# Investigating the use of numerical methods in physical modelling with initial value problems

To what extent are Runge-Kutta methods useful for solving initial value problems associated with modelling physical phenomena?

Extended Essay in Mathematics
Word count: 3801

## Table of Contents

Note: $\frac{d^n y}{dy^n}$ and $y^{(n)}$ both signify the $n^{th}$ derivative of y, and are used interchangeably.

All figures are the author's own work unless otherwise stated. Graphs of data and functions produced using Desmos, GeoGebra, and/or Excel.

## 1 Introduction and approach

In the sciences and beyond, ordinary differential equations (ODEs) are frequently used, in the form of *initial value problems* (IVPs), as tools for modelling complex phenomena. However, many ODEs cannot be solved analytically (i.e., using algebraic techniques) to produce a useful model, requiring the use of *numerical methods*—iterative algorithms that can yield approximate solutions (Soetaert et al., 2012). This essay explores Runge-Kutta methods, a foundational class of numerical methods, to answer the research question: **"To what extent are Runge-Kutta methods useful for solving initial value problems associated with modelling physical phenomena?"**. This topic is worthy of investigation, as many ODEs used in physical modelling are unsolvable, and the Runge-Kutta methods are among the most widely used numerical methods in physics and beyond (Workineh et al., 2024)

As such, ODEs and IVPs are introduced as powerful tools for modelling problems in physics, and one such problem, the ODE describing the motion of a simple pendulum, is introduced and developed from first principles. Its analytical intractability motivates discussion of numerical methods, and three Runge-Kutta methods are introduced: the Euler method, the improved Euler method, and the fourth-order Runge-Kutta method. While the Euler method was the original numerical method described, its newer relative, the fourth-order Runge-Kutta method, is still in common use.

These methods are applied to the pendulum model, and their effectiveness and necessity is established by comparing the numerical solutions to real-world data and a simplified model. Although their limitations in modelling certain specific scenarios specific are considered, it is ultimately argued that their accuracy, necessity, and applicability to real-world problems makes them significantly useful in modelling physical phenomena.

# 2 Background

## 2.1 Ordinary differential equations

An *ordinary differential equation* (ODE) involves a dependent variable (typically $y$), its derivatives, and a single independent variable (typically $x$) (Simmons, 1991). The general form of an ODE is

$$F\big(x, y, y', y'', \dots, y^{(n)}\big) = 0, \tag{1}$$

where $F$ denotes some arbitrary function of $x, y$, and the derivatives of $y$ (Zill, 2012, p. 3). ODEs are also classified by their order, where an $n^{\text{th}}$-order equation's highest-order derivative is $y^{(n)}$. For example, the differential equation

$$y' + ay = 0,$$

where $a \in \mathbb{R}$ is some constant, is a first-order ODE, as the order of the highest derivative of $y$ present, $y'$, is 1 (Polyanin, 2017).

## 2.2 Solutions to ODEs

Generally, in modelling with ODEs, the aim is to identify *solutions*. A solution to an $n^{\text{th}}$-order ordinary differential equation over some interval $I$ is defined as any function $\phi$, defined over that interval $I$ with at least $n$ derivatives continuous on $I$, that when substituted into an ODE reduces it to an identity (i.e., $1 = 1$) (Zill, 2012, p. 5). For example, the first-order ODE for exponential growth, which describes the rate of change of $y$ as proportional to $y$,

$$\frac{dy}{dt} = ky, \tag{2}$$

has a solution over the interval $I = [-\infty, \infty]$ of

$$y = e^{kt}, \tag{3}$$

as the exponential function in (3) is defined for all $t \in I$, and reduces (2) to an identity:

$$\frac{d}{dt}\big[e^{kt}\big] = k\big(e^{kt}\big)$$

$$\therefore \frac{d}{dt}[y] = ke^{kt}$$

$$\therefore ke^{kt} = ke^{kt}$$
$$1 = 1.$$

## 2.3 Analytical solutions

ODEs can be solved either *analytically* or *numerically* (Braun, 1993, pp. 96–98). Analytical approaches use algebraic manipulation and indefinite integration to give explicit, "closed-form" solutions in terms of elementary functions (polynomial, exponential, and trigonometric functions, and their inverse functions [Weisstein, n.d.]), as in solution (3) above (Hermann & Saravi, 2014). These can be applied to ODEs matching specific known patterns (Kolk & Lerman, 1992). For example, one of the simplest analytical techniques, the *separation of variables* method, only works for first-order ODEs that can be written in the form

$$\frac{dy}{dx} = g(x)h(y),$$

where $g$ and $h$ are functions of the independent and dependent variable respectively (Zill, 2012, pp. 46–47). Equation (2) is already in this format, taking $g(x) = 1$ and $h(y) = ky$. Dividing by $h(y)$, then integrating, yields a solution for $y$:

$$\frac{1}{y}\frac{dy}{dt} = k$$
$$\int \frac{1}{y}\frac{dy}{dt}\,dt = \int k\,dt$$
$$\int \frac{1}{y}\,dy = \int k\,dt$$
$$\ln|y| = kt + c_1$$
$$|y| = e^{kt+c_1}$$
$$\therefore y = \pm e^{kt}e^{c_1} = Ce^{kt},$$

where $c_1$ is the constant of integration, and the coefficient $\pm e^{c_1}$ is written $C$. As $c_1 \in \mathbb{R}, \Longrightarrow C \in \mathbb{R}$. Thus, $y = Ce^{kt}$ is called a one-parameter *family of solutions*, describing infinitely many solutions for all $C \in \mathbb{R}$. Conversely, the solution $y = e^{kt}$ is called a *particular solution*, as it is a specific solution for $C = 1$ (Zill, 2012, p. 7).

## 2.4 Initial value problems

An *initial-value problem* (IVP) is a first-order ODE coupled with a side-condition that specifies the solution's value at a point (Zill, 2012, pp. 13–14). The general first-order IVP takes the form

$$\frac{dy}{dx} = f(x, y), \qquad y(x_0) = y_0,$$

where the side-condition $y(x_0) = y_0$ specifies the solution's value at $x_0$ (Braun, 1993). IVPs are often used for modelling in the sciences, as the initial conditions describe of the state of a system at a specific time (Griffiths & Higham, 2010). For instance, Newton's law of cooling states that the rate of cooling of a body is proportional to the difference between the temperature of the body ($T$) and the temperature of its surroundings ($T_a$) (Herman, 2018):

$$\frac{dT}{dt} \propto T - T_a$$
$$\therefore \frac{dT}{dt} = -k(T - T_a),$$

where a cooling rate constant $k$ is introduced, $k > 0$. This ODE becomes an IVP when the initial temperature $T_0$ is defined, imposing the *initial condition* $T(0) = T_0$. This can be solved using separation of variables as before to obtain $T$ in terms of $t$:

$$\frac{1}{T - T_a}\frac{dT}{dt} = -k$$

$$\int \frac{1}{T - T_a}\frac{dT}{dt}\,dt = \int -k\,dt$$

$$\ln|T - T_a| = -kt + c_1$$

$$T - T_a = \pm e^{c_1}e^{-kt}$$

$$T(t) = Ce^{-kt} + T_a,$$

rewriting the constant $\pm e^{c_1}$ as $C$. Then, by substituting the initial condition, $C$ can be found in terms of $T_0$, thereby giving a family of solutions for this specific IVP with one parameter, $T_a$:

$$T(0) = T_0$$
$$\therefore Ce^{-k(0)} + T_a = T_0$$
$$C + T_a = T_0$$

$$C = T_0 - T_a$$

$$\therefore T(t) = (T_0 - T_a)e^{-kt} + T_a.$$

Note that while $T_a$ is a *parameter*, $T_0$ is not, as it is defined in relation to the solution at a specific point. With this, given values for $T_a$ and $T(0) = T_0$, one can find the particular solution for a system. For example, the temperature of a coffee cup starting at a temperature of $T_0 = 50°\text{C}$ that cools in a room at $T_a = 10°\text{C}$, with an arbitrary cooling rate constant[1] of $k = 0.1\ h^{-1}$, is given by

$$T(t) = (T_0 - T_a)e^{-kt} + T_a$$

$$= 40e^{-0.1t} + 10.$$

## 3 Numerical methods

Many ODEs used in practice are analytically unsolvable, as they do not match any patterns for which solution techniques are known (Hermann & Saravi, 2014, p. 3). To obtain solutions in these scenarios, one can use *numerical methods,* iterative algorithms which provide approximate solutions to IVPs in the form of a set of points in the Cartesian plane (Zill, 2012, p. 78). This essay focuses on the class of *Runge-Kutta* methods, which comprise the oldest and simplest numerical method, Euler's method, and its generalisations (Butcher, 2008, p. 93). The formal definition of Runge-Kutta methods will be presented after developing the prototypical Euler method; however, they can be conceptualised as techniques that approximate the value of the solution at discrete points within an interval, using the derivative function and the previous point to estimate the value of the next (Griffiths & Higham, 2010). Though other numerical methods exist, the Runge-Kutta methods are arguably the most significant, due to their simplicity and widespread applicability (Workineh et al., 2024).

### 3.1 The Euler method

The Euler method involves approximating the solution by taking small steps from known initial conditions, using the slope given by the differential equation (Griffiths & Higham, 2010, pp. 19–22).

[1] The value of this constant can be found for a given system with a similar substitution if the temperature of the object at another time is known.

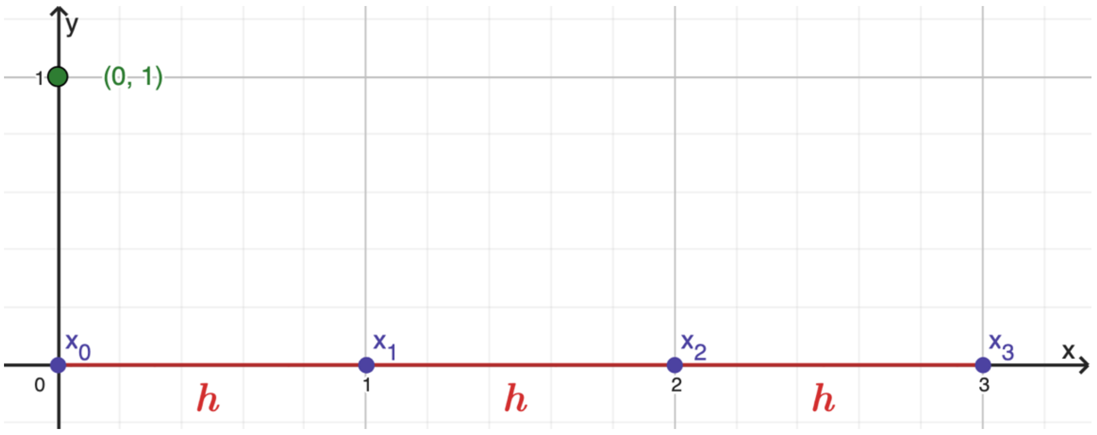
Take the general first-order IVP,

$$\frac{dy}{dx} = f(x, y), \qquad y(x_0) = y_0,$$

with the aim of finding a solution function $y(x)$, defined over some useful interval $[a, b]$ that begins at the initial condition point $x_0$ (such that $a = x_0$). Divide the interval into successive points from $x_0$ to $x_n$, with a separation of $h$, such that $x_n = x_0 + nh$, and $x_{n+1} - x_n = h$. This is shown in Figure 1 for $h = 1$ over an interval $x \in [0,3]$, for initial condition $y(0) = 1$ (shown in green).

