical fourth order and butcher sixth order Runge-Kutta methods with initial and boundary value problems. *International Journal of Material and Mathematical Sciences*, *3*(1), 8-21. doi: 10.34104/ijmms.021.08021

Boone, S. & McMahon, J. (2022, June). Semi-analytic spacecraft manoeuvre design with stochastic constraints. In *2022 American Control Conference (ACC)* (pp. 1608-1613). IEEE.doi: 10.23919/ACC53348.2022.9867632

Butcher, J.C. (1964). On Runge-Kutta processes of high order. Journal of the Australian Mathematical Society, 4(2), pp.179-194. doi: 10.1017/S1446788700023387

Carter, J. (2022, December). In Photos: Webb Telescope's First Look At Titan, Saturn's Giant Moon That May Once Have Hosted Life.Forbes. https://www.forbes.com/sites/jamiecartereurope/2022/12/02/in-photos-webb-telescopes-first-look-at-titan-saturns-giant-moon-that-may-once-have-have-hosted-life/

Chalup, S., & Maire, F. (1999). A study on hill climbing algorithms for neural network training. In Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Vol. 3, pp. 2014-2021). IEEE. doi: 10.1109/CEC.1999.785522

Ganse, B., Ganse, U., Ganse, B., & Ganse, U. (2020). How to Fly a Spacecraft. The Spacefarer's Handbook: Science and Life Beyond Earth, 87-128. doi: 10.1007/978-3-662-61702-1_3

Gianluca, F., Meoni, G. & Moloney, D. (2020) Towards the Use of Artificial Intelligence on the Edge in Space Systems: Challenges and Opportunities. IEEE Aerospace and Electronic Systems Magazine, 53. doi: 10.1109/MAES.2020.3008468

Hall, H. (2022). Global optimisation of interplanetary trajectories (Doctoral dissertation, University of Southampton). http://eprints.soton.ac.uk/id/eprint/471276

Hernandez, D. & Bertschinger, E. (2018). *Time-symmetric integration in astrophysics*. doi: 10.1093/MNRAS/STY184

Higham, Desmond. (2001). An Algorithmic Introduction to Numerical Simulation of Stochastic Differential Equations. SIAM Review. 43. 525-546. doi: 10.1137/S0036144500378302.

Jo, B., & Ho, K. (2023). Simultaneous Sizing of a Rocket Family with Embedded Trajectory Optimization. to find a related paper Jamilnia, R., & Naghash, A. (2012). Simultaneous optimization of staging and trajectory of launch vehicles using two different approaches. Aerospace Science and Technology, 23, 85-92. doi: 10.48550/arXiv.2302.12900

Johansson, R., & Johansson, R. (2019). Plotting and visualisation. Numerical Python: Scientific Computing and Data Science Applications with Numpy, SciPy and Matplotlib, 295 - 333. https://link.springer.com/book/10.1007/978-1-4842-4246-9

Lorenz, R. D. (2022, March). Decoding the Descent Dynamics of the Huygens Probe. In *2022 IEEE Aerospace Conference (AERO)* (pp. 1-14). IEEE. doi: 10.1109/AERO53065.2022.9843232

Lu, L., Meng, X., Mao, Z., & Karniadakis, G. E. (2021). DeepXDE: A deep learning library for solving differential equations. SIAM review, 63(1), 208-228. doi: 10.1137/19M1274067

Mahdi, F. M., Salih, A. R. H., & Jarad, M. M. (2020). Determination and evaluation of the orbital transition methods between two elliptical earth orbits. Iraqi Journal of Science, 224-234. doi: 10.24996/ijs.2020.61.1.25

Newton, I. (1966). Philosophiae naturalis principia mathematica, London 1687. English translation: Sir Isaac Newton's Mathematical Principles, Univ. of Calif. Press, Berkeley.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). Numerical Recipes: The Art of Scientific Computing (3rd ed.). Cambridge University Press.

