Extended Essay

How do the Euler method and RK4 method compare in simulating the motion of a double pendulum on a computer?

Mathematics: Comparing two different methods for numerical analysis applied to a physical simulation.

Word Count: 3952

Personal Code: hlf398

Session: November 2019

PREFACE

This paper aims to compare three important methods, known as integrators, to digitally simulate the motion of a double pendulum. These are the Euler Method, the RK4 Method and the symplectic RK4 Method, and have been derived in a new sector of mathematics. The field of numerical analysis is the root to the methods being used to show the physical simulation chosen for this investigation. The choice of the double pendulum provides a system of chaotic motion that proves challenging to accurately simulate, while still requiring the program produced to follow simple physical laws, such as conservation of energy.

Throughout this investigation, two primary sources are referenced. These are the MyPhysicsLab website by Eric Neumann, and a paper from the University of Turkey on the double pendulum written by Abdalftah Elbori. The process for recording results in this paper involves creating three programs of a double pendulum that each use one of the different methods for its simulation.

Table of Contents

1 CONVENTION

1.1 UNITS

Many functions within this investigation require a basic understanding of units used in physics. Table 1 below describes these units.

Table 1: Units used throughout investigation.

Unit	Symbol	Definition
Metres	m	The Standard International unit for translational distance.
Radians	rad	The Standard International unit for angles and rotation.
Seconds	s	The Standard International unit for time.
Joules	J	The unit for energy, according to the International System of Units.

1.2 NOTATION

Defined below in Table 2 are the specific symbols used in any of the equations and other mathematical expressions written throughout this paper.

Table 2: Specialised notation used throughout investigation

Unit	Symbol	Definition
General Representation of Variables	X_b^a	Defines a certain variable X, where a represents the power to which it's raised and b represents the identifier for the object that the variable describes (in systems where multiple bodies may use the same variable type).
Function Notation	$f(x, \dots)$	Defines a function with variable parameters in the brackets. f denotes the type/purpose of the function. Note that constant variables do not need to be included in the parameters of a function, as they do not change within the program.

Table 2: Specialised notation used throughout investigation		
Unit	**Symbol**	**Definition**
Array Notation	$A[i]$	Defines a value within an array of name/purpose A and of index i within the array. An array is a programmer's version of a matrix or list of values that are stored under a common name.
Derivative Dot Notation	x', x''	This form of dot notation represents derivatives of a variable or function. Multiple "apostrophes" represent multiple derivatives.
Derivative Quotient Notation	$\frac{dy}{dx}$	The quotient form represents derivatives, where the x is the variable of the function and y is the function. The d represents a change or delta in the variables.
Multiplication	$a \cdot b$	To represent multiplication, a dot will be used between the two variables.
Setting a variable	$a \to b$	Used to show when the value of a variable b is being reassigned to another value a.

2 BACKGROUND

Presented with the system of a double pendulum being only acted on by gravity, with negligible friction of any kind, what is the most effective way to simulate its motion in a computer program? This issue is faced by people in varying professions. Game programmers use these systems if their game requires physics, or physicists will use them in theoretical simulations. A basic understanding of numerical analysis and integrators is required to be able to definitively compare the methods proposed.

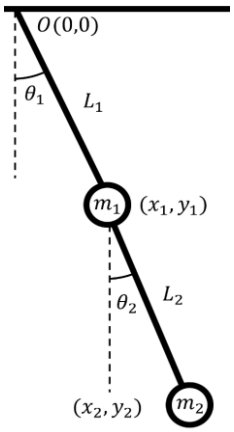


Figure 1: Setup of Double Pendulum

2.1 PHYSICS SIMULATIONS ON COMPUTERS

Computers do not operate continuously; they operate by quickly generating a discontinuous set of frames. This means that the interval of time between each frame needs to be considered, as any minute changes that might happen in that instant are not accounted for automatically. The challenge of minimising the inaccuracy by these intervals has spawned new areas in computing and mathematics.

2.2 NUMERICAL ANALYSIS

With the development of computers that could perform complicated algorithms, a branch of mathematics, name numerical analysis, was created to study their application in finding numerical solutions that included continuous variables (Atkinson K. E. 2017). It's application of algorithms in computers to solve functions with complex interactions of variables made it applicable for use in simulated models of the physical world.

2.3 Numerical Methods/Integrators

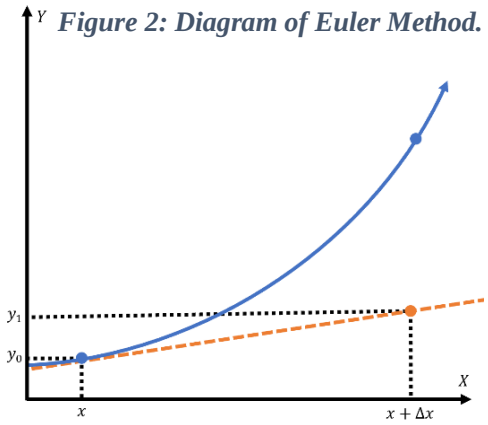


Figure 2: Diagram of Euler Method.

There are equations of motion for physical systems that cannot be directly solved for the state of the system at any point in time. Such equations are considered Ordinary Differential Equations (ODEs) and require the use of a numerical method/integrator to solve. Numerical methods/integrators are methods of numerical analysis where the integral of an ODE is calculated discontinuously over a certain step size. Instead of drawing a smooth curve, these methods draw the solutions in the form of connected line segments (As shown in Figure 2).

2.4 Chaotic Systems

The double pendulum is an example of a chaotic system. Chaotic systems are mechanical systems that are heavily dependent on their initial conditions and generate more intricate and individual motions as time progresses. This means that even the most infinitesimal change in the starting condition of the system will significantly vary the way it will operate as time progresses. In the double pendulum, a difference in starting angles of even a millionth of a radian over time means that the motion that it will follow will eventually be distinguishable as completely different.

2.5 Symplectic Motion

"Symplectic" refers to the idea that physical simulations are more accurate if they obey the law of conservation of energy. To support such a requirement, system-specific (i.e., cannot be applied to other systems) equations are calculated using a form of physics-based mathematics different from the traditional "Newtonian Mechanics", called "Lagrangian Mechanics". These equations consider the total energy within a system to be central to their solutions and in turn minimise any error caused by the intervals in simulations on computers.

3 NUMERICAL METHODS

Throughout this investigation, two integrators for simulating the double pendulum are compared in the form of three different methods. These are the Euler method, the Runge Kutta (also known as RK4) Method and the symplectic RK4 method.

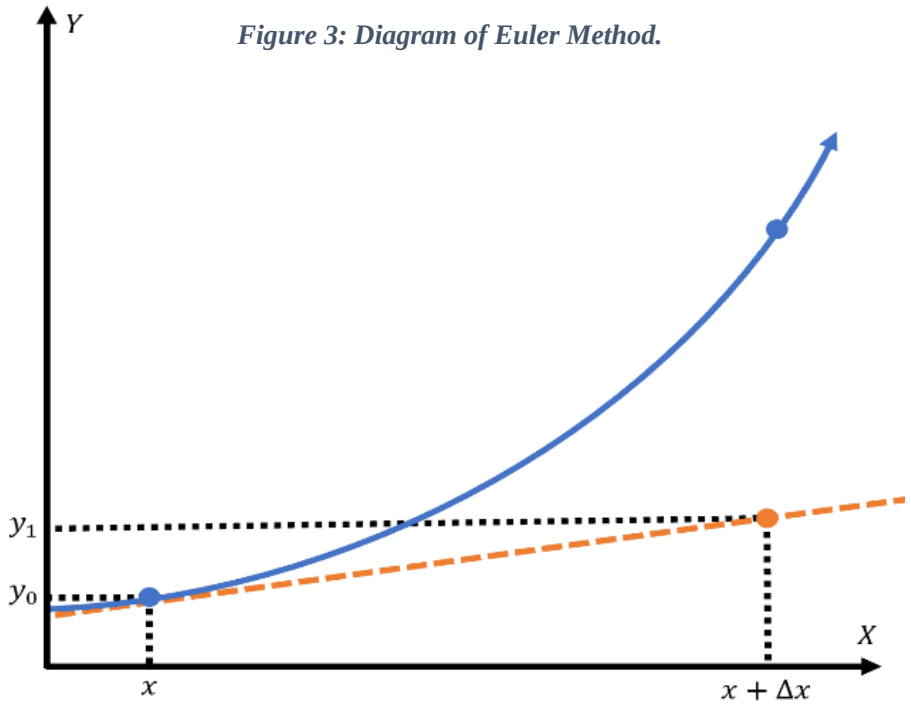
3.1 EULER METHOD

Named after the famous mathematician, the Euler method is one of the simplest integrators (Fiedler, G 2004). This method takes an initial or previous condition y_0, a step size Δx and a function for the rate of change of the initial condition $\frac{dy_0}{dx}$. To find the next step in the given function y_1, the rate of change for the previous step is added onto the previous step multiplied by its step size. This can be represented as the function:

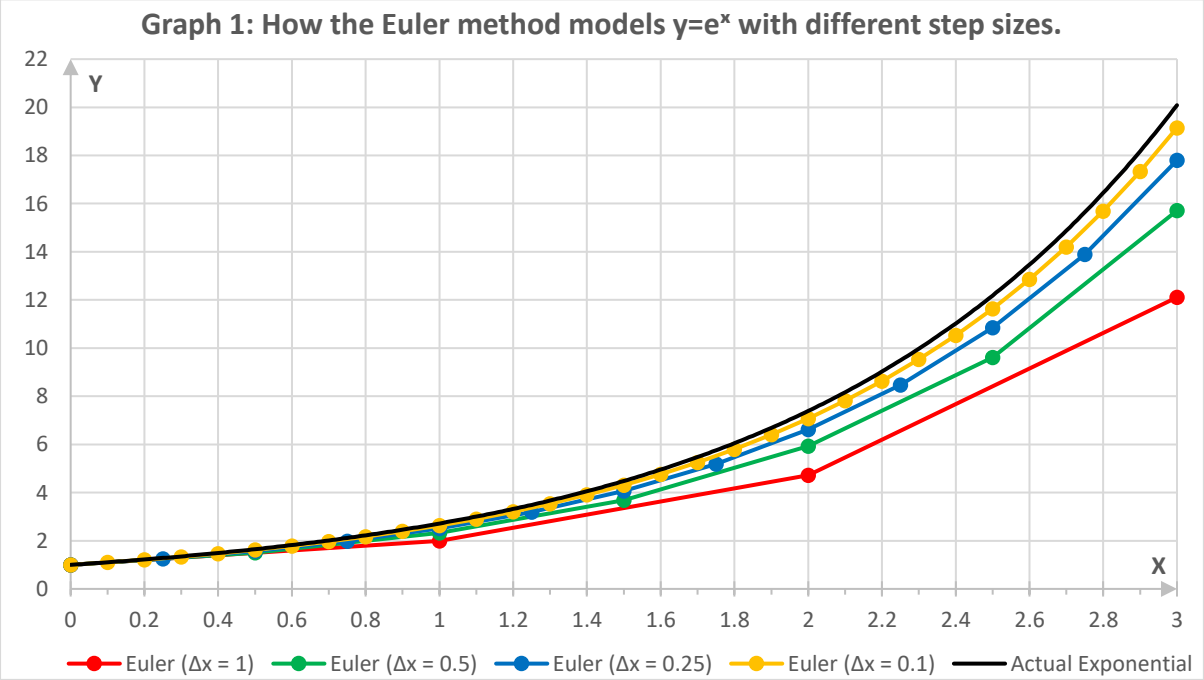
$$y_1 = y_0 + \frac{dy_0}{dx} \cdot \Delta x \tag{1}$$

The diagram shown in Figure 3 below represents how this method works graphically. The blue line represents the intended function, the point at (x, y_0) represents the previous state or initial condition, the orange line represents the tangent to the previous state as determined by its derivative, and the point at $(x + \Delta x, y_1)$ represents the next state determined by the Euler method.

Figure 3: Diagram of Euler Method.



Graph 1 shows the curves drawn by the different step sizes with the Euler method and the original curve of $y = e^x$. It demonstrates how step size plays an important role in providing accurate results. The graph also effectively shows the divergence of the predicted positions and the actual position of functions as steps progress.



3.2 Runge Kutta Methods

German physicists Carl Runge and Martin Kutta realised the inaccuracy in the Euler method as systems required smaller progressions of time in calculation steps and together developed a more accurate method for numerical analysis: the Runge Kutta method (Fiedler, G. 2004). Technically, there are multiple Runge Kutta methods. They are defined as integrators that use estimates of the gradient of a function to predict its next point. By this definition, the Euler method is a Runge Kutta method, as it uses the gradient at the previous point to determine the next point on a given curve.

Runge Kutta methods are classified based on their order, which is simply a measure of how many times per step the slope of the function is used. Following this definition, the Euler method is only a first order Runge Kutta method as it only uses a slope calculation once per step. The order of the method can be an indicator of both how accurate it can be at determining the needed solution, but also how efficient it is in calculations. The higher the order of the Runge Kutta method, the more accurate its results will be, but at the cost of requiring increasingly complex calculation.

3.3 THE RK4 METHOD

The physicists Runge and Kutta devised their own integrator. This is the RK4 method, which derives its name from the fact that it is a fourth order integrator. As it is of a higher order, it is bound to be more accurate, however will also use more processing power per step. Two of the three compared methods use the RK4 integrator, a method using the regular equations of motion for the double pendulum, and another using the symplectic equations of motion.

The method works by finding 4 different gradients (shown below) within each step and averaging them to find the next point. The general way to determine the next state of the function is:

$$y_1 = y_0 + \frac{\Delta x}{6} \cdot (s_a + 2 \cdot s_b + 2 \cdot s_c + s_d) \tag{2}$$

Where y_1 represents the state in the next step of the function, y_0 represents the previous step in the function, Δx represents the step size, and the values s_n represent the 4 different slopes that the method requires (n denoting the gradient). Each slope is determined slightly differently, as it accounts for different sub-steps within each step. These slopes are found using:

$$s_a = \frac{dy_0}{dx} \tag{3}$$

$$s_b = \frac{d(y_0 + \frac{\Delta x}{2} \cdot s_a)}{d(x + \frac{\Delta x}{2})} \tag{4}$$

$$s_c = \frac{d(y_0 + \frac{\Delta x}{2} \cdot s_b)}{d(x + \frac{\Delta x}{2})} \tag{5}$$

$$s_d = \frac{d(y_0 + \Delta x \cdot s_c)}{d(x + \Delta x)} \tag{6}$$

Each slope s_n represents a different sub-step that is calculated in the method, with n representing the step of the gradient being found (numbering is avoided as it may imply chronological order to the steps). s_a calculates the slope at the previous state of the function (or at the initial condition), s_b calculates the slope of a point halfway between the next and previous states using the previous state's slope, s_c calculates the slope halfway between the next and previous states using the slope found in s_b, and s_d calculates the slope at the next step using the slope found in s_c. The theory behind this method is that

the average of these slopes (where s_b and s_c are counted twice each due to their calculation from half the step size) is like the slope of the points separated by the given step size.

Figure 4 below shows how this method works graphically. In the diagram on the plot (left), the red line represents the RK4 method's averaged slope, the orange point represents the previous step or initial condition, the orange dashed line represents the slope at sub-step a, the green point and dotted lines represent the state and slope at sub-step b, the yellow point and dotted lines represent the state and slope at sub-step c, and the purple point and dotted line represent the state and slope at sub-step d. The slopes are then averaged, shown on the right portion of the diagram, to determine the average slope between the previous state and the next state.