**Figure 1**

*Division of interval for Euler method*



The initial condition is the first point, $(x_0, y_0)$, and so can be plotted as the beginning of the solution curve. However, for the next point, the value of $y(x_1)$ is unknown. Note that the differential equation gives the slope of the line tangent to the solution curve at an initial point $(x_0, y_0)$ as

$$\frac{dy}{dx} = f(x_0, y_0), \tag{4}$$

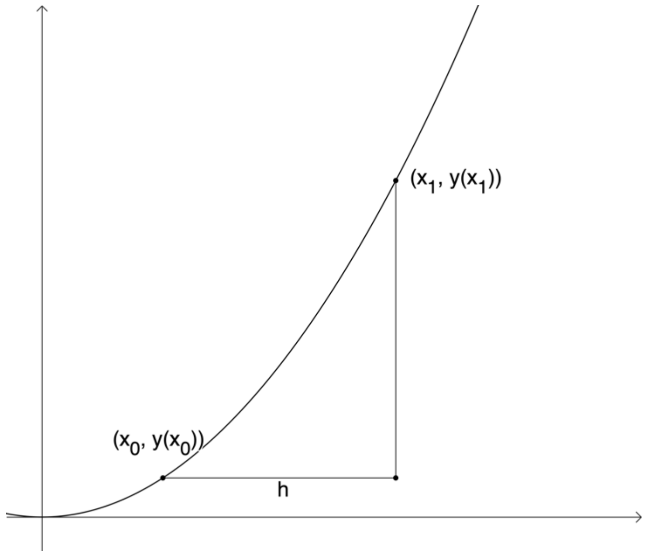
so, the slope at the point $(x_0, y_0)$ is given by $f(x_0, y_0)$. Furthermore, from the definition of the derivative, the slope at a point $x_0$ approaches the *difference quotient* between $y(x_0)$ and another value $y(x_1)$ as the separation between $x_0$ and $x_1$ $h$ approaches zero (see Figure 2).

$$\frac{dy}{dx} = \lim_{h \to 0} \frac{y(x_1) - y(x_0)}{h}$$

$$\therefore \frac{dy}{dx} \approx \frac{y(x_1) - y(x_0)}{h} \text{ for small } h.$$

**Figure 2**

*Definition of the derivative*



Substituting the value of the derivative given by (4), $y(x_1)$ is thus given:

$$f(x_0, y_0) \approx \frac{y(x_1) - y(x_0)}{h}$$
$$\therefore y(x_1) \approx y(x_0) + hf(x_0, y_0).$$

Writing the approximated value of $y(x_1)$ as $y_1$ yields the equality

$$y_1 = y_0 + f(x_0, y_0)h.$$

This process can be repeated using the approximated point $(x_1, y_1)$ as the 'initial point' to repeat the process for the next, and so on, leading to the *general form* of the Euler method,

$$y_{n+1} = y_n + f(x_n, y_n)h, \qquad y_0 = y(x_0).$$

Note that though $y_0 = y(x_0)$ from the initial condition, in general $y_n \approx y(x_n)$. Thus, finding any $y_n$ is an iterative process based on calculating $y_1, y_2, \dots, y_{n-1}$, and the accuracy increases with decreasing $h$, because the difference quotient approaches the derivative as $h \to 0$ (Herman, 2018, p. 76).

For example, take the Newton's law of cooling IVP, for the earlier case of a coffee cup starting at a temperature of $T_0 = 50°\text{C}$, with ambient temperature $T_a = 10°\text{C}$ and rate constant $k = 0.1$:

$$\begin{aligned}\frac{dT}{dt} &= -k(T - T_a) \\ &= -0.1(T - 10), \qquad T(0) = 50. \end{aligned} \tag{5}$$

To solve this IVP with the Euler method over an interval $t \in [0,5]$, with $h = 1$, start by calculating the slope at $t = 0$, which is given by the derivative:

$$\begin{aligned}\frac{dT}{dt} &= -0.1(T(0) - 10) \\ &= -0.1(50 - 10) \\ &= -4.\end{aligned}$$

Then, the approximate value of $T(t_1)$, $T_1$, can be found from this slope using the Euler method:

$$T_1 = T_0 + \frac{dT}{dt}h$$

$$= 50 - 4 \times 1 = 46.$$

The process can be repeated using this new point (1,46) to calculate subsequent values of $T_n$, as shown in Table 1. The graph of this against the analytical solution is shown in Figure 3 below. The Euler method solution appears to closely fit the analytical solution, which is supported by the high coefficient of determination (computed using Desmos) of $R^2 = 0.9941$.

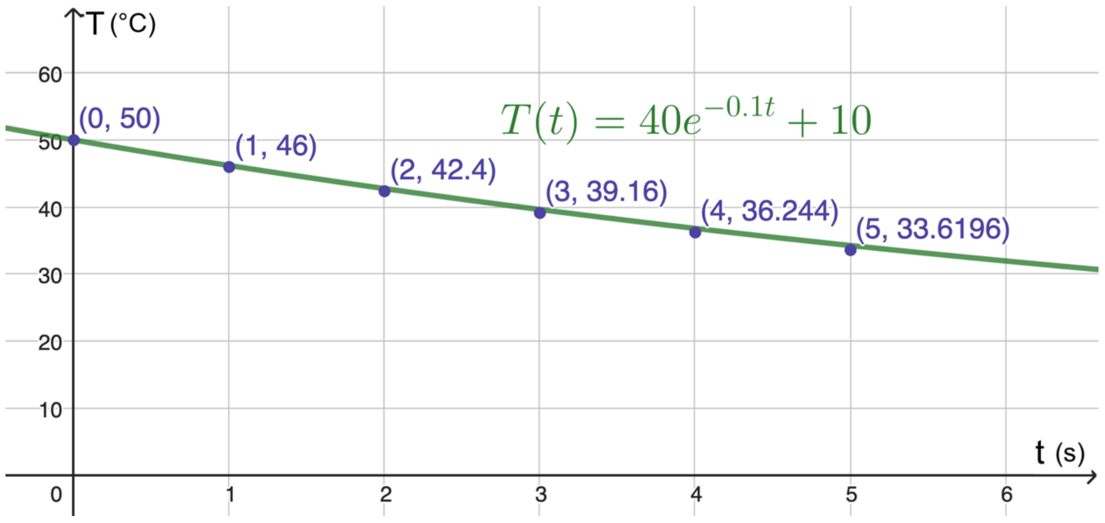
**Table 1**

*Euler method solution for cooling IVP*

| $\boldsymbol{t}$ (h) | $\boldsymbol{T}$ (°C) | $\boldsymbol{dT/dt}$ |
|---|---|---|
| 0 | 50 | $-0.1 \times (50 - 10) = -4.0$ |
| 1 | $50 + (-4.0 \times 1) = 46$ | $-0.1 \times (46 - 10) = -3.6$ |
| 2 | $46 + (-3.6 \times 1) = 42.4$ | $-0.1 \times (42.4 - 10) = -3.24$ |
| 3 | $42.4 + (-3.24 \times 1) = 39.16$ | $-0.1 \times (39.16 - 10) = -2.916$ |
| 4 | $39.16 + (-2.916 \times 1) = 36.244$ | $-0.1 \times (36.244 - 10) = -2.6244$ |
| 5 | $36.244 + (-2.6244 \times 1) = 33.6196$ | $-0.1 \times (33.6196 - 10) = -2.36196$ |

**Figure 3**

*Euler method (purple) and analytical (green) solutions to IVP (5), h=1*



## 3.2 Runge-Kutta methods

The Euler method is rarely practically applied, as other Runge-Kutta methods have greater accuracy at the same step-sizes (Fitzpatrick, 2006). This arises from a fundamental asymmetry in the Euler method: in evolving the solution from $x_n \rightarrow x_{n+1}$, the derivative is only evaluated at the beginning of the interval (Press, 2007, p. 907). Other Runge-Kutta methods address this asymmetry by replacing the slope function $f(x_n, y_n)$ in the Euler method formula with a weighted average of slopes calculated across the interval $x_n \leq x \leq x_{n+1}$. The general $m^{\text{th}}$-order Runge-Kutta method is

$$y_{n+1} = y_n + h(w_1 k_1 + w_2 k_2 + \cdots + w_m k_m),$$

where the values $w_i$ are constant weights that sum to 1, the values $k_i$ are evaluations of the slope function $f(x, y)$ for various values of $x \in [x_n, x_{n+1}]$, and the indices of $k$ and $w \in \{1, 2, \ldots, m\}$ (Zill, 2012, p. 368). Thus, the Euler method is simply the first-order Runge-Kutta method; for $m = 1$, $k_1 = f(x_n, y_n)$, and $w_1 = 1$:

$$\begin{aligned} y_{n+1} &= y_n + h(w_1 k_1) \\ &= y_n + f(x_n, y_n)h. \end{aligned}$$

Two other Runge-Kutta methods will be presented: the second-order “improved Euler” method, and a popular fourth-order method known as RK4. The improved Euler method is included to illustrate the derivation of a Runge-Kutta method from the Euler method, while RK4 is considered as a Runge-Kutta method commonly used in practice.

### 3.2.1 The improved Euler method

The second-order “improved” Euler method samples the derivative at the beginning and end of the interval (Thomas & Finney, 1988). It involves:

1. Evaluating the slope at $x_n$ using the initial point $(x_n, y_n)$ and the slope function $f(x, y)$;
2. Approximating the value of $y_{n+1}$ given $y_n$ using the normal Euler method;
3. Evaluating the slope at $x_{n+1}$ using the new point approximation and the slope function; and
4. Taking the average of the slopes at $x_n$ and at $x_{n+1}$, and using this averaged slope to approximate the value of $y_{n+1}$ more accurately than using $f(x_n, y_n)$.

