A Titanic Space Odyssey!

Group 18:
Karl Bradley, Grzegorz Stark, Alisa Todorova, Yutong Zhou

Maastricht University
Department of Data Science and Knowledge Engineering
Project 1.2

Abstract

Titan is the largest moon of the planet Saturn, which in many features resembles a very early planet Earth. Its atmosphere is four times denser than Earth's. Further, Titan is the only known body other than Earth to have stable surface liquids, namely lakes and seas, on its surface. Thus, scientists believe that exploring Titan could give new insights of how life may have formed on Earth.

The goal of our project "A Titanic Space Odyssey!" is to simulate a manned mission to Titan and back. The aim of this report is to answer the following research questions: How does the accuracy of the results improve as the step-size is decreased? Is there a minimum step-size needed for reasonable results?

Keywords: Euler solver, Verlet solver, Runge-Kutta solver, Differential equations, Solver, Higher-order differential equation solvers, Titan, Space mission, Step-sizes

Table of Contents

1. Introduction

Titan is the largest moon of the planet Saturn. It was telescopically discovered by the Dutch astronomer Christiaan Huygens in 1655 and named after the Titans from the Greek mythology (Hubbard & Buratti, 2020). In many features Titan resembles a very early planet Earth. Its atmosphere is four times denser than Earth's. Further, Titan is the only known body other than Earth to have stable surface liquids, namely lakes and seas, on its surface. Thus, scientists believe that exploring Titan could give new insights of how life may have formed on Earth (Northon, 2019).

The goal of our project "A Titanic Space Odyssey!" is to simulate a manned mission to Titan and back. First, we built a model of the Solar system and implemented it into a Graphical User Interface (GUI), using the Java library libGDX. We created and launched an exploratory probe, which we shot with a ballistic missile from Earth, to land somewhere on Titan. During this exploratory mission the probe had no engines, and therefore, only gravitational forces were acting on the probe. We implemented a physics engine for computing the paths of the celestial objects, as well as an Euler solver, a Verlet solver, and a Runge-Kutta solver. For our second mission we created a rocket, which was launched from Earth, landed on Titan, and then was brought back to Earth. This time there was a rocket engine, with which we could perform corrections to the rocket's trajectory. Our project and this report aim to answer the following research questions: How does the accuracy of the results improve as the step-size is decreased? Is there a minimum step-size needed for reasonable results?

This paper is divided into seven main sections in order to examine and discuss all aspects of this project. First is this introduction of the project and paper, which introduces the project, the problem statement of the project and thus, the focus of this paper. Then, in Methods is given the notation used throughout this paper. Furthermore, this section describes in detail the implemented solvers developed for this project. In Implementation, the basic structure of the implementation is presented through UML diagrams, as well as the GUI. In Experiments, the conducted experiments are discussed. In Results all results from the experiments are presented, which are then interpreted and explained in the Discussion section. In Conclusion a summary of this project and paper is provided.

2. Assumptions

Simulating every physical detail would be difficult. Therefore, we have made multiple simplifications and assumptions regarding the project and our approaches.

1. Number of celestial bodies in the solar system that affect each other is limited to: Sun, Mercury, Venus, Mars, Earth, Moon, Neptune, Jupiter, Uranus, Saturn and Titan.
2. Rotation of the planets is not taken into account.
3. Effects of the atmospheres on Earth and Titan are not taken into account.
4. Trajectories between Earth and Titan are calculated in 3 dimensions, regardless of the fact that visuals are rendered on the screen only in 2 dimensions.
5. In order to reduce complexity, landing on Titan is simulated in 2 dimensions but the world is visualised and modeled in 3 dimensions for a more pleasant visual experience.
6. Titan lander is assumed to be a cone with 15 meters of base diameter and 10 meters of height.

3. Methods

3.1 Differential equations

Differential equations can be used to directly observe and measure how systems change in time. A first-order differential equation for a quantity y is a formula that gives the time-derivative of y in terms of y itself:

$\dot{y} = f(t, y)$, where $\dot{y}$ denotes the Newton's notation for $\frac{dy}{dt}$ i.e. first derivative.

Second-order differential equations, are written in Newton's notation as $\ddot{y} = \frac{d^2y}{dt^2}$, and so on.

If y is finite-dimensional, then the formula is an ordinary differential equation (ODE). Thus, a differential equation is any equation that contains derivatives, either partial derivatives or ordinary derivatives (Burden & Faires, 2012).

3.2 Notation

In this paper, and specifically this section, SI standard symbols for physical units and mathematical notation are used. Standard SI units are also used for kilograms, meters,

seconds, etc. Further, f denotes a function, and t denotes time. y is a dependent variable, such as $y = f(x)$ is an unknown function of the independent variable x. G is the gravitational constant and $G = 6.67408e - 11$.

3.3 Euler's method

The Euler's method is a 1st-order numerical method, named after the Swiss mathematician Leonhard Euler in the 1800s. 1st-order means that the local truncation error, i.e. the error caused by one iteration, is on the order of $O(h^2)$. It's used to solve ordinary differential equations, such as $\dot{y} = f(t, y)$, with a given initial value $y(t_0) = y_0$. It's also the simplest Runge-Kutta method. Its definition is the following equation:

$$y_{n+1} = y_n + hf(t_n, y_n), \text{ for } n = 0, 1, 2, 3,....,$$

where $t_n = t_0 + nh$; h is the time step in the interval (t_n, y_n); and $t_{n+1} = t_n + h$.
Errors from the Euler's method depend on the value of h. Therefore, if h is small, then the errors are also small (Palasarn & Toutip, 2011).