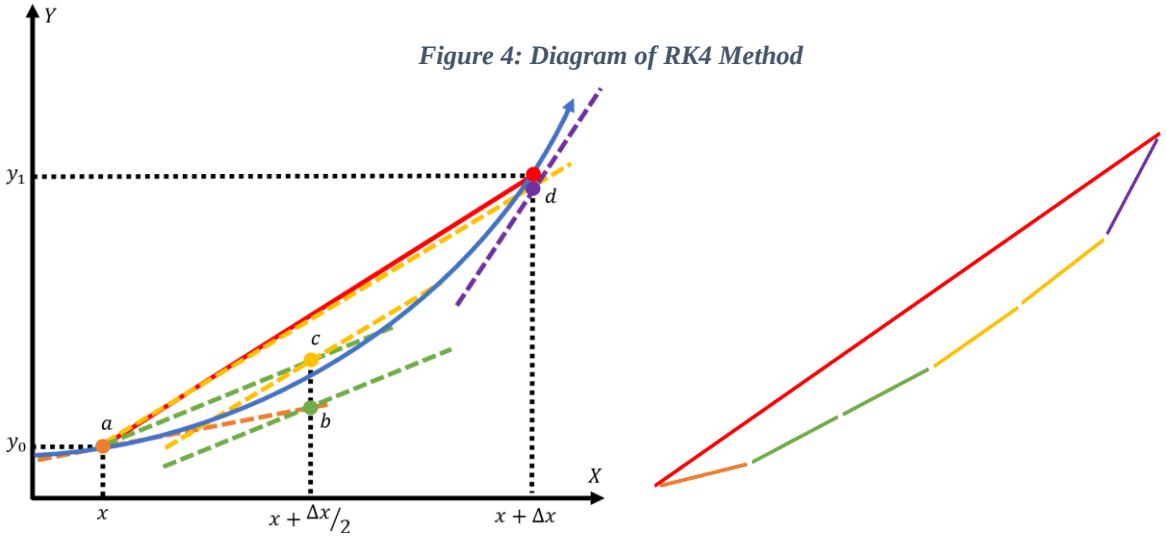
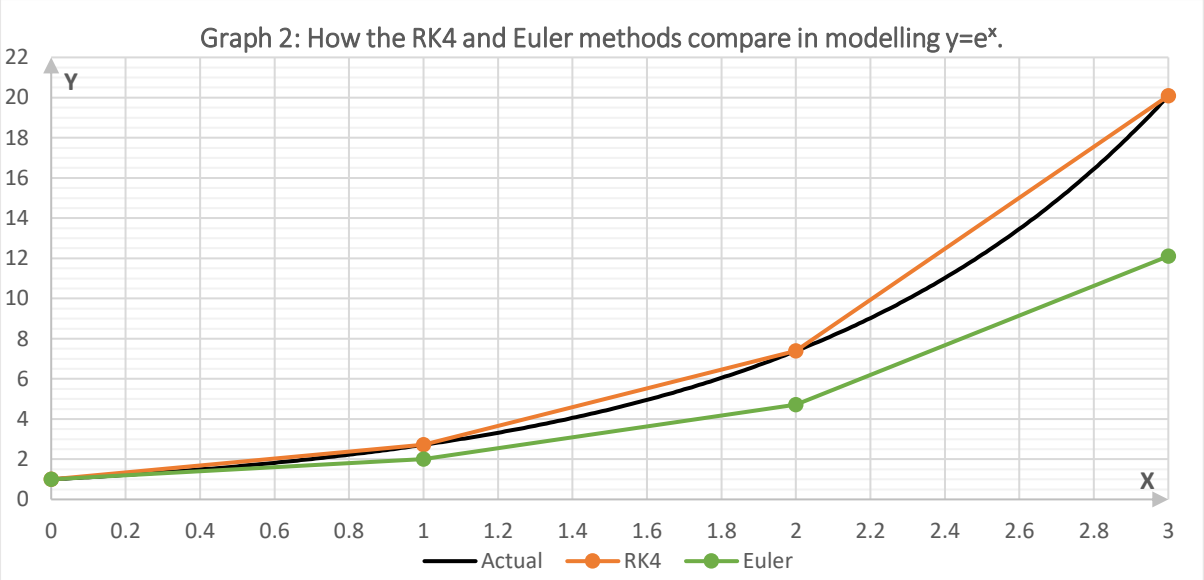


Figure 4: Diagram of RK4 Method

When graphically applied to the function $y = e^x$ with a step size $\Delta x = 1$ and compared to the Euler method, the RK4 result for the next point after three steps is still quite close to the actual value for the function (see Graph 2 below).



3.4 RK4 WITH MULTIPLE VARIABLES

One important requirement of the RK4 method's sub-steps to note is the need to find the whole state for each step. This means that any variable that is required to calculate the ODE and will change as each step progresses needs to be changed according to the step it is in. This becomes clearer when the equations of motion of the double pendulum are defined.

4 THE DOUBLE PENDULUM MODEL

4.1 THE DOUBLE PENDULUM DIAGRAM

For simplicity of calculations in this investigation, the following model for the double pendulum is assumed. The bobs are attached by massless sticks, meaning they will always be a fixed distance from each other. In the defined system, the bobs are defined with masses m_1 and m_2 attached to two sticks of lengths L_1 and L_2 at angles of θ_1 and θ_2 as shown in figure 5. The pendulum is suspended at the origin O and have coordinates (x_1, y_1) and (x_2, y_2). On top of this, the acceleration due to gravity is defined by g.

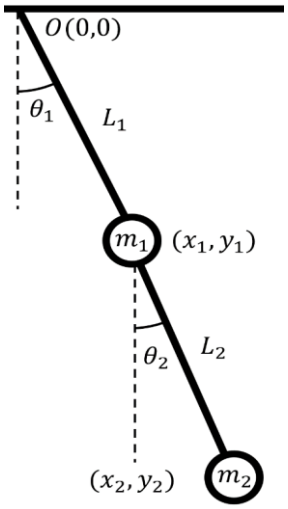


Figure 5: Setup of Double Pendulum

4.2 Coordinates of The Pendulum Bobs

From the angles and lengths of the sticks, basic trigonometry can be applied to define the coordinates of the bobs as:

$$x_1 = L_1 \cdot \sin(\theta_1) \tag{7}$$

$$y_1 = -L_1 \cdot \cos(\theta_1) \tag{8}$$

$$x_2 = x_1 + L_2 \cdot \sin(\theta_2) \tag{9}$$

$$y_2 = y_1 - L_2 \cdot \cos(\theta_2) = -L_1 \cdot \cos(\theta_1) - L_2 \cdot \cos(\theta_2) \tag{10}$$

4.3 Newtonian Equations of Angular Motion

From (7), (8), (9) and (10), Erik Neumann from MyPhysicsLab derives the equations for the angular acceleration of the double pendulum (Erik N. 2002):

$$\theta_1'' = \frac{-g \cdot (2 \cdot m_1 + m_2) \cdot \sin(\theta_1) - m_2 \cdot g \cdot \sin(\theta_1 - 2 \cdot \theta_2) - 2 \cdot \sin(\theta_1 - \theta_2) \cdot m_2 \cdot (\theta_2'^2 \cdot L_2 + \theta_1'^2 \cdot L_1 \cdot \cos(\theta_1 - \theta_2))}{L_1 \cdot (2 \cdot m_1 + m_2 - m_2 \cdot \cos(2 \cdot (\theta_1 - \theta_2))} \tag{11}$$

$$\theta_2'' = \frac{2 \cdot \sin(\theta_1 - \theta_2) \cdot (\theta_1'^2 \cdot L_1 \cdot (m_1 + m_2) + g \cdot (m_1 + m_2) \cdot \cos(\theta_1) + \theta_2'^2 \cdot L_2 \cdot m_2 \cdot \cos(\theta_1 - \theta_2))}{L_2 \cdot (2 \cdot m_1 + m_2 - m_2 \cdot \cos(2 \cdot (\theta_1 - \theta_2))} \tag{12}$$

4.4 Symplectic Equations of Angular Motion

The symplectic equations of motion are derived differently, given the requirement that the total energy within the system needs to be conserved. Using a known property of the Lagrangian of the double pendulum, Abdalftah Elbori from the University of Turkey finds the angular velocities and the rates of change of the momenta of the bobs (p_1 and p_2) in terms of variables A and B (Abdalftah E. 2017):

$$\theta_1' = \frac{L_2 \cdot p_1 - L_1 \cdot p_2 \cdot \cos(\theta_1 - \theta_2)}{L_1^2 \cdot L_2 \cdot \left(m_1 + m_2 \cdot sin^2(\theta_1 - \theta_2)\right)} \tag{13}$$

$$\theta_2' = \frac{L_1 \cdot (m_1 + m_2) \cdot p_2 - L_2 \cdot m_2 \cdot p_1 \cdot \cos(\theta_1 - \theta_2)}{L_1 \cdot L_2^2 \cdot \left(m_1 + m_2 \cdot sin^2(\theta_1 - \theta_2)\right)} \tag{14}$$

$$p_1' = -(m_1 + m_2) \cdot g \cdot L_1 \cdot \sin(\theta_1) - A + B \tag{15}$$

$$p_2' = -m_2 \cdot g \cdot L_2 \cdot \sin(\theta_2) + A - B \tag{16}$$

$$A = \frac{p_1 \cdot p_2 \cdot \sin(\theta_1 - \theta_2)}{L_1 \cdot L_2 \cdot \left(m_1 + m_2 \cdot sin^2(\theta_1 - \theta_2)\right)} \tag{17}$$

$$B = \frac{L_2^2 \cdot m_2 \cdot p_1^2 + L_1^2 \cdot (m_1 + m_2) \cdot p_2^2 - L_1 \cdot L_2 \cdot m_2 \cdot p_1 \cdot p_2 \cdot \cos(\theta_1 - \theta_2)}{2 \cdot L_1^2 \cdot L_2^2 \cdot \left(m_1 + m_2 \cdot sin^2(\theta_1 - \theta_2)\right)^2} \cdot \sin(2 \cdot (\theta_1 - \theta_2)) \tag{18}$$

These can then be applied to a method such as the Runge Kutta to find the new angle of each pendulum bob after the step.