This is represented symbolically as

$$y_{n+1} = y_n + \frac{h}{2}[f(x_n, y_n) + f(x_{n+1}, y_{n+1})],$$

or, substituting the Euler method projection for the unknown value $y_{n+1}$,

$$y_{n+1} = y_n + \frac{h}{2}\left[f(x_n, y_n) + f\big(x_{n+1}, y_n + hf(x_n, y_n)\big)\right].$$

Though this requires two evaluations per step, this extra approximation significantly improves the accuracy of the method, as will be shown (Zill, 2012, p. 368).

### 3.2.2 Fourth-order Runge-Kutta method

Though there exist many other Runge-Kutta methods of various orders, the most significant is the classical fourth-order method, RK4, as it is extensively used in real-world scenarios such as scientific modelling (Workineh et al., 2024). It takes the form of a four-point weighted average:

$$y_{n+1} = y_n + h(w_1k_1 + w_2k_2 + w_3k_3 + w_4k_4).$$

While any choice of parameters would technically constitute *a* fourth-order Runge-Kutta method, the standard Runge-Kutta method takes the following form (Zill, 2012, p. 369):

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

for the values of

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

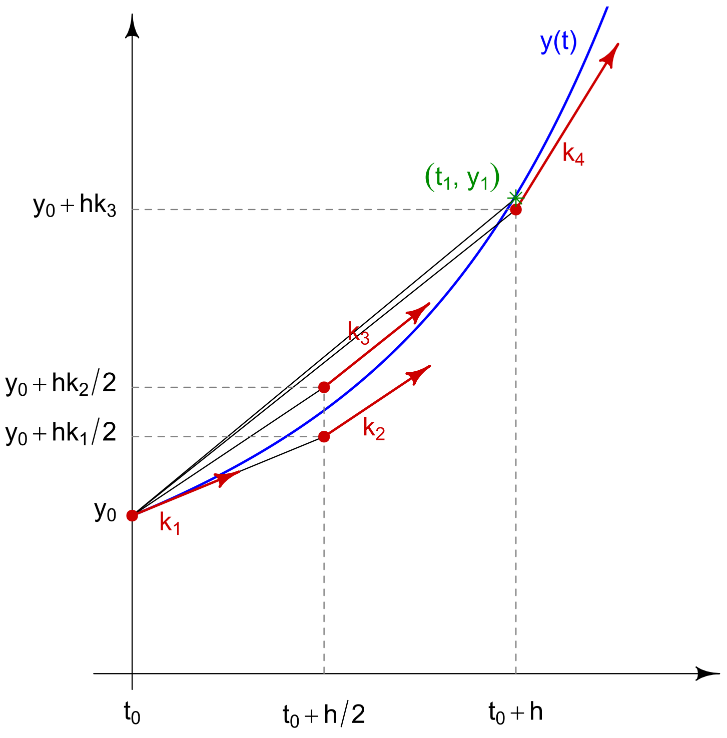
$$k_2 = f(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1)$$

$$k_3 = f(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_{2})$$

$$k_4 = f(x_n + h, y_n + hk_{3})$$

In other words, RK4 involves using a weighted average of four different slopes from within the interval $[x_n, x_{n+1}]$, $k_1$, $k_2$, $k_3$, and $k_4$, which are shown as red vectors in Figure 4. The mathematical rationale for using these particular parameters is beyond the

**Figure 4**
*Graphical depiction of slopes used in RK4 method*



*Note.* *t* used instead of *x* as independent variable. From: "Slopes used by the classical Runge-Kutta method", by HilberTraum, 2017, Wikimedia Commons https://w.wiki/Av9v CC BY-SA 4.0

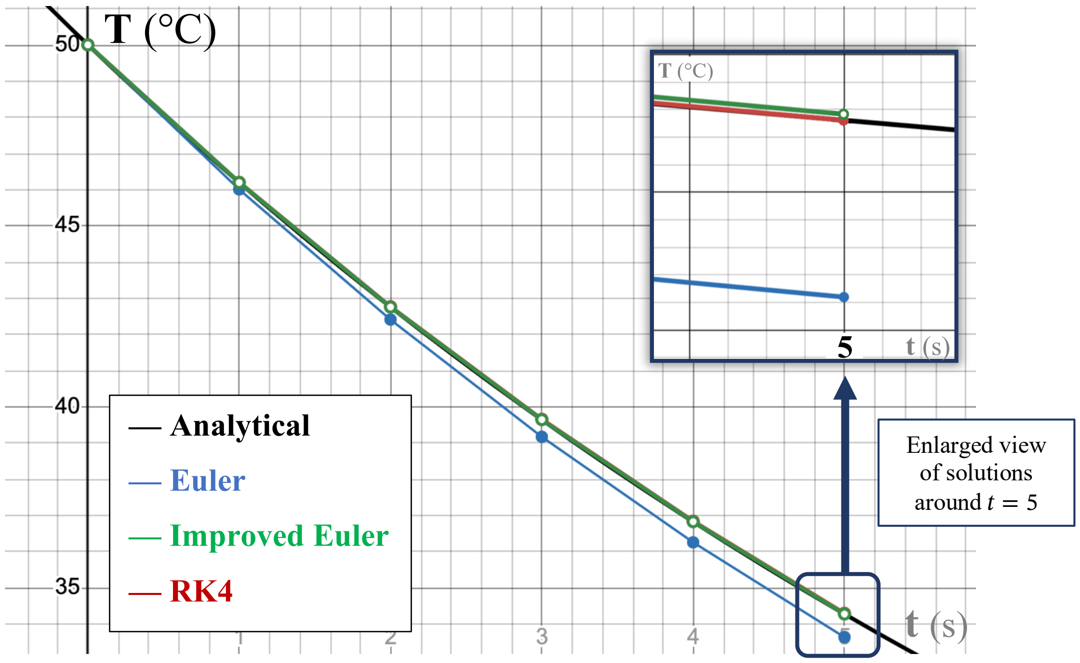
scope of this essay; however, it has been shown that these specific values maximise accuracy (Simmons, 1991, p. 570).

### 3.3 Applying Runge-Kutta methods to an IVP with an analytic solution

Using technology, solutions were computed for the same Newton's law of cooling IVP (5) that the Euler method was tested on above using the improved Euler and RK4 methods, at the same step-size of $h = 1$. Figure 5 shows the analytical (black line), Euler (blue), improved Euler (green) and RK4 (red; mostly hidden behind analytical curve, shown in enlarged view) solutions for $t \in [0,5]$.

**Figure 5**

*Graph of numerical solutions to the IVP (5)*



To compare the accuracy of these solutions[2], the extent to which a numerical solution diverges from the known analytical solution at an endpoint $x_n$ can be quantified by the *absolute error* $e_n$ there:

$$e_n = |y(x_n) - y_n|.$$

[2] In calculating numerical solutions to this ODE, the online tool PlanetCalc.com was used, which calculates the steps for the Euler, improved Euler, and RK4 methods, among others. For the more complicated IVPs presented later, custom programs written for the GNU Octave software package are used, which can be found the specific appendices given for each problem.

The absolute errors for the numerical methods at $y_5$ in the above example were (to four decimal places) 0.6416 for the Euler method; 0.0218 for the improved Euler method; and 0.0000 for RK4 (or, in expanded form, 0.0000109…). Furthermore, both the improved Euler and RK4 solutions had $R^2 = 1$ (to four decimal places), superior to the Euler method's $R^2 = 0.9941$ as given above. Thus, though all Runge-Kutta methods were effective in solving this IVP, for a given step-size RK4 clearly produced more accurate results than the improved Euler method, which in turn was superior to the normal Euler method.

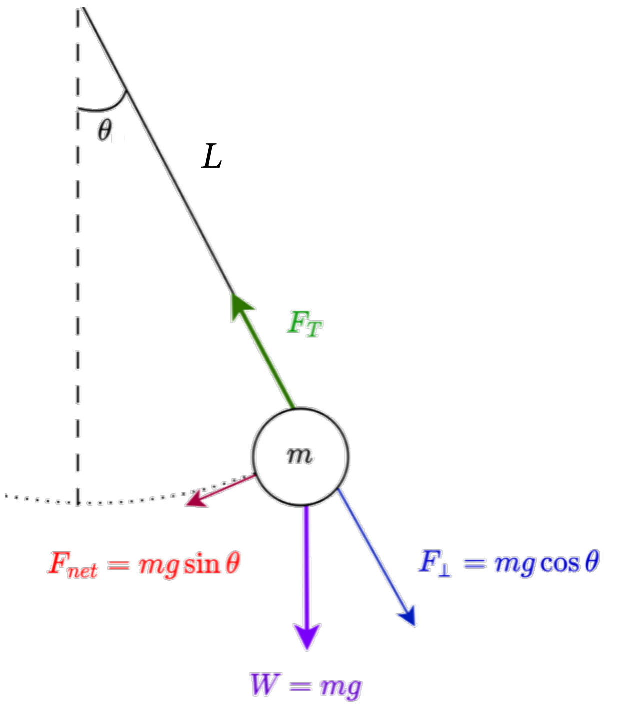
## 4 Modelling and numerically solving the pendulum as an IVP

Though Runge-Kutta methods were shown to be accurate in solving the cooling IVP relative to a known analytical solution, numerical methods are primarily used where an analytical solution is unknown or unattainable. One such unsolvable system is the ODE model of the pendulum.

### 4.1 Derivation of the pendulum ODE

Consider the simple pendulum, a point mass $m$ (the bob) attached to a massless string of length $L$ attached to a pivot, with an angular displacement from the central equilibrium $\theta$, in a gravitational field strength g (Herman, 2018). The forces acting on it are tension, $F_T$, from the string, and weight. Per Figure 6, tension force opposes the component of weight $mg\cos\theta$ in that direction, producing a net force in the opposite direction of the pendulum's displacement of $-mg\sin\theta$.