3.4 Fourth order Runge-Kutta solver

The 4th-order Runge-Kutta method was developed in the 1900 by the German mathematicians Carl Runge and Wilhelm Kutta. 4th-order means that the local truncation error, i.e. the error caused by one iteration, is on the order of $O(h^5)$, while the total accumulated error, which is caused by many iterations, is on the order of $O(h^4)$. It's used to solve the numerical solution of the ordinary differential equation, $\dot{y} = f(t, y)$. It's initial condition is $y(t_0) = y_0$ (Tan & Chen, 2012). The definition of the 4th-order Runge-Kutta method for this initial value problem is the following equation:

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \text{ , for } n = 0, 1, 2, 3,....,$$

where h is the time step, and k_1, k_2, k_3 and k_4 are coefficients, defined as follows:

$$k_1 = f(t_n, y_n)$$

$$k_2 = f(t_n + \frac{h}{2}, y_n + k_1\frac{h}{2})$$

$$k_3 = f(t_n + \frac{h}{2}, y_n + k_2\frac{h}{2})$$

$$k_4 = f(t_n + \frac{h}{2}, y_n + k_3 h)$$

These coefficients indicate the function's slope at three points in the time interval: the beginning, the midpoint and the end. k_1 is the slope at the beginning of the interval, using y. This is also known as the Euler's method (see section 2.2.1 for more information). k_2 is the slope at the midpoint of the interval and uses y and k_1. k_3 is also the slope at the midpoint, but uses y and k_2. k_4 is the slope at the end of the interval, using y and k_3 (Voesenek, 2008).

3.5 Verlet solver

The Verlet algorithm is a classical integration method in Newtonian mechanics. It is a numerical integration method applied to Newton's second law and widely used in mechanical calculations, especially in planetary motions. Compared with Euler's method, it offers greater stability, timer-reversibility and symplecticity (Huebl, 2010).

Given the position r and the velocity v of the particle at time t, we can get the position r and the velocity $v(t + h)$ at time $t + h$. Only by knowing the position and acceleration of the planet at the current moment, the position at the previous moment can be obtained at the next moment. The position expression can be obtained after adding the Taylor expansion of $x(t + h)$ and $x(t - k)$:

$$x(t + h) = x(t) + v(t)h + \frac{a(t)h^2}{2} + \frac{b(t)h^3}{6}$$

$$x(t - h) = x(t) - v(t)h + \frac{a(t)h^2}{2} - \frac{b(t)h^3}{6}$$

$$x(t + h) = 2x(t) - x(t - h) + a(t)h^2$$

Similarly, the current velocity can also be calculated by obtaining the position at the next time, so the velocity expression can be obtained by subtracting and dividing by $2h$:

$$v(t) = \frac{x(t+h) - x(t-h)}{2h}$$

4. Implementation

In this section, the implementation of the separate classes is briefly explained and supported by UML diagrams.

4.1 Differential equation solvers

4.1.1 Euler's method

The Euler's method is calculated in the method step.

EulerSolver
+ G: double
+ step(spaceObjects: List<SpaceObject>, step: double): void

Diagram 1: A UML diagram for our implementation of the Euler's method.

4.1.2 Fourth order Runge-Kutta solver

The 4th-order Runge-Kutta method is calculated in the method step.

The method F returns the value of the numerical solution of the ordinary differential equation, $\dot{y} = f(t, y)$.

RungeKuttaSolver
+ G: double
+ step(spaceObjects: List<SpaceObject>, step: double): void - F(t: double, y: State, obj1P: Vector3d, i: int): Vector3d

Diagram 2: A UML diagram for our implementation of the 4th-order Runge-Kutta method.

4.1.3 Verlet solver

The Verlet's method is calculated in the method step.

Verlet
+ G: double
+ step(spaceObjects: List<SpaceObject>, step: double): void

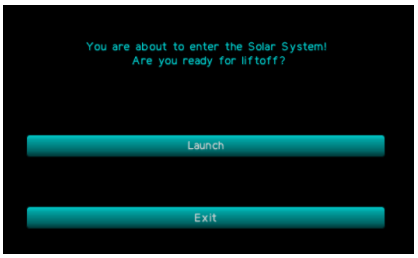
4.2 Graphical User Interface (GUI)

For this project we built a model of the Solar system and implemented it into a Graphical User Interface (GUI), using the Java library libGDX. This library was chosen

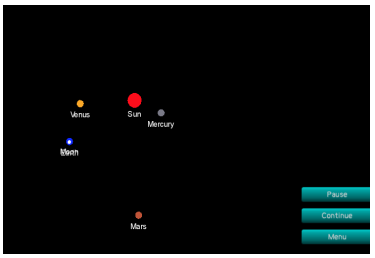
because it provides a unified API, which works across all supported platforms. As per the project's requirements, our Solar system consists of the Sun, the 8 planets (Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune), the Earth's moon Luna, and Titan.

When running our program from the file DesktopLauncher.java, the first scene that appears is the main menu. To create the illusion that the user is inside a rocket and there's a control panel in front of the user asking for take off initiation, we used neon colors and the buttons "Launch" and "Exit". (See picture 1 for reference). When the user clicks on "Launch", a new scene is loaded. It has our 2D model of the Solar system, as well as the buttons "Pause" and "Continue", with which the user can pause and then continue the simulation, i.e. the rotation of the planets, and the button "Menu", which leads to the first scene, i.e. the main menu. (See picture 2 for reference).

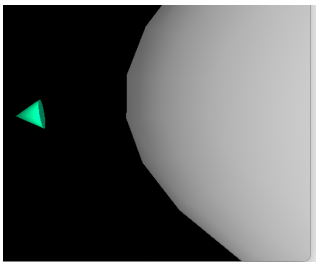
For faster testing during phase 3 of our project, we decided to remove the main menu, and instead turn our 2D model of the Solar System into a 3D model. (See picture 3 for reference). Both the 2D and the 3D models of our Solar System were created with the help of libGDX's 2D and 3D graphics and vectors packages. The Camera class was also used, so the user can move around the Solar system, as well as zoom in and out on a planet.