4.5 CHECKING ACCURACY

As the functions defined above cannot be integrated to find the state at any point in time of the pendulum, other methods must be used to determine how accurate and effective the integrator used will be. One such way is by measuring the total energy of the system as time progresses. If the system's total energy ever goes above or below the initial gravitational potential energy of the system, then it can be assumed that the integrator is not accurate enough.

In the double pendulum system, the gravitational potential energy can be determined based on the y values of the bobs. Assuming the pendulum is suspended at a height equal to the sum of the lengths of the rods ($L_1 + L_2$), the total GPE can be found as:

$$GPE = GPE_1 + GPE_2$$

$$GPE = m_1 \cdot g \cdot (L_1 + L_2 + y_1) + m_2 \cdot g \cdot (L_1 + L_2 + y_2)$$

$$GPE = m_1 \cdot g \cdot (L_1 + L_2 - L_1 \cdot \cos(\theta_1)) + m_2 \cdot g \cdot (L_1 + L_2 - L_1 \cdot \cos(\theta_1) - L_2 \cdot \cos(\theta_2)) \quad (19)$$

Next what is needed is the kinetic energy of the system, which Abdalftah Elbori calculates when finding his own set of symplectic equations as (Abdalftah E. 2017):

$$KE = \frac{1}{2} \cdot (m_1 + m_2) \cdot L_1^2 \cdot \theta'^2_1 + \frac{1}{2} \cdot m_2 \cdot \left(L_2^2 \cdot \theta'^2_2 + 2 \cdot L_1 \cdot L_2 \cdot \theta'_1 \cdot \theta'_2 \cdot \cos(\theta_1 - \theta_2)\right) \quad (20)$$

5 APPLICATION OF INTEGRATORS

With the needed equations of motion for the double pendulum defined, it is now time to apply them to each method in a program.

5.1 PROGRAMMING SETUP

The programming environment selected for this process is "Processing", a free online Java-based software for visual and artistic experiences (Processing.org 2019). Each Processing sketch is divided into 2 main sections: a setup and a draw. The setup is where the initial conditions for the double pendulum are input and the initial GPE is calculated. The draw section is what loops as the code runs. Each run of the draw produces a frame when the code is running.

For all the methods used, the setup code is the same, and includes (19) to find the initial energy of the system. All variables except for the initial angles will be set as defined in Table 4. Throughout testing, these will not be changed. However, changes to these are a point for further investigation.

Table 4: Initial conditions applied to all tests

Variables	$\theta_1'', \theta_2'' \ (\frac{rad}{s^2})$	$\theta_1', \theta_2' \ (\frac{rad}{s})$	$p_1, p_2 \ (\frac{kg\ m}{s})$	$p_1', p_2' \ (\frac{kg\ m}{s^2})$	$L_1, L_2 \ (m)$	$m_1, m_2 \ (kg)$	$g \ (\frac{m}{s^2})$
Value	0	0	0	0	1	2	9.8

In the draw loop for every method, (19) and (20) will be summed to find the total energy in the system.

5.2 EULER METHOD

The Euler method is quite simple to implement programmatically. The non-constant variables needed are the angular accelerations of the bobs (θ_1'' and θ_2''), the angular velocities of the bobs (θ_1' and θ_2'), and the angles of the bobs themselves (θ_1 and θ_2).

Using the equations for the angular acceleration for each bob (11) and (12), the following functions must be run in every draw loop in the order:

$$\frac{-g \cdot (2 \cdot m_1 + m_2) \cdot \sin(\theta_1) - m_2 \cdot g \cdot \sin(\theta_1 - 2 \cdot \theta_2) - 2 \cdot \sin(\theta_1 - \theta_2) \cdot m_2 \cdot (\theta_2'^2 \cdot L_2 + \theta_1'^2 \cdot L_1 \cdot \cos(\theta_1 - \theta_2))}{L_1 \cdot (2 \cdot m_1 + m_2 - m_2 \cdot \cos(2 \cdot (\theta_1 - \theta_2))} \rightarrow \theta_1''$$

$$\frac{2 \cdot \sin(\theta_1 - \theta_2) \cdot (\theta_1'^2 \cdot L_1 \cdot (m_1 + m_2) + g \cdot (m_1 + m_2) \cdot \cos(\theta_1) + \theta_2'^2 \cdot L_2 \cdot m_2 \cdot \cos(\theta_1 - \theta_2))}{L_2 \cdot (2 \cdot m_1 + m_2 - m_2 \cdot \cos(2 \cdot (\theta_1 - \theta_2))} \rightarrow \theta_2''$$

$$\theta_1' + \Delta x \cdot \theta_1'' \rightarrow \theta_1'$$

$$\theta_2' + \Delta x \cdot \theta_2'' \rightarrow \theta_2'$$

$$\theta_1 + \Delta x \cdot \theta_1' \rightarrow \theta_1$$

$$\theta_2 + \Delta x \cdot \theta_2' \rightarrow \theta_2$$

Once the angle variables are assigned, they can be substituted into the coordinate equations (7), (8), (9), (10) to draw the bobs on the screen. These new values also need to be substituted into the energy equations to find a new value for total energy.

5.3 RK4 METHOD

5.3.1 Setup

Application of the RK4 method is more complicated, as the variables all need to be updated for every sub-step (because the whole state of the pendulum is required for each sub-steps). This requires the use of arrays to save all the variables for each state. The arrays are indexed as described in Table 5 (see next page) to minimise confusion when later accessing the arrays in code. Keep in mind that array indices start at 0.

Table 5: Indexing used for arrays in RK4 method.

Variable Type	Angle		Angular Velocity		Angular Acceleration	
Index	0	1	2	3	4	5
Variable	θ_1	θ_2	θ_1'	θ_2'	θ_1''	θ_2''

Four arrays indexed in this way are required, allowing each state needed for the sub-steps to be stored. At the start of every step, an array named $previous$ is created, containing the values indexed above for the previous state within the double pendulum.

5.3.2 Implementation

First, the functions to find the angular accelerations in the pendulum for each sub-step need to be denoted based off (11) and (12):

$$\theta_1''(\theta_1, \theta_2, \theta'_1, \theta'_2) = \frac{-g \cdot (2 \cdot m_1 + m_2) \cdot \sin(\theta_1) - m_2 \cdot g \cdot \sin(\theta_1 - 2 \cdot \theta_2) - 2 \cdot \sin(\theta_1 - \theta_2) \cdot m_2 \cdot (\theta_2'^2 \cdot L_2 + \theta_1'^2 \cdot L_1 \cdot \cos(\theta_1 - \theta_2))}{L_1 \cdot (2 \cdot m_1 + m_2 - m_2 \cdot \cos(2 \cdot (\theta_1 - \theta_2))}$$

$$\theta_2''(\theta_1, \theta_2, \theta'_1, \theta'_2) = \frac{2 \cdot \sin(\theta_1 - \theta_2) \cdot (\theta_1'^2 \cdot L_1 \cdot (m_1 + m_2) + g \cdot (m_1 + m_2) \cdot \cos(\theta_1) + \theta_2'^2 \cdot L_2 \cdot m_2 \cdot \cos(\theta_1 - \theta_2))}{L_2 \cdot (2 \cdot m_1 + m_2 - m_2 \cdot \cos(2 \cdot (\theta_1 - \theta_2))}$$