**Figure 6**
*Diagram of forces acting on simple pendulum*



These expressions can be substituted into Newton's second law of motion:

$$F = ma$$

$$\therefore -mg\sin\theta = ma.$$

Acceleration is the second time-derivative of displacement ($x$), so can be abbreviated as $\ddot{x}$:

$$-mg\sin\theta = m\ddot{x}.$$

The displacement can be derived from the angular displacement of the pendulum $\theta$, and the length of the string $L$, using the arc-length equation,

$$\ell = r\theta$$

$$\therefore x = L\theta,$$

and substituted to find the differential equation for a pendulum's motion (Herman, 2018, pp. 41–42):

$$-mg\sin\theta = m(\ddot{L\theta})$$

$$-mg\sin\theta = mL\ddot{\theta}$$

$$L\ddot{\theta} + g\sin\theta = 0$$

$$\therefore \ddot{\theta} = -\frac{g}{L}\sin\theta\,. \tag{6}$$

## 4.2 Numerical solution

To be solved numerically, the second-order differential equation (6) must be converted into a system of first-order equations. Define a new quantity $v$ as the derivative of angular displacement:

$$v = \dot{\theta}(t). \tag{7}$$

Then, (6) can be rewritten:

$$\dot{v} = \ddot{\theta}(t)$$

$$\therefore \dot{v} = -\frac{g}{L}\sin\theta \tag{8}$$

Together, the system of first-order equations (7) and (8) are equivalent to (6), but can be solved using numerical methods in the same way as a single ODE: at each step, the values for $v$ and $\dot{v}$ are independently calculated based on their prior value (Herman, 2018, pp. 100–104). An IVP in this set of equations thus requires initial conditions for the angular displacement $\theta$ and the angular speed $\dot{\theta}$.

To evaluate the effectiveness of Runge-Kutta methods in modelling pendulum motion, an open-access video and datafile of a pendulum half-oscillation from Titus (2013) were analysed using the Logger Pro software. Time-series data for bob's the position and velocity of the bob (decomposed into $x$ and $y$ components), the angular displacement ($\theta$) in radians, the angular speed ($\dot{\theta}$) in radians-per-second were extracted (full dataset and screenshots in Appendix B). Key quantities are the $t = 0$ values of $\theta$ and $\dot{\theta}$ (the initial conditions, reproduced in Table 2) and the half-oscillation duration, 0.9009s.

**Table 2**
*Initial condition for pendulum motion, taken from Appendix B*

| **Time $t$ (s)** | **Angle $\theta$ (rad)** | **Angular speed $\dot{\theta}$ (rad $s^{-1}$)** |
|---|---|---|
| **0** | -0.759914507 | 0.104307387 |

From this, the initial conditions can be imposed on the system of ODEs (7) and (8):

$$\dot{\theta}(0) = 0.104307387, \qquad \theta(0) = -0.759914507.$$

Using the known parameters of the system ($l = 0.8164$ m and $g = 9.81$ ms$^{-2}$ [Titus, 2013]), the Euler, improved Euler, and RK4 methods were implemented in GNU Octave to solve the IVP, with step-sizes[3] of $h$ = 0.01, 0.05, and 0.10 giving 9 distinct solutions (code in Appendix A) with the final points of the solutions at $t \geq 0.9009$ $s$, the half-oscillation duration (full solutions in Appendix C). As an example, Figure 7 plots the Euler method solution for $h$ = 0.1 against the experimental data.

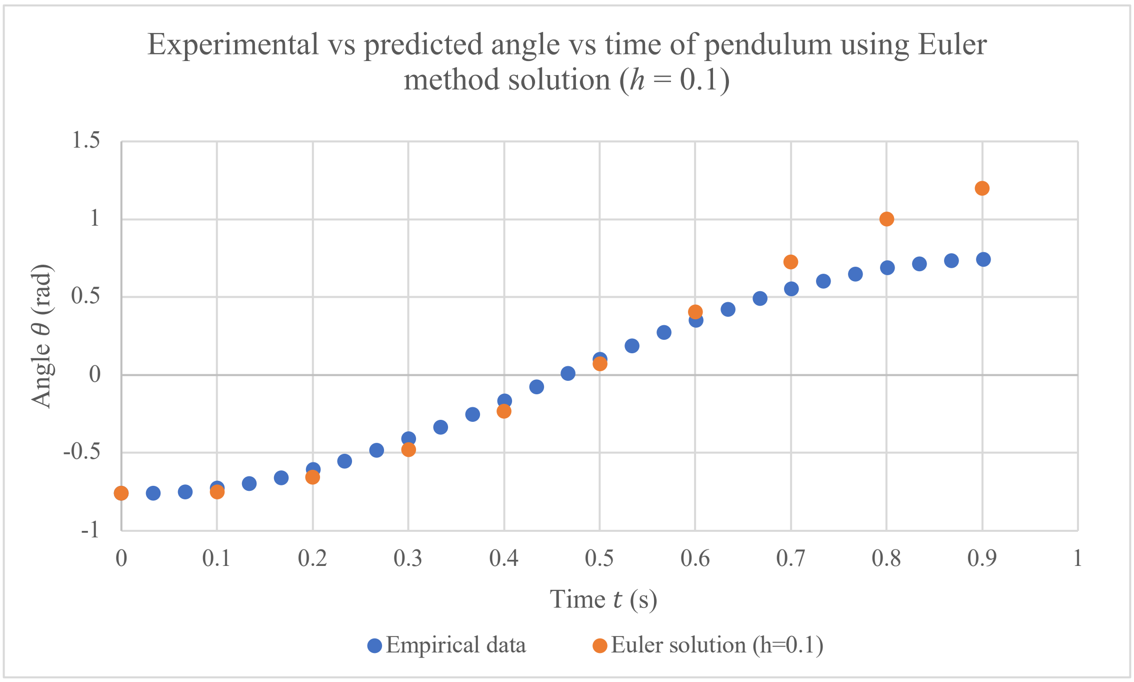
To directly compare the solution points with the empirical data, *interpolation* is needed, because the solution points do not align with the empirical points; the numerical solutions' points are separated by intervals of the step-size $h$ (0.01, 0.05, or 0.10) seconds, while the empirical data points, recorded at 29.97 frames per second, are approximately $1/_{29.97} \approx 0.0334$ s apart. Without common time values, direct comparison is impossible, so interpolation (using a simple function to join the points) is necessary to evaluate the goodness of fit (Gordon & Shampine, 1974). Linear interpolation, a simple

[3] These were arbitrarily chosen to illustrate trends in solution accuracy with escalating step-sizes.

method involving line-segments connecting each point, was implemented in Octave, and the $R^2$ values of each model against the experimental data were calculated (code in Appendix A).

**Figure 7**

*Euler method solution (h=0.1) to nonlinear pendulum against empirical data*



As per Table 3, all Runge-Kutta methods effectively modelled the pendulum's behaviour. Clear trends emerged: RK4 is more accurate than improved Euler, which is more accurate than Euler, and generally, accuracy improves with decreasing step-size. This effect is most pronounced with the Euler method, where accuracy significantly increases as $h$ decreases, while the others show smaller gains.

**Table 3**

*Coefficients of determination of numerical solutions against experimental data*

| *Step size, h* | **$R^2$ against experimental pendulum data over half-swing** | | |
|---|---|---|---|
| | **Euler** | **Improved Euler** | **RK4** |
| 0.01 | 0.997836 | 0.998965 | 0.998975 |
| 0.05 | 0.973244 | 0.998627 | 0.998924 |
| 0.1 | 0.90193 | 0.996913 | 0.998698 |

These remarkably accurate predictions, derived from a model considering only gravity and tension, illustrate the power of ODE models and numerical methods. Arguably, any remaining inaccuracies between these predictions and data likely stem from random errors or differences between the ideal pendulum model and the experimental apparatus (e.g., air resistance or friction) rather than from an insufficiently precise numerical solution.

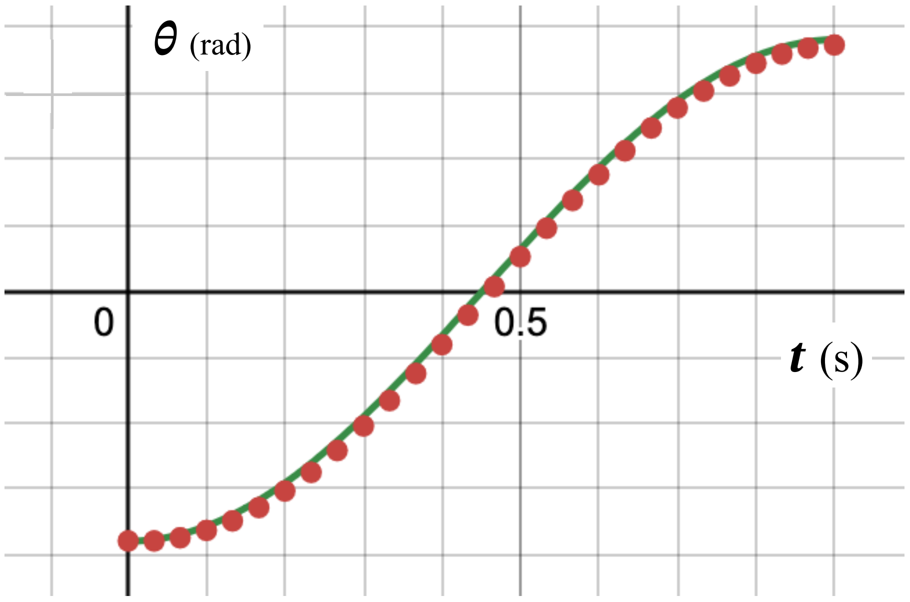
### 4.3 Comparison with simplified analytical solution

The typical equation used in elementary physics to model the motion of the pendulum is

$$\theta(t) = \theta_0 \cos\left(\sqrt{\frac{g}{L}}t\right),$$

given an initial angular displacement $\theta_0$ (Herman, 2018). This results from using the approximation $\sin\theta \approx \theta$ for small $\theta$, which renders the ODE (6) analytically solvable to yield a closed-form solution. This can be used instead of Runge-Kutta methods to model the motion of the pendulum by substituting the initial angle from Table 2 as $\theta_0$ and the same parameters $g$ and $L$ of the system, as shown in Figure 8, for $0 \leq t \leq 0.9009\ s$.

**Figure 8**
*Simplified analytical solution for pendulum (red) vs experimental data (green)*



Though it fits the experimental data well, the simple analytical model has $R^2 = 0.9983$, which is inferior to all RK4 solutions, and to the $h = 0.05$ and $h = 0.01$ improved Euler method solutions. Additionally, while the small-angle approximation and thus the analytical solution become less accurate with increasing $\theta$, the numerical solution should continue to accurately model the behaviour of the system for *any* value of $\theta$ (Herman, 2018, pp. 41–42).