Picture 1: The main menu of our program.



Picture 2: 2D Solar system scene.



Picture 3: Final version of 3D Solar system scene.

4.3 Trajectory

Based on Newton's gravitational theorem, the launch of a probe at a specific location is affected by the tug of the other planets, and we predetermined the spacecraft's initial velocity toward Titan's atmosphere, and then manually selected its trajectory according to the situation.

4.4 Landing on Titan

Landing module and its physics are simulated differently than our previous phases. The differential equations describing the motion are:

$$\ddot{x} = u\,sin(\theta)$$
$$\ddot{y} = u\,cos(\theta) - g$$
$$\ddot{\theta} = v$$

Where x is the horizontal position, y is the vertical position, θ is the angle of rotation, u is the acceleration provided by the main thruster, v the torque provided by the side thrusters, and $g \approx 1.352\ m/s$ the acceleration due to gravity on Titan. We also calculate torque using the following, simplified formula:

$$v = F,$$

where F is the thrust produced by side thrusters. The lander can be then dropped from a chosen altitude.

4.4.1 Stochastic wind model

Titan was the first moon in the solar system to be found to have a thick atmosphere , mostly filled with nitrogen, methane and hydrogen. Because of different composition, an opaque layer of haze blocks much of the visible light from the sun and other sources. Plus, Titan has lower gravity than Earth. Under this situation, its atmosphere would be more expansive and complicated than Earth's (Greeley & Iversen, 1987). To land safely on Earth, four tolerances are listed under the premises which will be applied in the future successfully:

1. the tolerance for velocity should less than 0.1 meters per second
2. the tolerance for angular velocity should less than 0.01
3. the tolerance for the angle between the vertical axis of the ground should be less than 0.02
4. the distance to the ground should less than 0.1 meter

Thus, the stochastic wind model is proposed. It is aimed to provide controllers with precise velocity to land steadily on the surface of Titan when facing with varying distance from its surface and wind speeds at specific height. The reason Titan has stochastic wind blowing is the unpredictable angle of the wind and the force of it. Wind is especially vital when the thrusters are trying to land on the surface of Titan. Relatively reliable data for wind in different stages can be found in many materials.

If the attitude from ground to the probe is larger than 500 kilometers (because the atmosphere of Titan is up to 600 kilometer, we can assume this is where the atmosphere extends to), the wind speed will be 0. However, when the lander is 120 kilometers above the surface, it can measure a wind velocity of 120 meter per second. After the lander hit the

ground(0 kilometer above the surface), the wind speed dropped significantly to 0.3 meter per second. In order to improve the accuracy of the data and confirm the actual situation, the variation of wind speed can be expanded by 30 percent, which means, the random wind speed can reach up to 156 meter per second at high altitude. Under the results of this study, a basic calculation of wind speed can be provided in a simple linear function based on real-life Titan observation:

$$f(h) = (0.0009975 \times h + 0.3) \times a$$

where a is a random factor and can be any number between 0.7 and 1.3.

4.4.2 Closed-loop controllers

Closed-loop controllers are devices or systems that automatically regulate the lander's thrust in real time, without any human interactions. In order to land on the surface of Titan safely, we have implemented 2 different controllers to manage a single main thruster and two small thrusters on the sides.

4.4.2.1 On-Off Controller

The first and the most basic controller is the On-Off Controller. It has been initially created to handle rotational control and its main advantage is the straightforward and intuitive implementation.

```
algorithm on-off-control is
          input: Current rotation of the lander in degrees theta,
                 current horizontal position pos,
          output: Side thrust value

          error ← TARGET_X - pos

          if theta >= 45
              return -MAX_THRUST
          else if t <= -45
              return MAX_THRUST

          if error < 0
              return -MAX_THRUST
          else
              return MAX_THRUST
```

Where *TARGET_X*, *MAX_THRUST* are predefined variables and constants of a class *OnOff*.

4.4.2.2 PID Controller

The more sophisticated controller is the PID Controller, which in our case, is responsible for controlling the main thruster. It tries to solve the inertia problem discussed before by observing the state over time and applying proportional, integral, and derivative terms as corrections to the generated output.

1. Proportional term describes the scale of the output with respect to the error term. Intuitively, the larger the error, the larger thrust we want to generate to minimize it.
2. Integral term adjusts the output based on accumulated error over time. It results in diminishing the effect of the proportional term as the error decreases.
3. Derivative term takes into account the current rate of change of the error. The more rapid the change, the more rapid the change of thrust should be.

Those terms are predefined parameters that need to be first tuned, in order to land safely.

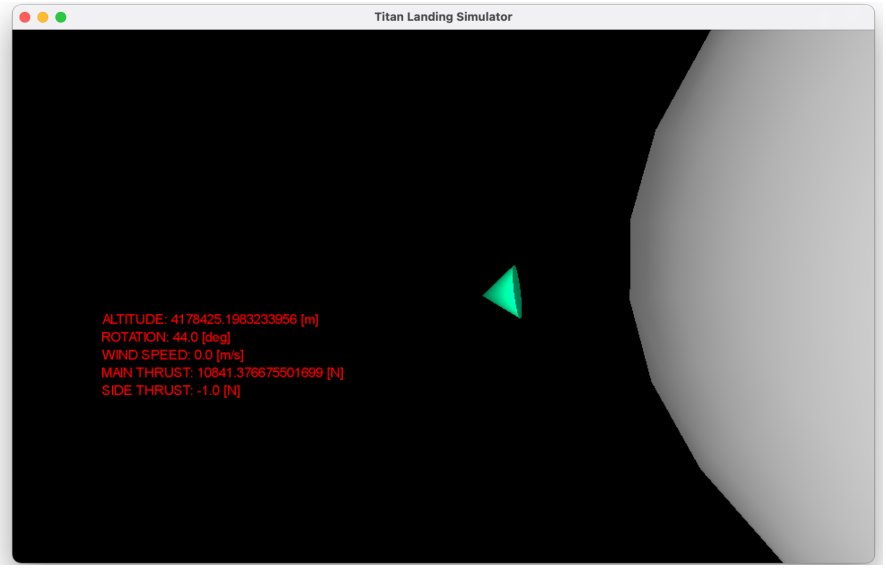
```
algorithm pid-control is
          input: Current vertical position pos
          output: Main thrust value

          error ← TARGET_Y - pos
          integralError ← integralError + error * TIME_STEP
          derivativeError ← (error - errorLast) / TIME_STEP
          errorLast ← error
          output ← kp*error + ki*integralError + kd*derivativeError

          if output >= MAX_THRUST
              output ← MAX_THRUST
          if output <= 0
              output ← 0

          return output
```