Next each sub-step needs to be calculated, beginning with a, which is arguably the simplest, given that most of its contents are the previous state and do not need to be calculated. The array S_a is defined with the values (use Table 5 for reference in array indexing and see Table 2 on array notation):

$$S_a[0] = previous[0]$$

$$S_a[1] = previous[1]$$

$$S_a[2] = previous[2]$$

$$S_a[3] = previous[3]$$

$$S_a[4] = \theta_1''(S_a[0], S_a[1], S_a[2], S_a[3])$$

$$S_a[5] = \theta_2''(S_a[0], S_a[1], S_a[2], S_a[3])$$

Then the state for sub-step b needs to be found and used to calculate the angular accelerations for the sub-step, which is saved to array S_b:

$$S_b[0] = previous[0] + \frac{\Delta x}{2} \cdot S_a[2]$$

$$S_b[1] = previous[1] + \frac{\Delta x}{2} \cdot S_a[3]$$

$$S_b[2] = previous[2] + \frac{\Delta x}{2} \cdot S_a[4]$$

$$S_b[3] = previous[3] + \frac{\Delta x}{2} \cdot S_a[5]$$

$$S_b[4] = \theta_1''(S_b[0], S_b[1], S_b[2], S_b[3])$$

$$S_b[5] = \theta_2''(S_b[0], S_b[1], S_b[2], S_b[3])$$

The same is done for the state of sub-step c and then used to calculate the angular accelerations for the sub-step which is saved to array S_c:

$$S_c[0] = previous[0] + \frac{\Delta x}{2} \cdot S_b[2]$$

$$S_c[1] = previous[1] + \frac{\Delta x}{2} \cdot S_b[3]$$

$$S_c[2] = previous[2] + \frac{\Delta x}{2} \cdot S_b[4]$$

$$S_c[3] = previous[3] + \frac{\Delta x}{2} \cdot S_b[5]$$

$$S_c[4] = \theta_1''(S_c[0], S_c[1], S_c[2], S_c[3])$$

$$S_c[5] = \theta_2''(S_c[0], S_c[1], S_c[2], S_c[3])$$

Sub-step d is different because the step size is no longer halved. Other than this, the same applies (values are saved in array S_d):

$$S_d[0] = previous[0] + \Delta x \cdot S_c[2]$$

$$S_d[1] = previous[1] + \Delta x \cdot S_c[3]$$

$$S_d[2] = previous[2] + \Delta x \cdot S_c[4]$$

$$S_d[3] = previous[3] + \Delta x \cdot S_c[5]$$

$$S_d[4] = \theta_1''(S_d[0], S_d[1], S_d[2], S_d[3])$$

$$S_d[5] = \theta_2''(S_d[0], S_d[1], S_d[2], S_d[3])$$

Once all these arrays are completely calculated, the program can then set the new variables for the angles, the angular velocity and angular acceleration using the sub-steps:

$$\frac{(S_a[4] + 2 \cdot S_b[4] + 2 \cdot S_c[4] + S_d[4])}{6} \rightarrow \theta_1''$$

$$\frac{(S_a[5] + 2 \cdot S_b[5] + 2 \cdot S_c[5] + S_d[5])}{6} \rightarrow \theta_2''$$

$$\frac{(S_a[2] + 2 \cdot S_b[2] + 2 \cdot S_c[2] + S_d[2])}{6} \rightarrow \theta_1'$$

$$\frac{(S_a[3] + 2 \cdot S_b[3] + 2 \cdot S_c[3] + S_d[3])}{6} \rightarrow \theta_2'$$

$$\theta_1 + \frac{\Delta x \cdot \theta'_1}{6} \rightarrow \theta_1$$

$$\theta_2 + \frac{\Delta x \cdot \theta'_2}{6} \rightarrow \theta_2$$

With the new angles set, they can then be substituted into the equations for the coordinates of the pendulum bobs (7), (8), (9), (10) to display them on the screen.

5.4 SYMPLECTIC RK4 METHOD

5.4.1 Setup

The symplectic RK4 method requires a different set of functions as the equations derived for it do not consider angular acceleration. As such, arrays need to be index differently. Table 6 describes the indexation for arrays in this method.

Table 6: Indexing used for arrays in symplectic RK4 method.

Variable Type	Angle		Momentum		Angular Velocity		Rate of Change of Momentum	
Index	0	1	2	3	4	5	6	7
Variable	θ_1	θ_2	p_1	p_2	θ_1'	θ_2'	p_1'	p_2'

In the same way as with the RK4 method, four arrays are used (one for each sub-step) and each step begins with the array $previous$ which stores the previous state of the pendulum.

5.4.2 Implementation

As explained in Section 4.4 (Symplectic Equations of Angular Motion), The equations that need to be declared for this method are (13), (14), (15), (16), (17) and (18):

$$A(\theta_1, \theta_2, p_1, p_2) = \frac{p_1 \cdot p_2 \cdot \sin(\theta_1 - \theta_2)}{L_1 \cdot L_2 \cdot \left(m_1 + m_2 \cdot sin^2(\theta_1 - \theta_2)\right)}$$

$$B(\theta_1, \theta_2, p_1, p_2) = \frac{L_2^2 \cdot m_2 \cdot p_1^2 + L_1^2 \cdot (m_1 + m_2) \cdot p_2^2 - L_1 \cdot L_2 \cdot m_2 \cdot p_1 \cdot p_2 \cdot \cos(\theta_1 - \theta_2)}{2 \cdot L_1^2 \cdot L_2^2 \cdot \left(m_1 + m_2 \cdot sin^2(\theta_1 - \theta_2)\right)^2} \cdot \sin(2 \cdot (\theta_1 - \theta_2))$$

$$p_1'(\theta_1, \theta_2, p_1, p_2) = -(m_1 + m_2) \cdot g \cdot L_1 \cdot \sin(\theta_1) - A(\theta_1, \theta_2, p_1, p_2) + B(\theta_1, \theta_2, p_1, p_2)$$

$$p_2'(\theta_1, \theta_2, p_1, p_2) = -m_2 \cdot g \cdot L_2 \cdot \sin(\theta_2) + A(\theta_1, \theta_2, p_1, p_2) - B(\theta_1, \theta_2, p_1, p_2)$$

$$\theta_1'(\theta_1, \theta_2, p_1, p_2) = \frac{L_2 \cdot p_1 - L_1 \cdot p_2 \cdot \cos(\theta_1 - \theta_2)}{L_1^2 \cdot L_2 \cdot \left(m_1 + m_2 \cdot sin^2(\theta_1 - \theta_2)\right)}$$

$$\theta_2'(\theta_1, \theta_2, p_1, p_2) = \frac{L_1 \cdot (m_1 + m_2) \cdot p_2 - L_2 \cdot m_2 \cdot p_1 \cdot \cos(\theta_1 - \theta_2)}{L_1 \cdot L_2^2 \cdot \left(m_1 + m_2 \cdot sin^2(\theta_1 - \theta_2)\right)}$$

Next, as with the regular RK4 method, each sub-step needs to be calculated individually, beginning with sub-step a (values will be saved in array S_a and indexed according to Table 6):

$$S_a[0] = previous[0]$$

$$S_a[1] = previous[1]$$

$$S_a[2] = previous[2]$$

$$S_a[3] = previous[3]$$

$$S_a[4] = \theta_1'(S_c[0], S_c[1], S_c[2], S_c[3])$$

$$S_a[5] = \theta_1'(S_c[0], S_c[1], S_c[2], S_c[3])$$

$$S_a[6] = p_1'(S_c[0], S_c[1], S_c[2], S_c[3])$$

$$S_a[7] = p_2'(S_c[0], S_c[1], S_c[2], S_c[3])$$

Then, the state of sub-step b is calculated and applied to the needed equations (in array S_b):

$$S_b[0] = previous[0] + \frac{\Delta x}{2} \cdot S_a[4]$$

$$S_b[1] = previous[1] + \frac{\Delta x}{2} \cdot S_a[5]$$

$$S_b[2] = previous[2] + \frac{\Delta x}{2} \cdot S_a[6]$$

$$S_b[3] = previous[0] + \frac{\Delta x}{2} \cdot S_a[7]$$

$$S_b[4] = \theta_1'(S_b[0], S_b[1], S_b[2], S_b[3])$$

$$S_b[5] = \theta_1'(S_b[0], S_b[1], S_b[2], S_b[3])$$

$$S_b[6] = p_1'(S_b[0], S_b[1], S_b[2], S_b[3])$$

$$S_b[7] = p_2'(S_b[0], S_b[1], S_b[2], S_b[3])$$

Sub-step c is calculated next with values saved in array S_c:

$$S_c[0] = previous[0] + \frac{\Delta x}{2} \cdot S_b[4]$$

$$S_c[1] = previous[1] + \frac{\Delta x}{2} \cdot S_b[5]$$

$$S_c[2] = previous[2] + \frac{\Delta x}{2} \cdot S_b[6]$$

$$S_c[3] = previous[0] + \frac{\Delta x}{2} \cdot S_b[7]$$

$$S_c[4] = \theta_1'(S_c[0], S_c[1], S_c[2], S_c[3])$$

$$S_c[5] = \theta_1'(S_c[0], S_c[1], S_c[2], S_c[3])$$

$$S_c[6] = p_1'(S_c[0], S_c[1], S_c[2], S_c[3])$$

$$S_c[7] = p_2'(S_c[0], S_c[1], S_c[2], S_c[3])$$

Then sub-step d is calculated, with values stored in array S_d:

$$S_d[0] = previous[0] + \Delta x \cdot S_b[4]$$

$$S_d[1] = previous[1] + \Delta x \cdot S_b[5]$$

$$S_d[2] = previous[2] + \Delta x \cdot S_b[6]$$

$$S_d[3] = previous[0] + \Delta x \cdot S_b[7]$$

$$S_d[4] = \theta_1'(S_d[0], S_d[1], S_d[2], S_d[3])$$

$$S_d[5] = \theta_1'(S_d[0], S_d[1], S_d[2], S_d[3])$$

$$S_d[6] = p_1'(S_d[0], S_d[1], S_d[2], S_d[3])$$

$$S_d[7] = p_2'(S_d[0], S_d[1], S_d[2], S_d[3])$$

Finally, the values in the sub-step arrays can be applied to (2) to find the values for the next state of the pendulum:

$$\frac{(S_a[6] + 2 \cdot S_b[6] + 2 \cdot S_c[6] + S_d[6])}{6} \rightarrow p_1'$$

$$\frac{(S_a[7] + 2 \cdot S_b[7] + 2 \cdot S_c[7] + S_d[7])}{6} \rightarrow p_2'$$

$$\frac{(S_a[4] + 2 \cdot S_b[4] + 2 \cdot S_c[4] + S_d[4])}{6} \rightarrow \theta_1'$$

$$\frac{(S_a[5] + 2 \cdot S_b[5] + 2 \cdot S_c[5] + S_d[5])}{6} \rightarrow \theta_2'$$

$$p_1 + \frac{\Delta x \cdot p'_1}{6} \to p_1$$

$$p_2 + \frac{\Delta x \cdot p'_2}{6} \to p_2$$

$$\theta_1 + \frac{\Delta x \cdot \theta'_1}{6} \to \theta_1$$

$$\theta_2 + \frac{\Delta x \cdot \theta'_2}{6} \to \theta_2$$

These can then be substituted into the coordinate equations (7), (8), (9), (10) to display the pendulum on the screen.

6 DATA FOR ANALYSIS

The testing conditions set in Table 7 were applied to the Euler, RK4 and symplectic RK4 methods and screenshots were taken at 10 second intervals for 100 seconds. The screenshots were automatically taken by the program for maximum precision. The following conditions accounts for how different step sizes in the methods affect their accuracy and provide two different initial condition scenarios. Either the initial angle is at an extreme point where there will be a large transfer of energy with a fall (where the angles are both equal to π) and where there is a simple start with no large energy transfer. With the initial upright position (both angles equal to π) and the larger step size (0.1 seconds per step), Test 2 creates a worst-case scenario to accurately simulate in the pendulum's motion.

Table 7: Initial conditions used in testing

Test #	Initial Angles (Radians)		Step Size Δx (s)
	$\boldsymbol{\theta_1}$	$\boldsymbol{\theta_2}$	
1	$\pi/3$	$\pi/4$	0.1
2	π	π	0.1
3	$\pi/3$	$\pi/4$	0.025
4	π	π	0.025

6.1 Recorded Results

The following tables and graphs show the recorded energy loss as time progressed for each of the simulation methods derived earlier.

Table 8: How the energy lost in the double pendulum varies over time when simulated using the Euler Method.

Time (s)	Energy Lost (J)			
	Test 1	Test 2	Test 3	Test 4
0	1.38	0.00	0.09	0.00
10	-73.04	104.71	-3.22	41.67
20	NA	94.28	-9.50	90.85
30	NA	65.85	-20.59	105.56
40	NA	NA	-45.90	104.10
50	NA	NA	-82.57	103.31
60	NA	NA	-5.68	100.64
70	NA	NA	20.62	99.44
80	NA	NA	23.42	98.08
90	NA	NA	23.98	95.06
100	NA	NA	24.39	92.95

Note that NA denotes that the pendulum does not appear on the screen (see Reference Frame 1).

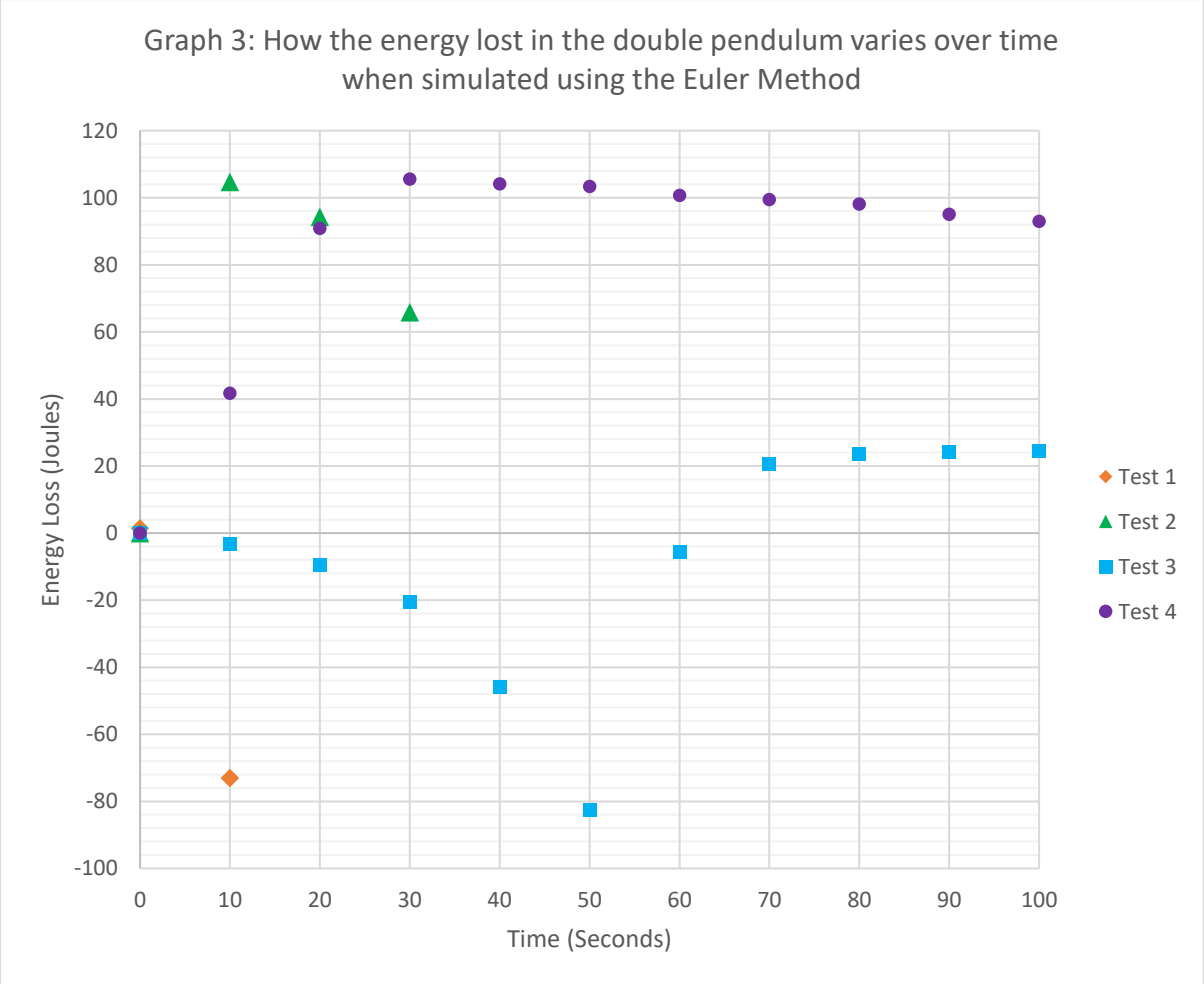


Table 9: How the energy lost in the double pendulum varies over time when simulated using the RK4 Method.				
Time (s)	Energy Lost (J)			
	Test 1	Test 2	Test 3	Test 4
0	0.00	0.00	0.00	0.00
10	0.10	60.79	0.00	0.15
20	0.17	75.16	0.00	0.82
30	0.26	82.23	0.00	2.36
40	0.33	91.39	0.00	2.64
50	0.40	95.07	0.00	3.04
60	0.48	99.97	0.00	3.79
70	0.54	102.06	0.00	4.67
80	0.60	103.77	0.00	5.71
90	0.67	105.41	0.00	6.14
100	0.73	106.18	0.00	6.66

Graph 4: How the energy lost in the double pendulum varies over time when simulated using the RK4 Method.