# 5 Considerations

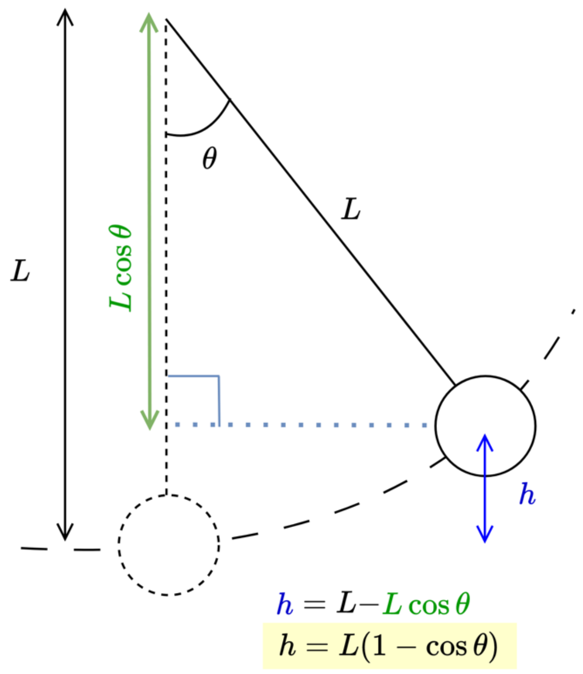
Clearly, Runge-Kutta methods were useful in modelling the pendulum. However, this raises the question of whether their utility in solving this problem generalises to others, and what factors may influence their suitability for specific problem types. Although this essay acknowledges that the specific example of the pendulum may not appear to represent all physical systems, it is argued that the mathematical principles presented above, and the considerations outlined below, are applicable to the vast majority of modelling applications.

## 5.1 Energy conservation

A major concern relevant to physical modelling applications is that Runge-Kutta methods are not energy conserving; they do not necessarily keep constant the *total energy* of a physical system (Arendt, 2023). This is an invariant quantity for closed systems, which are mechanically and thermally isolated from their environment (Ling et al., 2016) — i.e., there is no change in the heat or kinetic energy of the system. For example, the closed system of the pendulum has a constant total energy ($E$), comprised of the sum of the potential ($P$) and kinetic ($K$) energies of the pendulum (Urone et al., 2022):

$$\begin{aligned} E &= P + K \\ &= \frac{1}{2}mv^2 + mgh. \end{aligned}$$

**Figure 9**
*Graphical representation of pendulum height calculation*



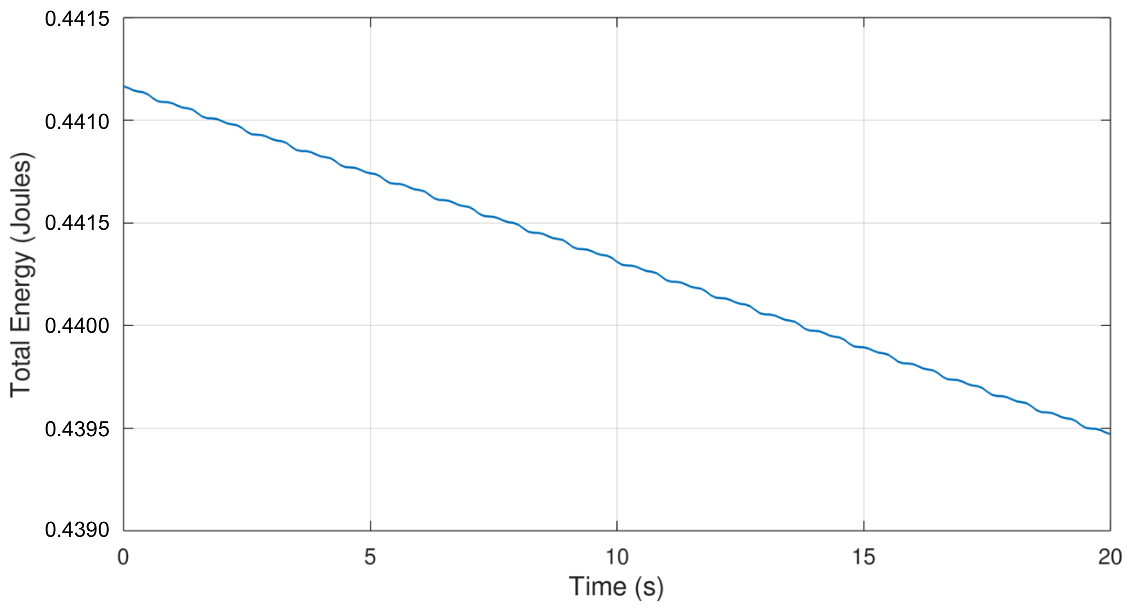
The linear velocity $v$ can be written as the product of the angular velocity $\dot{\theta}$ and the length $L$, while $h$, the height above the resting position of the pendulum, can be written $L(1-\cos\theta)$, as in Figure 9:

$$\begin{aligned} E &= \frac{1}{2}m\left(L\dot{\theta}\right)^2 + mgh \\ &= \frac{1}{2}m\left(L\dot{\theta}\right)^2 + mgL(1-\cos\theta). \end{aligned}$$

This equation was evaluated at each calculation step while solving the pendulum IVP specified in Table 2 above, for $h = 0.1$, and a mass of $m = 0.2$ kg, over an interval of 20 seconds using RK4 (code in Appendix D). As per Figure 10, the total energy is not conserved, but drifts (decreases) somewhat as the system evolves.

**Figure 10**

*Pendulum total energy (J) over time (s) over interval [0,20] seconds with RK4,* $h = 0.1$



This drift is inherent to the Runge-Kutta methods, as they were not designed to conserve energy. Lowering the step-size improves energy conservation due to increased accuracy, but it can significantly increase computation time, especially in long-term closed-system simulations, such as orbital modelling, where small drifts in total energy can derail simulations, and increasing the step-size impractically lengthens the runtime (Arendt, 2023; Montenbruck, 1992). For these cases, it is preferable to use a different type of method which prioritises conserving energy, though these are beyond the scope of this essay (Brorson, 2022).

However, in this example, the energy drift was minimal, decreasing from 0.4412 J to 0.4395 J (~0.4%). Furthermore, halving $h$ to 0.05 gave a final energy of 0.4411 J, reducing the drift to ~0.02% (all code in Appendix D). Thus, except for in special cases, Runge-Kutta methods' lack of energy conservation is arguably insignificant, as it can be compensated for by decreasing the step-size.

## 5.2 Computational complexity

A major advantage of Runge-Kutta methods is their simplicity, as each calculation relies only on the value of the previous point, so they are easier to implement and require less memory than alternatives such as the *linear multistep methods* (LMMs), which use multiple prior points to calculate the next point (Walters et al., 2022). However, while LLMs only evaluate the derivative function once per step, Runge-Kutta methods of order >1 require multiple evaluations, increasing runtime, especially with complex derivative functions (Hull et al., 2006). Although this is insignificant for problems like the pendulum, where the derivative (6) only involves simple multiplication and evaluation of the sine function, trade-offs between the number of derivative evaluations and memory use may warrant consideration of alternative methods for other problems (Hull et al., 2006).

# 6 Conclusion

In this essay, the usefulness of Runge-Kutta methods for solving IVPs associated with modelling physical phenomena was examined. After introducing the concepts of solutions to differential equations and the utility of IVPs using the example of Newton's law of cooling, three Runge-Kutta methods were introduced as tools to solve analytically unsolvable IVPs: the Euler method, the improved Euler method, and RK4. Their usefulness in a real-world modelling scenario was explored by applying them to an ODE describing the motion of a simple pendulum, which lacks an analytical solution. Given the initial conditions and parameters from experimental data of a pendulum's swing, the equation was framed as an IVP, and the Runge-Kutta methods were used to produce solutions, which were then compared against the corresponding empirical data.

It was found that all methods tested were able to effectively model the motion of the pendulum, though the fourth-order Runge-Kutta method (RK4) gave the most accurate results at all step-sizes tested. Furthermore, it was shown that several numerical solutions, including all RK4 solutions, were more accurate than an analytical approximation in modelling the motion of the pendulum. This shows not only that Runge-Kutta solutions make accurate models, but that they also reveal the full complexity of real-world phenomena encoded by ODEs—a level of detail that simpler approximations cannot approach. Although the unsuitability of Runge-Kutta methods in certain edge-case circumstances was considered—they are not energy-conserving, a major limitation for problems such as long-term orbital modelling, and they can be less efficient than other methods when evaluating the derivative function is computationally intense—it was argued that for most physical systems, these are mitigable and/or insignificant.

As a result, it is concluded that Runge-Kutta methods, in particular the fourth-order Runge-Kutta method, are in general significantly useful for solving IVPs in modelling physical systems. This is because they are not only accurate in solving IVPs, but in many cases, they are necessary for this purpose, as many useful ODEs are unsolvable using analytical techniques alone.

## 7 References

Arendt, V. (2023). Numerical methods, energy conservation, and a new method for particle motion in magnetic fields. *Mathematics and Computers in Simulation*, *205*, 142–185. https://doi.org/10.1016/j.matcom.2022.09.015

Braun, M. (1993). *Differential equations and their applications: An introduction to applied mathematics* (4th ed). Springer-Verlag.

Brorson, S. (2022, July 17). *1.7: Symplectic integrators*. Mathematics LibreTexts. https://math.libretexts.org/Bookshelves/Differential_Equations/Numerically_Solving_Ordinary_Differential_Equations_(Brorson)/01%3A_Chapters/1.07%3A_Symplectic_integrators

Butcher, J. C. (2008). *Numerical methods for ordinary differential equations* (2nd ed). Wiley.

Fitzpatrick, R. (2006, March 29). *Runge-Kutta methods*. Computational Physics: An Introductory Course. https://farside.ph.utexas.edu/teaching/329/lectures/node35.html

Gordon, M. K., & Shampine, L. F. (1974). Interpolating numerical solutions of ordinary differential equations. *Proceedings of the 1974 Annual Conference on  - ACM 74*, 46–53. https://doi.org/10.1145/800182.810378

Griffiths, D. F., & Higham, D. J. (2010). *Numerical Methods for Ordinary Differential Equations: Initial Value Problems*. Springer London. https://doi.org/10.1007/978-0-85729-148-6

Herman, R. L. (2018). *A First Course in Differential Equations for Scientists and Engineers*.

Hermann, M., & Saravi, M. (2014). *A first course in ordinary differential equations: Analytical and numerical methods*. Springer.

Hull, T. E., Enright, W. H., Fellen, B. M., & Sedgwick, A. E. (2006). Comparing Numerical Methods for Ordinary Differential Equations. *SIAM Journal on Numerical Analysis*. https://doi.org/10.1137/0709052

Kolk, W. R., & Lerman, R. A. (1992). Analytic Solutions to Nonlinear Differential Equations. In W. R. Kolk & R. A. Lerman (Eds.), *Nonlinear System Dynamics* (pp. 23–60). Springer US. https://doi.org/10.1007/978-1-4684-6494-8_3

Ling, S. J., Moebs, W., & Sanny, J. (2016, October 6). *Ch. 3 Key Terms*. University Physics Volume 2 | OpenStax. https://openstax.org/books/university-physics-volume-2/pages/3-key-terms

Montenbruck, O. (1992). Numerical integration methods for orbital motion. *Celestial Mechanics and Dynamical Astronomy*, *53*(1). https://doi.org/10.1007/BF00049361

Polyanin, A. D. (2017). *Exact Solutions—EqWorld* [Dataset]. https://eqworld.ipmnet.ru/en/solutions.htm

Press, W. H. (2007). *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press.