Where *TARGET_Y*, *kp*, *ki*, *kd*, *error*, *integralError*, *errorLast*, *derivativeError*, *TIME_STEP*, *MAX_THRUST* are predefined variables and constants of a class *PID*.



Picture 4: 3D visualization of our landing simulation

5. Experiments

5.1 Pendulum motion

In order to find check the validity of our 3 solvers, as well as see the effect of different step sizes, we conducted the following experiment on our solvers:

We simulate the motion of a pendulum on earth, implementing the same solver as we use to simulate the motion of the planets.

The acceleration of the pendulum can be calculate with the following formula:

$$a = \ -\mu * v - \frac{G}{L} * sin(\theta),$$

where a is acceleration, μ is air resistance, v is velocity, L is the length of pendulum, G is the gravitational constant, and θ is theta - the angle of the pendulum in radians.

$-\mu * v$ represents the air resistance relative to velocity.

We let the air resistance $\mu = 0.1$, we let the gravitational constant $G = 9.8$, and we let the length $L = 2$.

For all the experiments, we have initial θ = π/3 (60°) and initial velocity $v = 0$.

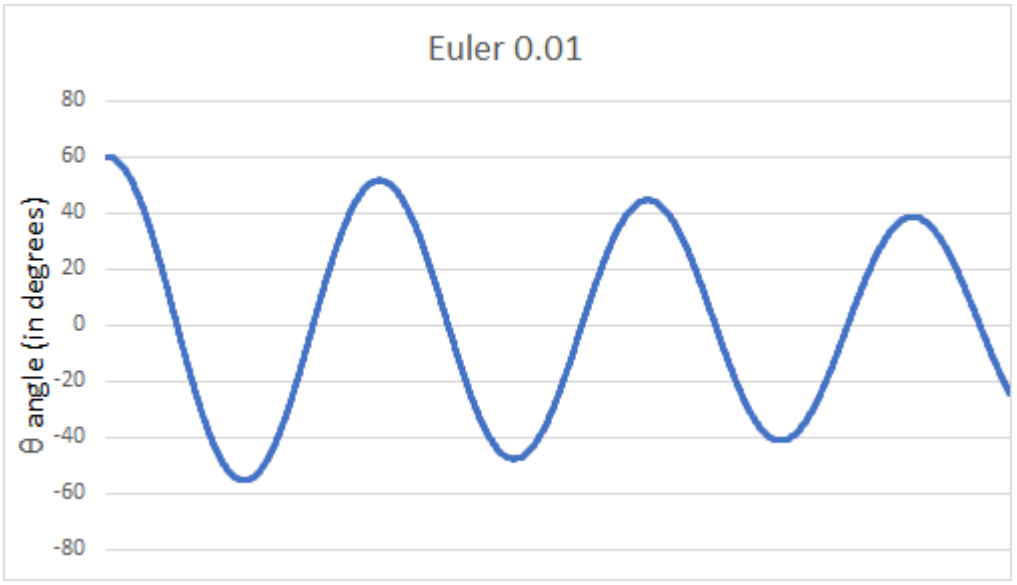
5.2 Landing controller

In order to avoid crashing the lander, we have experimented with proportional, integral and derivative terms of PID Controller such that we can touch the surface of Titan as

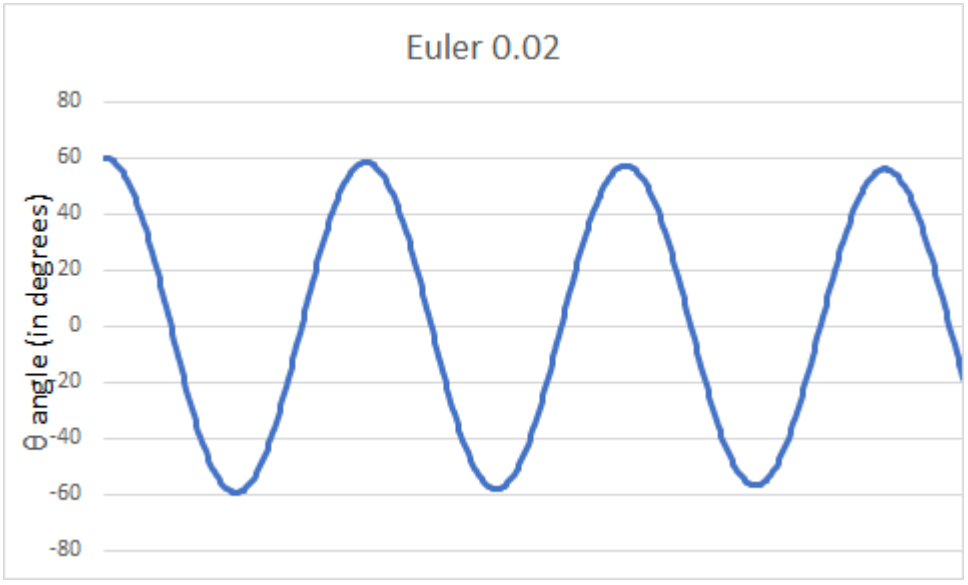
gently as possible by automatically controlling the main thruster. The best values we could find are *kp = 0.1, ki = 0, kd = 500.*