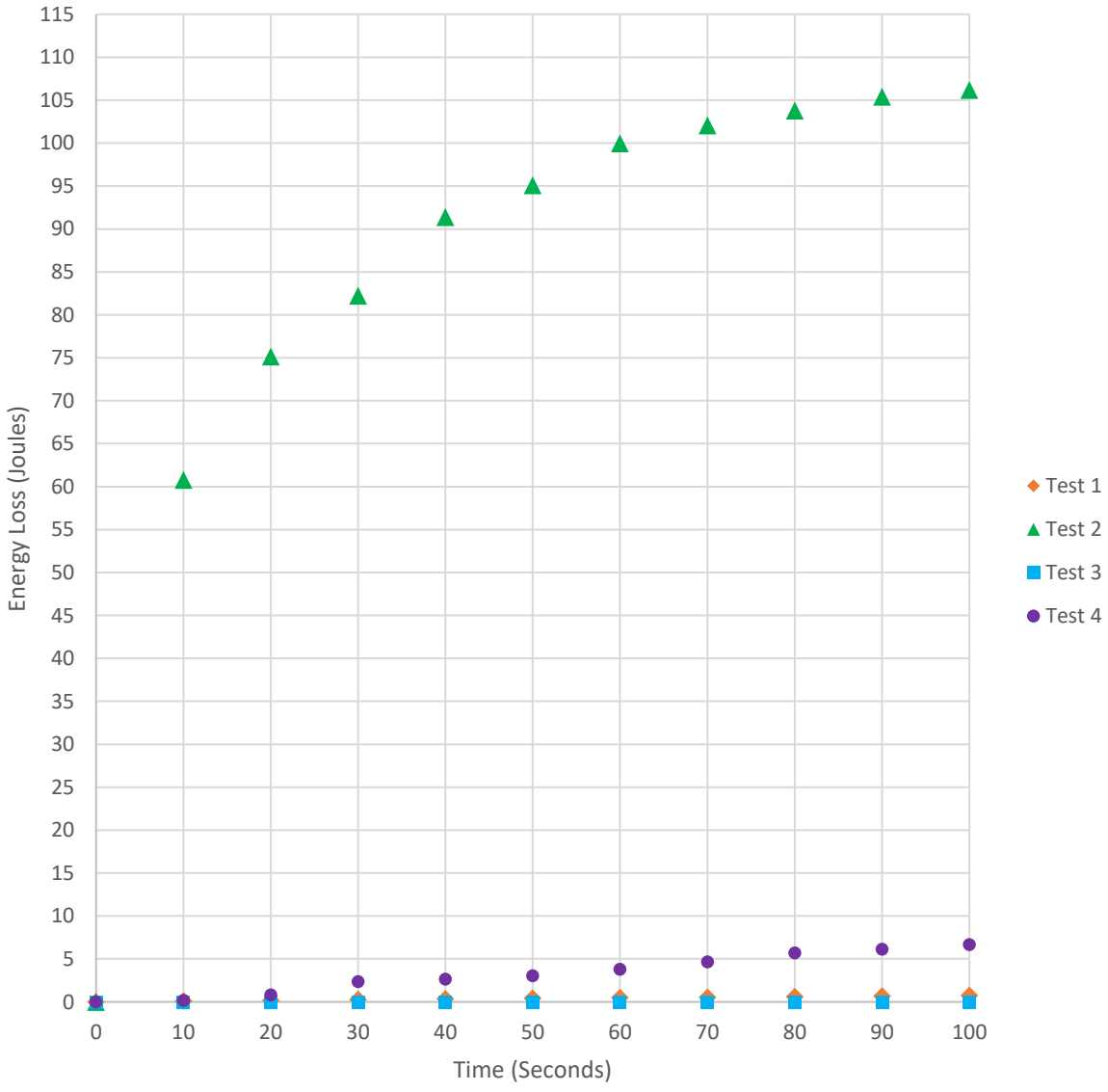
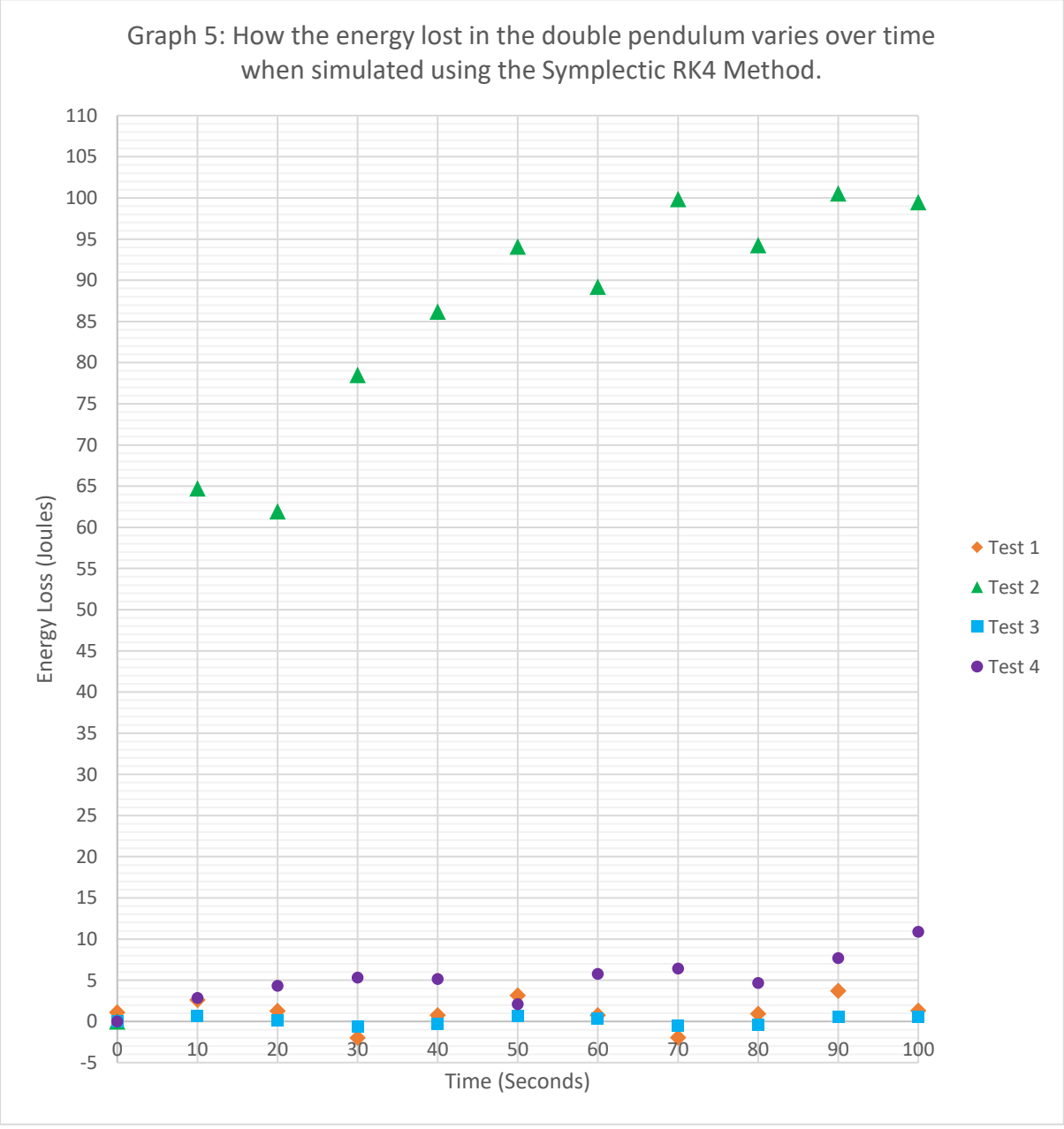
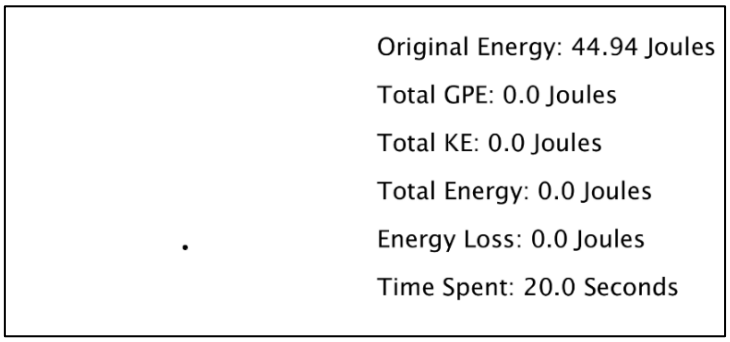