Simmons, G. F. (1991). *Differential equations, with applications and historical notes* (2nd ed). McGraw-Hill.

Soetaert, K., Cash, J., & Mazzia, F. (2012). *Solving Differential Equations in R*. Springer Science & Business Media.

Thomas, G. B., & Finney, R. L. (1988). *Calculus and analytic geometry* (7th ed). Addison-Wesley Pub. Co.

Titus, A. (2013, May 24). *Video Analysis*. High Point University. https://physics.highpoint.edu/~atitus/videos/

Urone, P. P., Hinrichs, R., Urone, P. P., & Hinrichs, R. (2022, July 13). *16.5 Energy and the Simple Harmonic Oscillator—College Physics 2e | OpenStax*. OpenStax; OpenStax. https://openstax.org/books/college-physics-2e/pages/16-5-energy-and-the-simple-harmonic-oscillator

Walters, S. J., Turner, R. J., & Forbes, L. K. (2022). A COMPARISON OF EXPLICIT RUNGE–KUTTA METHODS. *The ANZIAM Journal*, *64*(3), 227–249. https://doi.org/10.1017/S1446181122000141

Weisstein, E. W. (n.d.). *Elementary Function* [Text]. Wolfram Research, Inc. Retrieved June 20, 2024, from https://mathworld.wolfram.com/

Workineh, Y., Mekonnen, H., & Belew, B. (2024). Numerical methods for solving second-order initial value problems of ordinary differential equations with Euler and Runge-Kutta fourth-order methods. *Frontiers in Applied Mathematics and Statistics*, *10*. https://doi.org/10.3389/fams.2024.1360628

Zill, D. G. (2012). *A First Course in Differential Equations with Modeling Applications*. Cengage Learning.

# Appendix A — Code for numerical methods implementation

Code is written in GNU Octave (https://octave.org) and is MATLAB-compatible.

## Euler method

```
% Initial conditions
theta_0 = -0.75991450687; % initial angle in radians
x_0 = 0.104307386618; % initial angular velocity

% Define the parameters
g = 9.81; % acceleration due to gravity in m/s^2
L = 0.816; % length of the pendulum in meters
t_end = 1; % end time

h = 0.1; % step size
t = 0:h:t_end; % time vector


% Preallocate arrays for efficiency
theta = zeros(1, length(t));
x = zeros(1, length(t));

% Set initial values
theta(1) = theta_0;
x(1) = x_0;

% Define the system of ODEs
f1 = @(theta, x) x; % dtheta/dt = x
f2 = @(theta, x) - (g / L) * sin(theta); % dx/dt = - (g / L) * sin(theta)

% Euler method
for i = 1:length(t) - 1
    theta(i + 1) = theta(i) + h * f1(theta(i), x(i));
    x(i + 1) = x(i) + h * f2(theta(i), x(i));
end
```

```
% Combine results into a single matrix
results = [t', theta', x'];

% Define the output file path
output_file = 'pendulum_simulation_euler.csv';

% Export the results to a CSV file
csvwrite(output_file, results);

disp(['Results using Euler method have been written to ', output_file]);
```

### RK4 method

```
% Define the parameters
g = 9.81; % acceleration due to gravity in m/s^2
L = 0.816; % length of the pendulum in meters
h = 0.01; % step size
t_end = 1; % end time
t = 0:h:t_end; % time vector

theta_0 = -0.75991450687; % initial angle in radians
x_0 = 0.104307386618; % initial angular velocity


% Preallocate arrays for efficiency
theta = zeros(1, length(t));
x = zeros(1, length(t));

% Set initial values
theta(1) = theta_0;
x(1) = x_0;

% Define the system of ODEs
f1 = @(theta, x) x; % dtheta/dt = x
f2 = @(theta, x) - (g / L) * sin(theta); % dx/dt = - (g / L) * sin(theta)
```

```
% RK4 method
for i = 1:length(t) - 1
    k1_theta = f1(theta(i), x(i));
    k1_x = f2(theta(i), x(i));

    k2_theta = f1(theta(i) + 0.5 ∗ h ∗ k1_theta, x(i) + 0.5 ∗ h ∗ k1_x);
    k2_x = f2(theta(i) + 0.5 ∗ h ∗ k1_theta, x(i) + 0.5 ∗ h ∗ k1_x);

    k3_theta = f1(theta(i) + 0.5 ∗ h ∗ k2_theta, x(i) + 0.5 ∗ h ∗ k2_x);
    k3_x = f2(theta(i) + 0.5 ∗ h ∗ k2_theta, x(i) + 0.5 ∗ h ∗ k2_x);

    k4_theta = f1(theta(i) + h ∗ k3_theta, x(i) + h ∗ k3_x);
    k4_x = f2(theta(i) + h ∗ k3_theta, x(i) + h ∗ k3_x);

    theta(i + 1) = theta(i) + (h / 6) ∗ (k1_theta + 2 ∗ k2_theta + 2 ∗ k3_theta + k4_theta);
    x(i + 1) = x(i) + (h / 6) ∗ (k1_x + 2 ∗ k2_x + 2 ∗ k3_x + k4_x);
end

% Combine results into a single matrix
results = [t', theta', x'];

% Define the output file path
output_file = 'pendulum_simulation.csv';

% Export the results to a CSV file
csvwrite(output_file, results);

disp(['Results have been written to ', output_file]);
```

### Improved Euler method

```
% Initial conditions
theta_0 = −0.75991450687; % initial angle in radians
x_0 = 0.104307386618; % initial angular velocity
```

```
% Define the parameters
g = 9.81; % acceleration due to gravity in m/s^2
L = 0.816; % length of the pendulum in meters
t_end = 1; % end time

h = 0.1; % step size

t = 0:h:t_end; % time vector

theta = zeros(1, length(t));
x = zeros(1, length(t));

% Set initial values
theta(1) = theta_0;
x(1) = x_0;

% Define the system of ODEs
f1 = @(theta, x) x; % dtheta/dt = x
f2 = @(theta, x) - (g / L) * sin(theta); % dx/dt = - (g / L) * sin(theta)

% Improved Euler method
for i = 1:length(t) - 1
    k1_theta = f1(theta(i), x(i));
    k1_x = f2(theta(i), x(i));

    % Predictor step
    theta_pred = theta(i) + h * k1_theta;
    x_pred = x(i) + h * k1_x;

    k2_theta = f1(theta_pred, x_pred);
    k2_x = f2(theta_pred, x_pred);

    % Corrector step
    theta(i + 1) = theta(i) + 0.5 * h * (k1_theta + k2_theta);
    x(i + 1) = x(i) + 0.5 * h * (k1_x + k2_x);
end

% Combine results into a single matrix
results = [t', theta', x'];

% Define the output file path
output_file = 'pendulum_simulation_improved_euler.csv';

% Export the results to a CSV file
csvwrite(output_file, results);

disp(['Results have been written to ', output_file]);
```