6. Results

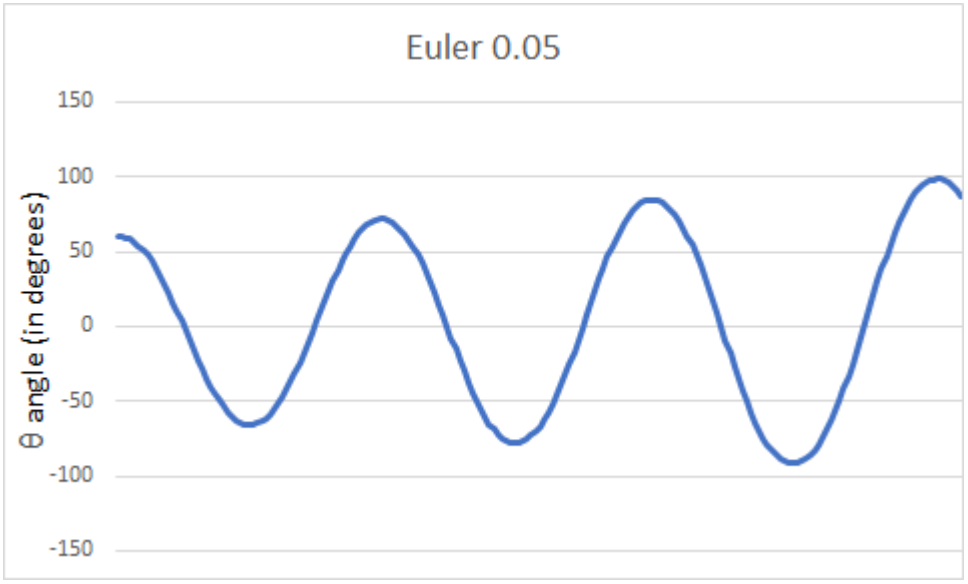
6.1 Euler's method



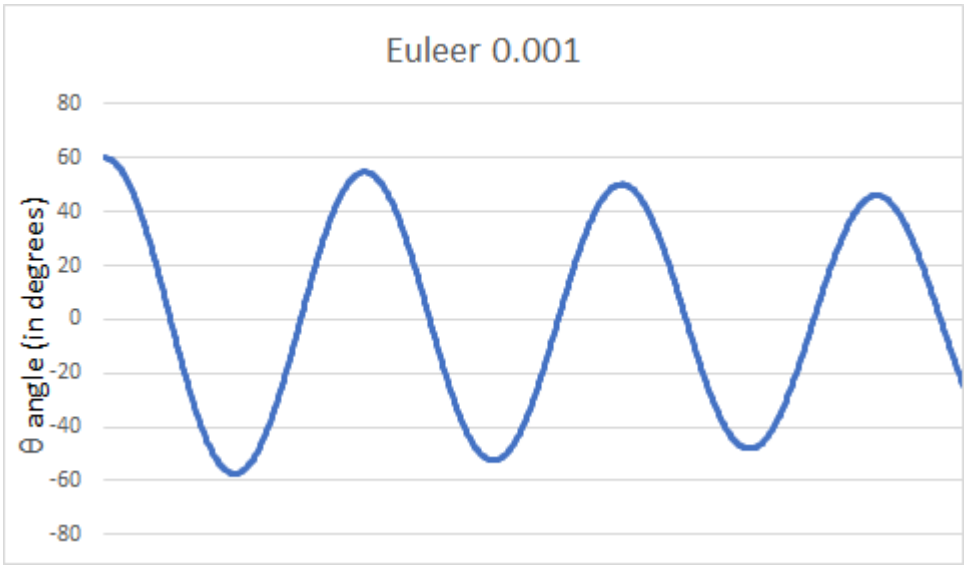
Graph 1: Euler method with step-size 0.01 span, time t = 10



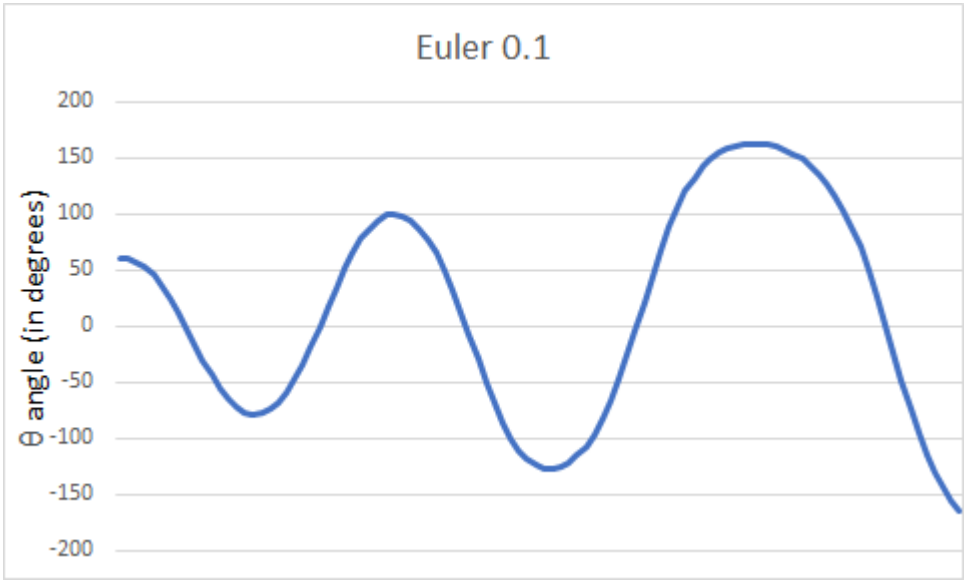
Graph 2: Euler method with step-size 0.02 span, time t = 10