Prügel-Bennett, A. (2004). When a genetic algorithm outperforms hill-climbing. Theoretical Computer Science, 320(1), 135-153. doi: 10.1016/j.tcs.2004.03.038

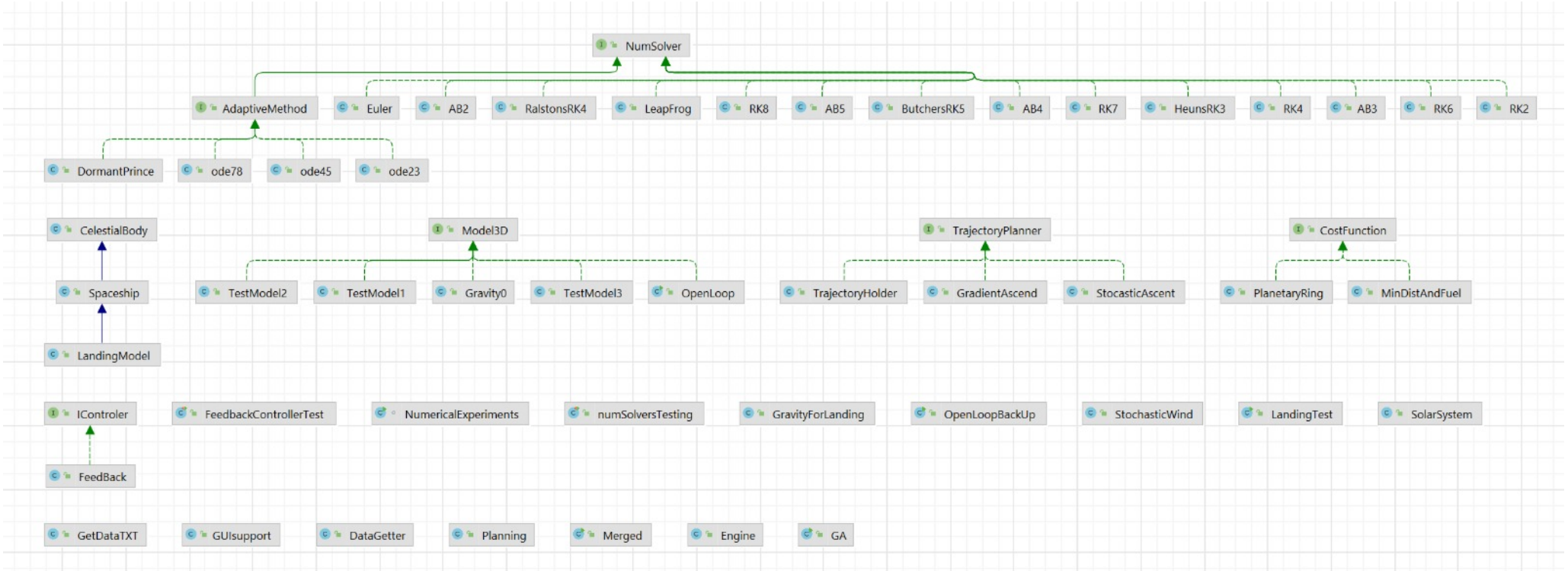
Sanz-Serna, J. M., Verwer, J. G., & Hundsdorfer, W. H. (1986). Convergence and order reduction of Runge-Kutta schemes applied to evolutionary problems in partial differential equations. *Numerische Mathematik*, *50*, 405-418.

Steigerwald, B. (2023). NASA Instrument Bound for Titan Could Reveal Chemistry Leading to Life. *NASA*. https://www.nasa.gov/feature/goddard/2023/dragonfly-drams-instrument

Toniato, J. D., Rodrigues, D. C., & Wojnar, A. (2020). Palatini f (R) gravity in the solar system: Post-Newtonian equations of motion and complete PPN parameters. Physical Review D, 101(6), 064050. doi: 10.1103/PhysRevD.101.064050

Zhou, X., Gao, F., Fang, X., & Lan, Z. (2021). Improved bat algorithm for UAV path planning in three-dimensional space. IEEE Access, 9, 20100-20116. doi: 10.1109/ACCESS.2021.3054179

10. Appendix


NumSolver
AdaptiveMethod
Euler
AB2
RalstonsRK4
LeapFrog
RK8
AB5
ButchersRK5
AB4
RK7
HeunsRK3
RK4
AB3
RK6
RK2
DormantPrince
ode78
ode45
ode23
CelestialBody
Model3D
TrajectoryPlanner
CostFunction
Spaceship
TestModel2
TestModel1
Gravity0
TestModel3
OpenLoop
TrajectoryHolder
GradientAscend
StocasticAscent
PlanetaryRing
MinDistAndFuel
LandingModel
IControler
FeedbackControllerTest
NumericalExperiments
numSolversTesting
GravityForLanding
OpenLoopBackUp
StochasticWind
LandingTest
SolarSystem
FeedBack
GetDataTXT
GUIsupport
DataGetter
Planning
Merged
Engine
GA