## Interpolation program

```
% Experimental data
x2 = [0, 0.033366667, 0.066733333, 0.1001, 0.133466667, 0.166833333, 0.2002, 0.233566667,
0.266933333, 0.3003, 0.333666667, 0.367033333, 0.4004, 0.433766667, 0.467133333, 0.5005,
0.533866667,   0.567233333,   0.6006,   0.633966667,   0.667333333,   0.7007,   0.734066667,
0.767433333, 0.8008, 0.834166667, 0.867533333, 0.9009];
y2 = [-0.759914507, -0.759914507, -0.749693941, -0.727929188, -0.698523561, -0.658751517,
-0.608141076,  -0.551968957,  -0.485560058,  -0.411127889,  -0.334938807,  -0.252483116,  -
0.165176763, -0.076091087, 0.010851274, 0.101485748, 0.187491301, 0.271379124, 0.349290463,
0.421597513, 0.491309727, 0.552326642, 0.603210607, 0.648678697, 0.687353289, 0.714451839,
0.73249349, 0.742708611];



methods = {'E', 'IE', 'RK4'};
step_sizes = {'001', '005', '010'};
data = {


[0,0.01,0.02,0.03,0.04,0.05,0.06,0.07,0.08,0.09,0.1,0.11,0.12,0.13,0.14,0.15,0.16,0.17,0.
18,0.19,0.2,0.21,0.22,0.23,0.24,0.25,0.26,0.27,0.28,0.29,0.3,0.31,0.32,0.33,0.34,0.35,0.3
6,0.37,0.38,0.39,0.4,0.41,0.42,0.43,0.44,0.45,0.46,0.47,0.48,0.49,0.5,0.51,0.52,0.53,0.54
,0.55,0.56,0.57,0.58,0.59,0.6,0.61,0.62,0.63,0.64,0.65,0.66,0.67,0.68,0.69,0.7,0.71,0.72,
0.73,0.74,0.75,0.76,0.77,0.78,0.79,0.8,0.81,0.82,0.83,0.84,0.85,0.86,0.87,0.88,0.89,0.9,0
.91],             [-0.75991450687,-0.75887143300382,-0.75700020822577,-0.754301741996829,-
0.75077766810157,-0.746430347685992,-0.741262873322853,-0.735279074058441,-
0.728483521379108,-0.720881536020495,-0.71247919552725,-0.703283342456211,-
0.693301593101744,-0.682542346608087,-0.671014794320433,-0.658728929214139,-
0.645695555230009,-0.631926296333236,-0.617433605104413,-0.602230770663289,-
0.586331925719648,-0.569752052541212,-0.552506987625762,-0.534613424864078,-
0.516088916981831,-0.496951875052445,-0.477221565879272,-0.456918107054249,-
0.43606245951169,-0.414676417409932,-0.392782595190288,-0.370404411682047,-
0.347566071144057,-0.324292541157574,-0.300609527311323,-0.276543444647964,-
0.252121385870932,-0.227371086341757,-0.202320885929964,-0.176999687810107,-
0.151436914333014,-0.125662460130323,-0.0997066426425096,-0.0736001502902034,-
0.0473739885363384,-
```

```
0.0210594241119357,0.00531207229921853,0.0317088846025587,0.0580993107313676,0.0844516226
403456,0.110734126504864,0.136915222773367,0.162963465721464,0.18884762216164,0.214536728
972224,0.240000149122954,0.265207625892123,0.290129334991412,0.314735934338838,0.33899861
1247305,0.362889126825641,0.38637985742014,0.409443832957147,0.432054772080477,0.45418711
4011038,0.475816047089398,0.496917533994751,0.517468333665341,0.537446019975599,0.5568289
97253517,0.575596512748092,0.593728666180476,0.611206416533901,0.628011586256084,0.644126
863063837,0.659535799552721,0.674222810825085,0.688173170357462,0.701373004333441,0.71380
9284670652,0.725469820970718,0.736343251618978,0.746419034256681,0.755687435842397,0.7641
39522511675,0.771767149434788,0.778562950861816,0.78452033053256,0.789633452615986,0.7938
97233330181,0.797307333379382,0.79986015132951]; % E_001

[0,0.05,0.1,0.15,0.2,0.25,0.3,0.35,0.4,0.45,0.5,0.55,0.6,0.65,0.7,0.75,0.8,0.85,0.9,0.95]
,                 [-0.75991450687,-0.7546991375391,-0.728779995411452,-0.682270987871945,-
0.615746453913358,-0.530270428798974,-0.427435511148851,-0.3093997024376,-
0.178904881103019,-
0.0392586612717978,0.105735933305099,0.251910149651494,0.39491237521392,0.53042322690103,
0.654371042709911,0.763114005934287,0.853563593443701,0.923239807325515,0.970265634621479
,0.993320673367264]; % E_005
    [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1], [-0.759914507, -0.749083768, -
0.655937938, -0.480790517, -0.231591642, 0.07253885, 0.404380206, 0.727628454, 1.003686575,
1.199875733, 1.294728612]; % E_010

[0,0.01,0.02,0.03,0.04,0.05,0.06,0.07,0.08,0.09,0.1,0.11,0.12,0.13,0.14,0.15,0.16,0.17,0.
18,0.19,0.2,0.21,0.22,0.23,0.24,0.25,0.26,0.27,0.28,0.29,0.3,0.31,0.32,0.33,0.34,0.35,0.3
6,0.37,0.38,0.39,0.4,0.41,0.42,0.43,0.44,0.45,0.46,0.47,0.48,0.49,0.5,0.51,0.52,0.53,0.54
,0.55,0.56,0.57,0.58,0.59,0.6,0.61,0.62,0.63,0.64,0.65,0.66,0.67,0.68,0.69,0.7,0.71,0.72,
0.73,0.74,0.75,0.76,0.77,0.78,0.79,0.8,0.81,0.82,0.83,0.84,0.85,0.86,0.87,0.88,0.89,0.9,0
.91],           [-0.75991450687,-0.758457357547885,-0.756173147416852,-0.753063871901013,-
0.749132254155525,-0.744381748982911,-0.738816547685311,-0.732441583778471,-
0.725262539478311,-0.717285852856367,-0.708518725546276,-0.69896913086999,-
0.688645822239557,-0.677558341678304,-0.66571702829414,-0.653133026527725,-
0.639818293989414,-0.625785608691477,-0.6110485754762,-0.595621631436232,-
0.579520050121162,-0.562759944323919,-0.545358267242288,-0.527332811814797,-
0.508702208036559,-0.489485918069375,-0.46970422897164,-0.449378242887333,-
0.428529864549565,-0.407181785972852,-0.385357468229204,-0.363081120226339,-
```

```
0.340377674431445,-0.317272759510819,-0.293792669884115,-0.269964332221375,-
0.245815268941351,-0.221373558800214,-0.196667794690355,-0.171727038799047,-
0.146580775305827,-0.121258860825154,-0.0957914728267007,-0.0702090562891258,-
0.0445422688639858,-
0.0188219248441632,0.00692106175445593,0.0326557346786118,0.0583511531642735,0.0839764495
749499,0.109500886623517,0.13489391385443,0.160125223072145,0.185164802414264,0.209982988
784113,0.234550518376879,0.258838575055712,0.282818836359011,0.306463516946942,0.32974540
932378,0.352637921702305,0.375115112906914,0.397151724242764,0.418723208288754,0.43980575
4602079,0.460376312351012,0.480412609920199,0.499893171558763,0.518797331165603,0.5371052
43328315,0.554797891751904,0.571857095230795,0.588265511332575,0.604006637974253,0.619064
813081748,0.633425212530723,0.647073846571934,0.659997554947002,0.672184000901081,0.68362
166429741,0.694299834035344,0.704208599968313,0.713338844511419,0.721682234120204,0.72923
1210812641,0.735978983895799,0.741919522047024,0.747047545887006,0.751358521168901,0.7548
48652693808,0.757514879048588,0.759354868247179]; % IE_001

[0,0.05,0.1,0.15,0.2,0.25,0.3,0.35,0.4,0.45,0.5,0.55,0.6,0.65,0.7,0.75,0.8,0.85,0.9,0.95]
,               [-0.75991450687,-0.744347251140726,-0.708304004290425,-0.652592794942084,-
0.578514358967762,-0.487887875159485,-0.38306387078201,-0.266908286501772,-
0.142744288592116,-
0.0142457052375095,0.114713142919906,0.240235023075758,0.358580753804627,0.46632848249696
,0.560492424805145,0.638593134430723,0.698682874217911,0.7393381716077,0.759635484045288,
0.759125304729966]; % IE_005
    [0,   0.1,   0.2,   0.3,   0.4,   0.5,   0.6,   0.7,   0.8,   0.9,   1],   [-0.75991450687,-
0.708076222614703,-0.576192991229164,-0.376864291814328,-
0.131475334565704,0.130990355144067,0.378223169564728,0.580624247482955,0.715971413488593
,0.77093808281402,0.740466056459093]; % IE_010

[0,0.01,0.02,0.03,0.04,0.05,0.06,0.07,0.08,0.09,0.1,0.11,0.12,0.13,0.14,0.15,0.16,0.17,0.
18,0.19,0.2,0.21,0.22,0.23,0.24,0.25,0.26,0.27,0.28,0.29,0.3,0.31,0.32,0.33,0.34,0.35,0.3
6,0.37,0.38,0.39,0.4,0.41,0.42,0.43,0.44,0.45,0.46,0.47,0.48,0.49,0.5,0.51,0.52,0.53,0.54
,0.55,0.56,0.57,0.58,0.59,0.6,0.61,0.62,0.63,0.64,0.65,0.66,0.67,0.68,0.69,0.7,0.71,0.72,
0.73,0.74,0.75,0.76,0.77,0.78,0.79,0.8,0.81,0.82,0.83,0.84,0.85,0.86,0.87,0.88,0.89,0.9,0
.91],           [-0.75991450687,-0.758457539176344,-0.756173751546375,-0.753065139390519,-
0.749134425698587,-0.744385064943273,-0.738821247916104,-0.732447907421778,-
0.725270724741873,-0.71729613676438,-0.708531343661467,-0.69898431698436,-
```

```
0.688663808031497,-0.677579356334056,-0.665741298091965,-0.653160774383486,-
0.639849738962732,-0.625820965452067,-0.611088053730475,-0.595665435314814,-
0.57956837752848,-0.562812986251647,-0.545416207049004,-0.527395824474834,-
0.508770459361657,-0.489559563907352,-0.469783414386927,-0.44946310132877,-
0.428620517011457,-0.407278340155742,-0.385460017707324,-0.363189743629035,-
0.340492434646228,-0.31739370291588,-0.293919825618267,-0.270097711499414,-
0.245954864422696,-0.2215193440185,-0.196819723551288,-0.171885045153371,-
0.146744772603698,-0.121428741857491,-0.095967109558316,-0.0703902997875391,-
0.0447289493268617,-
0.0190138517272764,0.00672409950789963,0.0324539683075787,0.0581448336598275,0.0837658471
530048,0.109286290103994,0.134675629951427,0.159903575600677,0.184940131420003,0.20975564
9603337,0.234320880634479,0.258607021609659,0.282585762200075,0.306229328062742,0.3295105
21536355,0.35240275948843,0.374880108210229,0.39691731528655,0.418489838397834,0.43957387
1041831,0.460146365190952,0.480185050928952,0.499668453136526,0.518575905319492,0.5368875
60695189,0.554584400672455,0.571648240877916,0.588061734896178,0.603808375903967,0.618872
496388137,0.633239266144958,0.646894688763169,0.659825596796079,0.672019645828601,0.68346
5307643687,0.6941518626893,0.704069392041953,0.713208769056193,0.721561650881253,0.729120
470016707,0.735878426068384,0.741829477854263,0.746968335997627,0.751290456131606,0.75479
2032825437,0.757469994328438,0.759321998212951]; % RK4_001

[0,0.05,0.1,0.15,0.2,0.25,0.3,0.35,0.4,0.45,0.5,0.55,0.6,0.65,0.7,0.75,0.8,0.85,0.9,0.95]
,               [-0.75991450687,-0.744385099385328,-0.708531601767754,-0.653161456231264,-
0.579569692953115,-0.489561724641319,-0.385463220278079,-0.270102110620142,-
0.14675044876765,-
0.0190207810173368,0.109278254015811,0.234312004843371,0.352393409664099,0.46013696917893
2,0.554575408264381,0.633231114576743,0.694144953198805,0.735873112060331,0.7574665784839
78,0.758447690625265]; % RK4_005
    [0,   0.1,   0.2,   0.3,   0.4,   0.5,   0.6,   0.7,   0.8,   0.9,   1],   [-0.75991450687,-
0.708533218581146,-0.579585198767511,-0.385506293207246,-
0.146832315472387,0.109156149268454,0.352244076280842,0.554422595941728,0.694014735460751
,0.757381566081535,0.738772908539928]; % RK4_010
};

% Loop through each method and step size, interpolate, and calculate R²
for i = 1:length(data)
```

```
    method = methods{ceil(i/3)};
    step_size = step_sizes{mod(i-1, 3) + 1};
    x1 = data{i, 1};
    y1 = data{i, 2};

    % Interpolate the numerical solution at the experimental x-coordinates
    y1_interpolated = interp1(x1, y1, x2, 'linear');

    % Calculate the R² value
    SStot = sum((y2 - mean(y2)).^2);
    SSres = sum((y2 - y1_interpolated).^2);
    R2 = 1 - (SSres / SStot);

    % Print the R² value
    fprintf('R² value for %s_%s: %f\n', method, step_size, R2);
end


hold on;

% Plot the experimental points
plot(x2, y2, 'ro', 'DisplayName', 'Experimental Data');


colors = {'bo-', 'go-', 'co-', 'mo-', 'yo-', 'ko-', 'bd-', 'gd-', 'cd-'};
for i = 1:length(data)
    x1 = data{i, 1};
    y1 = data{i, 2};
    plot(x1,   y1,   colors{i},   'DisplayName',   sprintf('%s_%s',   methods{ceil(i/3)},
step_sizes{mod(i-1, 3) + 1}));
end

% Annotate the plot
xlabel('x');
ylabel('y');
```