Graph 3: Euler method with step-size 0.05 span, time t = 10

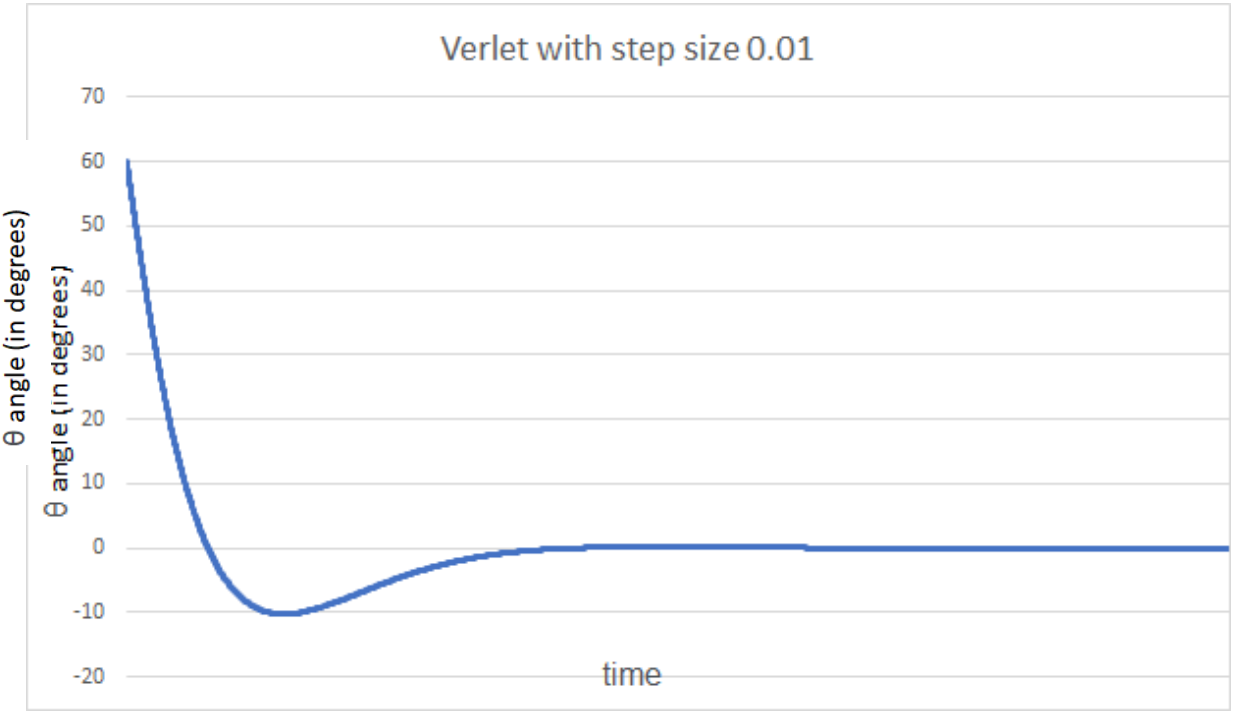


Graph 4: Euler method with step-size 0.001 span, time t = 1



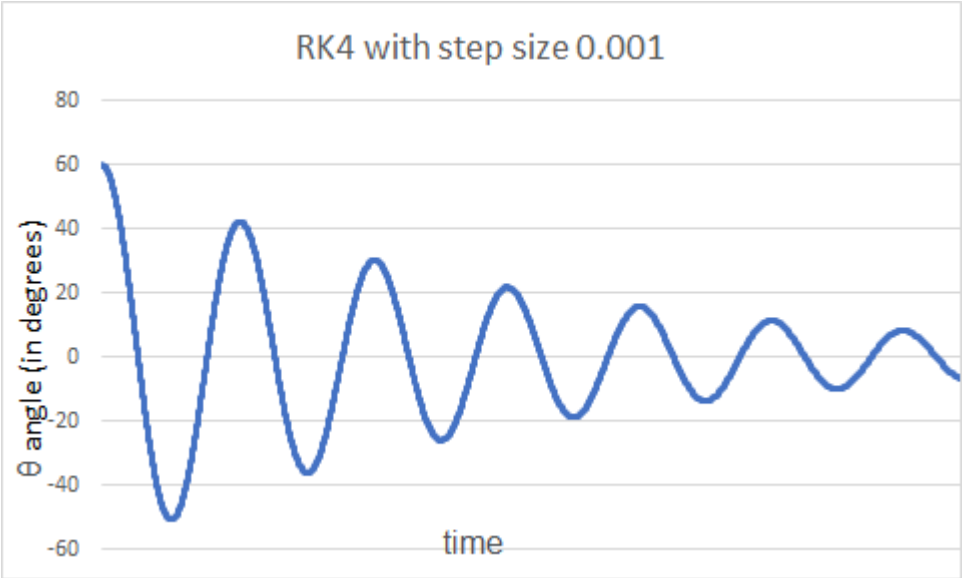
Graph 5: Euler method with step-size 0.1 span, time t = 10