Table 10: How the energy lost in the double pendulum varies over time when simulated using the Symplectic RK4 Method.				
Time (s)	Energy Lost (J)			
	Test 1	Test 2	Test 3	Test 4
0	1.08	0.00	0.07	0.00
10	2.62	64.74	0.63	2.83
20	1.25	61.96	0.11	4.32
30	-1.99	78.49	-0.63	5.32
40	0.75	86.21	-0.31	5.16
50	3.17	94.08	0.65	2.10
60	0.74	89.23	0.35	5.78
70	-1.95	99.87	-0.53	6.43
80	0.92	94.26	-0.45	4.66
90	3.69	100.56	0.53	7.69
100	1.30	99.47	0.52	10.89

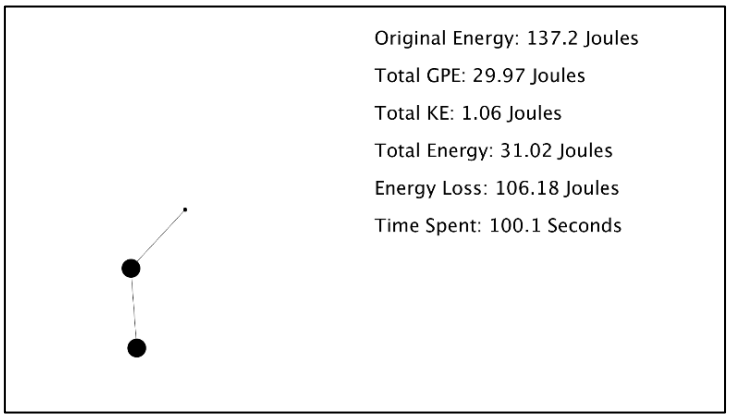


6.2 REFERENCE FRAMES

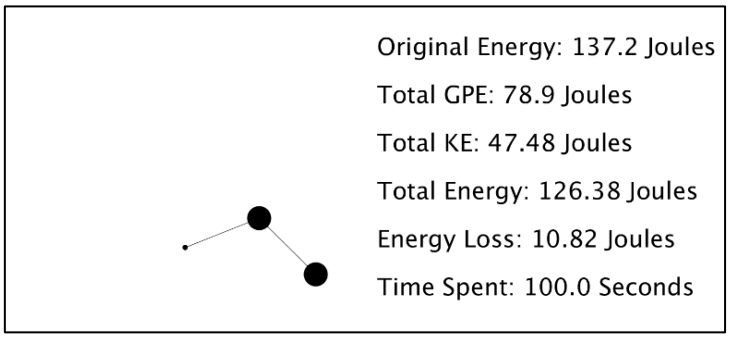
Reference frames such as the ones below were used to collect the data in the previous section. The following frames showed examples different points for discussion in the Section 7 (Conclusion).



Reference Frame 1: Using the Euler method, and following conditions from Test 1, the double pendulum disappears from the screen.



Reference Frame 2: Using the RK4 method, and following conditions from Test 2, which creates the conditions for the least accuracy, the energy loss is maximised.



Reference Frame 3: Using the Symplectic RK4 method, and following conditions from Test 4, the average energy loss is very minimal

7 CONCLUSION

Assuming calculated energy loss allows for the most effective measurement of the accuracy of the methods used in the double pendulum simulation, the RK4 method is most accurate, unless in extreme situations (such as Test 2), where its symplectic counterpart is better.

To begin with, the Euler method is not accurate as in some tests the pendulum no longer displayed on the screen (See Reference Frame 1) and due to too great a variability in recorded energy losses (see Graph 3). Table 8 shows that the double pendulum stopped working properly in Tests 1 and 2, which both featured the higher step size. When the program is running, the pendulum begins to experience extreme acceleration due to an interesting side effect of the chaotic nature of the pendulum. As the Euler method becomes increasingly accurate as the step size decreases, it is logical to assume that the larger step size may have caused a scenario where the accelerations of the pendulum calculated by the method were higher than they would be in reality (see Figure 6 below). Another factor to note is the gain of energy that the Euler method creates. Table 8 shows that the pendulum in Test 3 gained energy as time progressed (indicated by negative values), which breaks a fundamental law of physics that energy cannot be created. After 50 seconds, for example, the pendulum had gained almost 83 Joules of energy. This factor and the results from tests 1 and 2 show that the Euler method is not the most accurate.

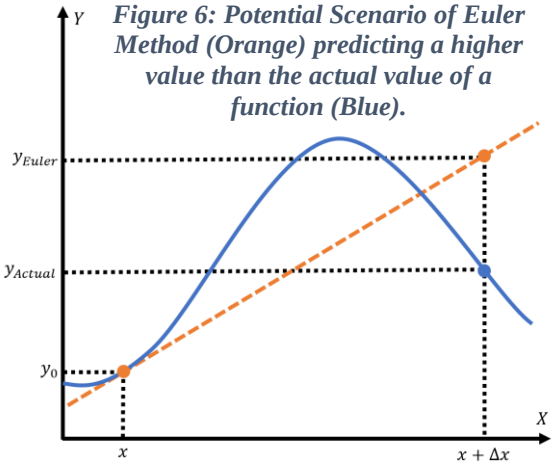


Figure 6: Potential Scenario of Euler Method (Orange) predicting a higher value than the actual value of a function (Blue).

The RK4 method was the most accurate as the energy lost on all the tests, except for Test 2, were the lowest out of all the methods (See Table 9). There is a correlation in this relationship between the time elapsed in the program, and the energy lost, as shown in Tests 1 and 4 of Table 9 and in Graph 4. Test 3 in this method was the most accurate out of all the other tests on any other method, showing 0.00

Joules of energy loss. Though this doesn't necessarily mean that there was no energy loss, only that it occurred very slowly. The numbers that the program produces are truncated to two decimal places, but indications from other tests on this method suggest that energy was likely lost, albeit not very quickly (see Graph 4). Test 2 results show there is a large initial loss of energy as the pendulum begins its fall from an upright stance. As Test 2 creates a worst-case scenario to simulate, it shows that in extreme cases, the RK4 method can be inaccurate. Other than this test, the RK4 method's slow rate of energy loss makes it the most accurate under normal circumstances.

The symplectic RK4 showed a different pattern in the energy lost. Tests 1 and 3 in Table 10 show that there is a fluctuation in the energy loss around 0 Joules (see Graph 5). This means that on average, there is little to no energy loss in this method, possibly meaning it can work accurately over a longer period when compared to the other methods in this investigation. Test 2 for this method was the most accurate out of all the methods, because although the pendulum still experiences the large loss of energy in the initial seconds of the test, the subsequent loss of energy is lower than for the RK4 method or the Euler method (compare Graphs 3, 4 and 5). Although the fluctuations mean that the energy lost in this method is the lowest, on average, it cannot be predicted how they affect the pendulums chaotic motion. It is important to note that the chaotic nature of this system means that small inaccuracies can cause large differences in motion over time.

In conclusion, data collected from each method when compared suggest that the most accurate method for simulating the chaotic motion of a double pendulum is the RK4 method, unless there are initial conditions in place that cause a large initial transfer of energy, in which case the symplectic RK4 method proves to minimise energy loss in Joules. The results from this investigation prompt interesting points to further test or analyse in the double pendulum simulations produced. Perhaps testing how changes to initial conditions other than the starting angles (such as angular velocities, rod lengths or masses) may affect the pendulum's motion as well as the accuracy of the simulation method. It should also be addressed that this investigation only compares three methods for the simulation of the double pendulum. However, there are many other Runge Kutta methods of potentially higher orders that should be considered.

8 BIBLIOGRAPHY

Atkinson, K. E. (May 31, 2017). *Numerical Analysis.* Retrieved April 22, 2019 from https://www.britannica.com/science/numerical-analysis

Cheever, E. (2005). *Fourth Order Runge Kutta*. Retrieved April 22, 2019 from https://lpsa.swarthmore.edu/NumInt/NumIntFourth.html

Elbori, Abdalftah. (2017). Simulation of Double Pendulum. Journal of Software Engineering and Simulation. 3. 01-13. (accessed April 22, 2019).

Euler's Method. *Brilliant.org*. Retrieved 17:05, April 22, 2019, from https://brilliant.org/wiki/eulers-method/

Fiedler, G. (2004 June 1). Integration Basics [Web log post]. Retrieved April 22, 2019, from https://gafferongames.com/post/integration_basics/

Neumann, E. (2002, February). *Double Pendulum*. Retrieved April 22, 2019 from https://www.myphysicslab.com/pendulum/double-pendulum-en.html