6.2 Verlet method

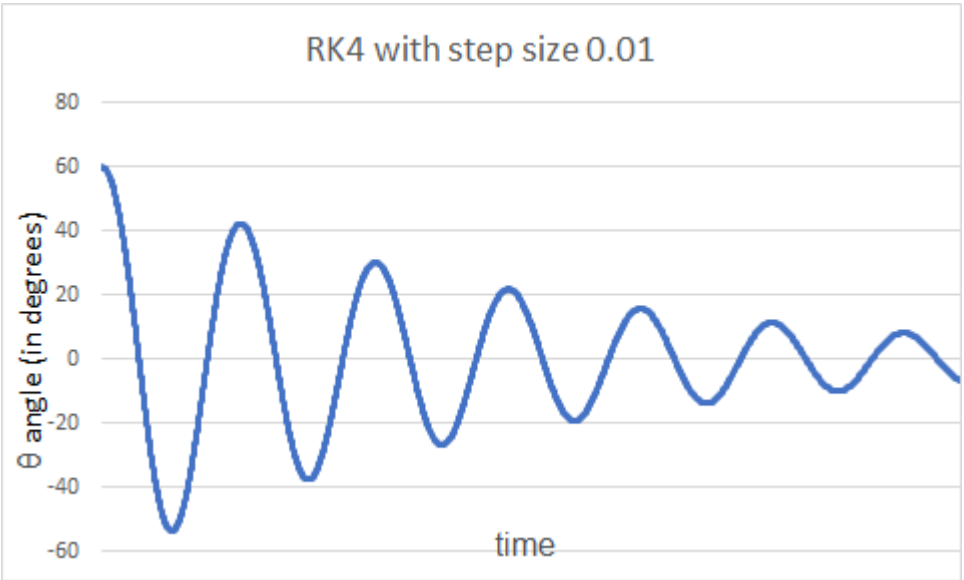


Graph 6: Verlet method with step-size 0.01 span, time t = 10

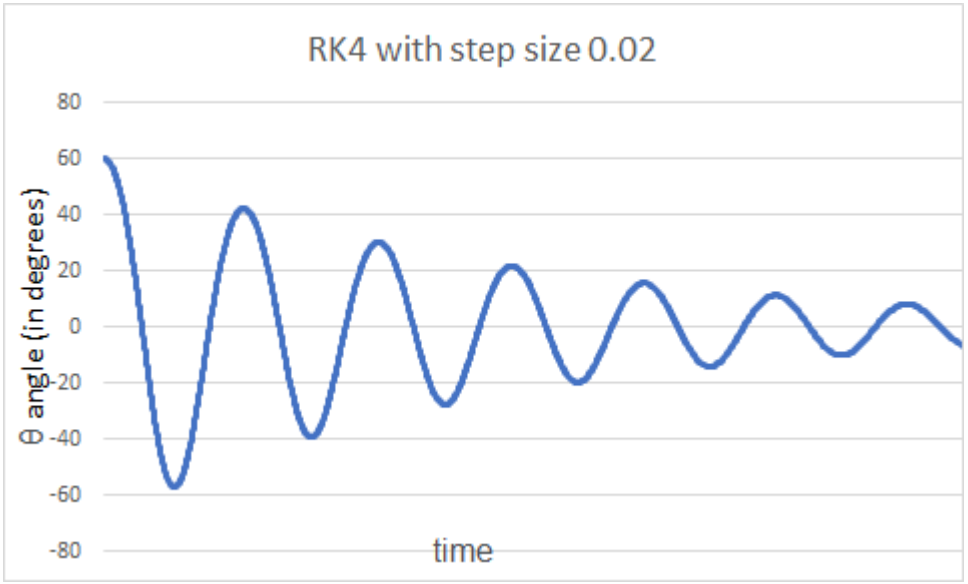
6.3 4th-order Runge-Kutta method



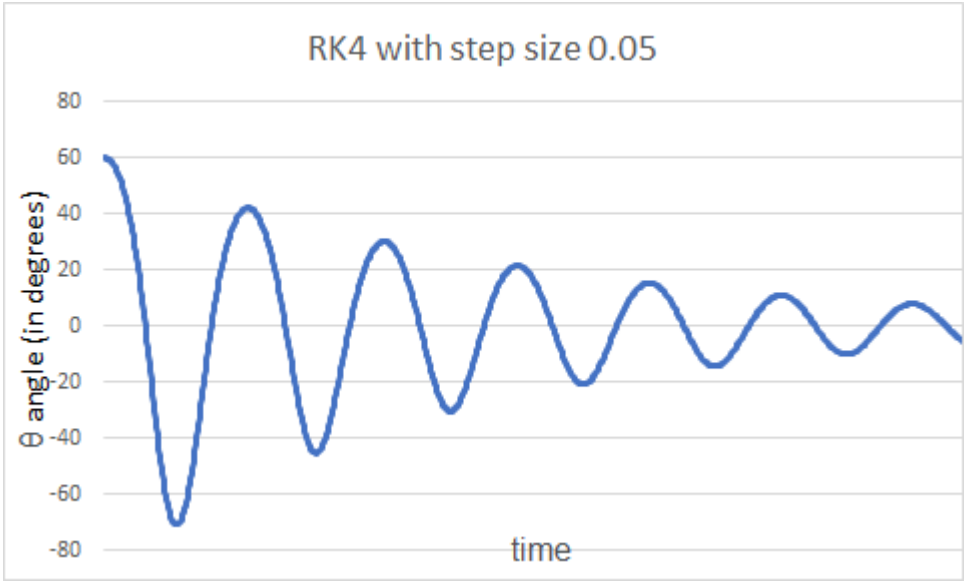
Graph 7: 4th-Order RungeKutta method with step-size 0.001, span time t = 50



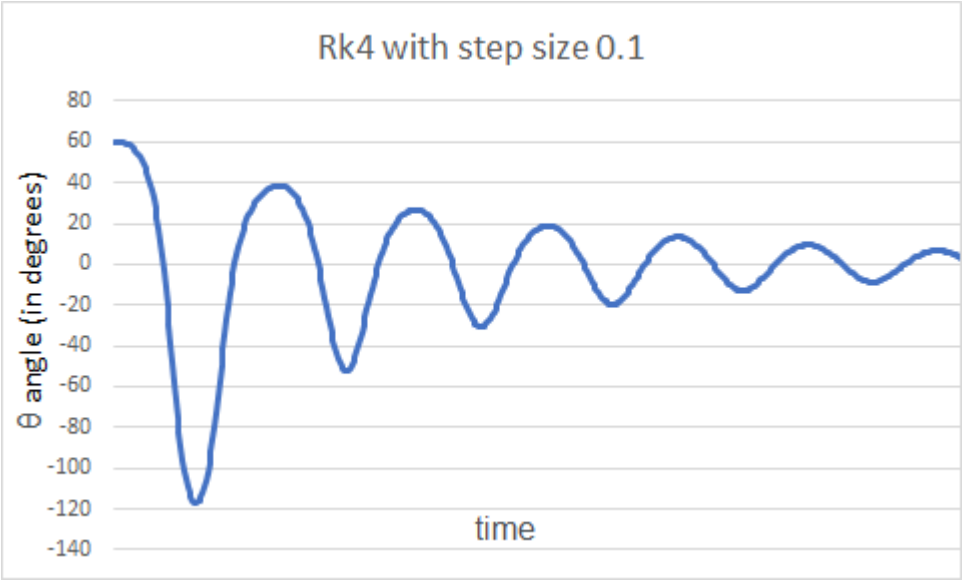
Graph 8: 4th-Order RungeKutta method with step-size 0.01, span time t = 50



Graph 9: 4th-Order RungeKutta method with step-size 0.02, span time t = 50



Graph 10: 4th-Order RungeKutta method with step-size 0.05, span time t = 50



Graph 11: 4th-Order RungeKutta method with step-size 0.1, span time t = 50

7. Discussion

When examining the graphes from section 6, we notice that Graph 6 does not plot the correct motion of the pendulum we would expect from the Verlet solver. This is due to an error in our implementation of this solver.

Instead of updating velocity as $v(t) = \frac{x(t+h)-x(t-h)}{2h}$, we instead implemented it as $v(t) = v(t-h) + \frac{a(t)}{2h}$ in our program.

This would explain not getting the results expected.

We expected from this experiment for the pendulums to reach a smaller and smaller angle after every swing, but the angle should remain similar from the previous oscillation. We can conclude by observing the change in the angle of θ in Graphe 6, and from our error in the implementation, that the velocity of the pendulum is not accurately calculated ; this makes the pendulum rest at a 0 degrees angle and 0 velocity after 2 oscillations.

The Euler method has a proper oscillation and decreases the θ angle steadily across the graphes 1, 2 and 4 (for step sizes 0.01, 0.02 and 0.001 respectively). This tells us that the implementation of the Euler method was successful.

However, for the graph 3 and 5 (step sizes 0.05 and 0.1), the angle increases after every iteration. This happens when the step size is too high, and therefore, the pendulum over-shoots by conserving too much velocity, despite the air resistance.

We estimate that in this case, the Euler solver is better to use a step size of 0.01, as it remains accurate without using too much computation as the step size 0.001 does.

For the 4th-Order Runge-Kutta method, we notice good oscillation for every graphe. This tells us that the implementation of the 4th-Order Runge-Kutta method has been accurately done. A step size of 0.02 would be appropriate for this solver given the more heavy computational power required.

8. Conclusion

The aim of this report was to simulate the Solar system as accurately as possible, while sending a spacecraft from earth to Titan, testing different solvers and examine how the accuracy of the results improve as the step-size is decreased and whether there is a minimum step-size needed for reasonable results. To derive answers, experiments were done with Euler's solver, 4-th order Runge Kutta solver and Verlet solver.

Two different solvers have been successfully implemented into the project. From our results, we conclude that the most accurately implemented is the 4-th order Runge-Kutta method. However, we can see from our results that our Verlet solver has not been properly implemented. We notice there are floating and different results depending on the time step. The accuracy of results increases as the step size decreases.

9. References

Burden, R., & Faires, J. (2012). *Numerical Methods, International Edition* (4th ed.). Brooks Cole.

Ciornei, M. C., Alaci, S., Ciornei, F. C., & Romanu, I. C. (2017). A method for the determination of the coefficient of rolling friction using cycloidal pendulum. *IOP Conference Series: Materials Science and Engineering*, *227*, 012027. https://doi.org/10.1088/1757-899x/227/1/012027

Greeley, R., & Iversen, J. D. (1987). *Wind as a Geological Process: On Earth, Mars, Venus and Titan (Cambridge Planetary Science Old, Series Number 4)*. Cambridge University Press. https://doi.org/10.1017/CBO9780511573071

Northon, K. (Ed.). (2019, June 27). *NASA's Dragonfly Mission to Titan Will Look for Origins, Signs of Life*. NASA. https://www.nasa.gov/press-release/nasas-dragonfly-will-fly-around-titan-looking-for-origins-signs-of-life.

Hubbard, W. B., & Buratti, Bonnie. (2020, April 15). Titan. *Encyclopedia Britannica*. https://www.britannica.com/place/Titan-astronomy

Huebl, A. (2010). *December 2010 Report number: FZJ-JSC-IB-2010-04Affiliation: FZ Juelich Authors: Axel Huebl Lawrence Berkeley National Laboratory* (FZJ-JSC-IB-2010-04). https://www.researchgate.net/publication/263125636_Implementation_of_a_Parallel_IO_Module_for_the_Particle-in-Cell_Code_PSC

Palasarn, S., & Toutip, W. (2011). Satellite Orbit Calculation Using Applied Numerical Method Of Ode. *The 22nd National Graduate Research Research Conference*, 1–7. https://home.kku.ac.th/wattou/research/research/paper08-wattanasurasit.pdf

Tan, D., & Chen, Z. (2012). On A General Formula of Fourth Order Runge-Kutta Method. *Journal of Mathematical Sciences & Mathematics Education*, *7*(2), 1–10. http://w.msme.us/2012-2-1.pdf

Voesenek, C. J. (2008). Implementing a Fourth Order Runge-Kutta Method for Orbit

Simulation, *14*, 1–3. http://spiff.rit.edu/richmond/nbody/OrbitRungeKutta4.pdf