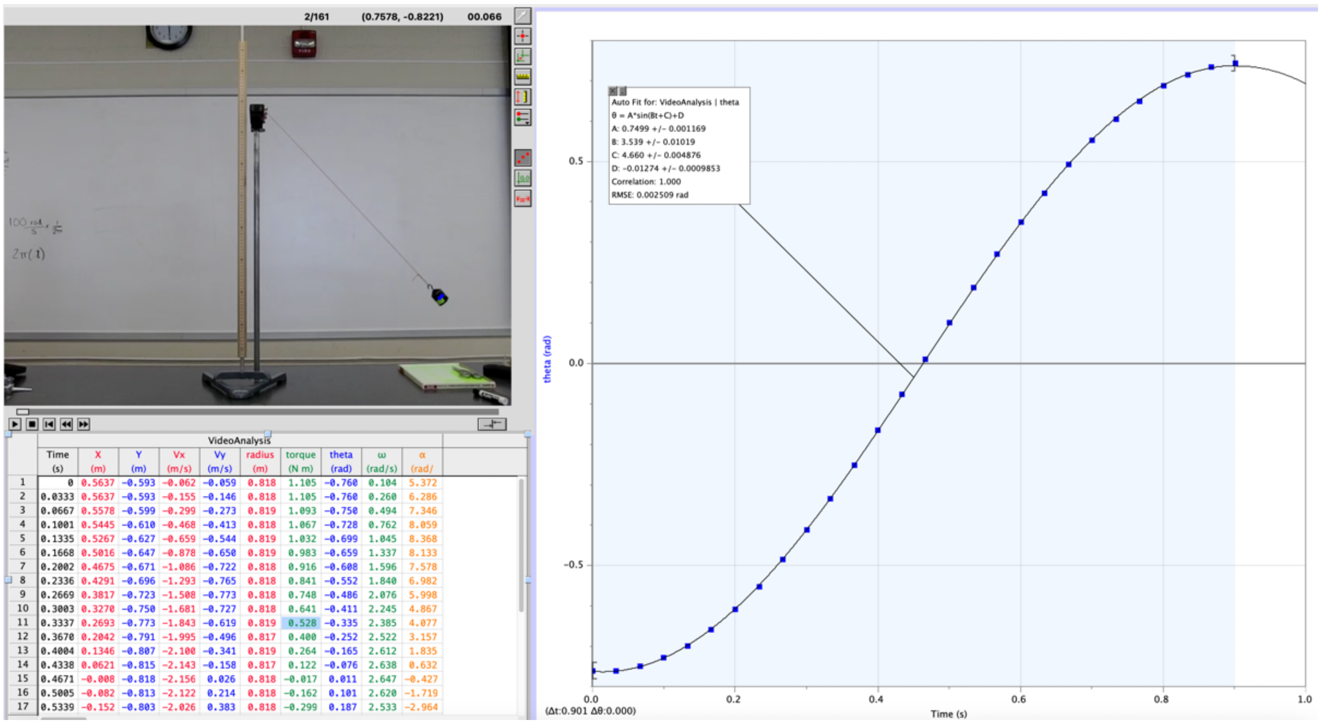
```
title('Interpolation of Numerical Solution Points for Various Methods and Step Sizes');
legend('show');
grid on;

hold off;
```

## Appendix B — Pendulum dataset

**Table B: Time-series data of pendulum angle $\theta$, angular velocity $\dot{\theta}$**

| **Time *t* (s)** | **Angle $\theta$ (rad)** | **Angular speed $\dot{\theta}$ (rad $s^{-1}$)** |
|---|---|---|
| 0 | -0.759914507 | 0.104307387 |
| 0.033366667 | -0.759914507 | 0.259750905 |
| 0.066733333 | -0.749693941 | 0.494036213 |
| 0.1001 | -0.727929188 | 0.762400785 |
| 0.133466667 | -0.698523561 | 1.044957692 |
| 0.166833333 | -0.658751517 | 1.336544239 |
| 0.2002 | -0.608141076 | 1.595683355 |
| 0.233566667 | -0.551968957 | 1.839564433 |
| 0.266933333 | -0.485560058 | 2.075794886 |
| 0.3003 | -0.411127889 | 2.245265646 |
| 0.333666667 | -0.334938807 | 2.384973952 |
| 0.367033333 | -0.252483116 | 2.521961483 |
| 0.4004 | -0.165176763 | 2.611981477 |
| 0.433766667 | -0.076091087 | 2.637853013 |
| 0.467133333 | 0.010851274 | 2.647400884 |
| 0.5005 | 0.101485748 | 2.619578837 |
| 0.533866667 | 0.187491301 | 2.532506289 |
| 0.567233333 | 0.271379124 | 2.411743431 |
| 0.6006 | 0.349290463 | 2.259625105 |
| 0.633966667 | 0.421597513 | 2.111834198 |
| 0.667333333 | 0.491309727 | 1.927757379 |
| 0.7007 | 0.552326642 | 1.686991587 |

| | | |
|---|---|---|
| 0.734066667 | 0.603210607 | 1.455339247 |
| 0.767433333 | 0.648678697 | 1.236133444 |
| 0.8008 | 0.687353289 | 0.974183233 |
| 0.834166667 | 0.714451839 | 0.712411882 |
| 0.867533333 | 0.73249349 | 0.513467451 |
| 0.9009 | 0.742708611 | 0.386383158 |



Auto Fit for: VideoAnalysis | theta
θ = A*sin(Bt+C)+D
A: 0.7499 +/- 0.001169
B: 3.539 +/- 0.01019
C: 4.660 +/- 0.004876
D: -0.01274 +/- 0.0009853
Correlation: 1.000
RMSE: 0.002509 rad

VideoAnalysis

| | Time (s) | X (m) | Y (m) | Vx (m/s) | Vy (m/s) | radius (m) | torque (N m) | theta (rad) | ω (rad/s) | α (rad/ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0.5637 | -0.593 | -0.062 | -0.059 | 0.818 | 1.105 | -0.760 | 0.104 | 5.372 |
| 2 | 0.0333 | 0.5637 | -0.593 | -0.155 | -0.146 | 0.818 | 1.105 | -0.760 | 0.260 | 6.286 |
| 3 | 0.0667 | 0.5578 | -0.599 | -0.299 | -0.273 | 0.819 | 1.093 | -0.750 | 0.494 | 7.346 |
| 4 | 0.1001 | 0.5445 | -0.610 | -0.468 | -0.413 | 0.818 | 1.067 | -0.728 | 0.762 | 8.059 |
| 5 | 0.1335 | 0.5267 | -0.627 | -0.659 | -0.544 | 0.819 | 1.032 | -0.699 | 1.045 | 8.368 |
| 6 | 0.1668 | 0.5016 | -0.647 | -0.878 | -0.650 | 0.819 | 0.983 | -0.659 | 1.337 | 8.133 |
| 7 | 0.2002 | 0.4675 | -0.671 | -1.086 | -0.722 | 0.818 | 0.916 | -0.608 | 1.596 | 7.578 |
| 8 | 0.2336 | 0.4291 | -0.696 | -1.293 | -0.765 | 0.818 | 0.841 | -0.552 | 1.840 | 6.982 |
| 9 | 0.2669 | 0.3817 | -0.723 | -1.508 | -0.773 | 0.818 | 0.748 | -0.486 | 2.076 | 5.998 |
| 10 | 0.3003 | 0.3270 | -0.750 | -1.681 | -0.727 | 0.818 | 0.641 | -0.411 | 2.245 | 4.867 |
| 11 | 0.3337 | 0.2693 | -0.773 | -1.843 | -0.619 | 0.819 | 0.528 | -0.335 | 2.385 | 4.077 |
| 12 | 0.3670 | 0.2042 | -0.791 | -1.995 | -0.496 | 0.817 | 0.400 | -0.252 | 2.522 | 3.157 |
| 13 | 0.4004 | 0.1346 | -0.807 | -2.100 | -0.341 | 0.819 | 0.264 | -0.165 | 2.612 | 1.835 |
| 14 | 0.4338 | 0.0621 | -0.815 | -2.143 | -0.158 | 0.817 | 0.122 | -0.076 | 2.638 | 0.632 |
| 15 | 0.4671 | -0.008 | -0.818 | -2.156 | 0.026 | 0.818 | -0.017 | 0.011 | 2.647 | -0.427 |
| 16 | 0.5005 | -0.082 | -0.813 | -2.122 | 0.214 | 0.818 | -0.162 | 0.101 | 2.620 | -1.719 |
| 17 | 0.5339 | -0.152 | -0.803 | -2.026 | 0.383 | 0.818 | -0.299 | 0.187 | 2.533 | -2.964 |

*Figure B: Screenshot of Logger Pro user interface*

# Appendix C — Numerical solutions to the nonlinear pendulum

**Table C1: Numerical solutions, h=0.01**

| **RK4** | | **Euler** | | **Improved Euler** | |
|---|---|---|---|---|---|
| **Time *t* (s)** | **Angle *θ* (rad)** | **Time *t* (s)** | **Angle *θ* (rad)** | **Time *t* (s)** | **Angle *θ* (rad)** |
| *0* | -0.75991451 | 0 | -0.7599145 | 0 | -0.7599145 |
| 0.01 | -0.75845754 | 0.01 | -0.7588714 | 0.01 | -0.7584574 |
| 0.02 | -0.75617375 | 0.02 | -0.7570002 | 0.02 | -0.7561731 |
| 0.03 | -0.75306514 | 0.03 | -0.7543017 | 0.03 | -0.7530639 |
| 0.04 | -0.74913443 | 0.04 | -0.7507777 | 0.04 | -0.7491323 |
| 0.05 | -0.74438506 | 0.05 | -0.7464303 | 0.05 | -0.7443817 |
| 0.06 | -0.73882125 | 0.06 | -0.7412629 | 0.06 | -0.7388165 |
| 0.07 | -0.73244791 | 0.07 | -0.7352791 | 0.07 | -0.7324416 |
| 0.08 | -0.72527072 | 0.08 | -0.7284835 | 0.08 | -0.7252625 |
| 0.09 | -0.71729614 | 0.09 | -0.7208815 | 0.09 | -0.7172859 |
| 0.1 | -0.70853134 | 0.1 | -0.7124792 | 0.1 | -0.7085187 |
| 0.11 | -0.69898432 | 0.11 | -0.7032833 | 0.11 | -0.6989691 |
| 0.12 | -0.68866381 | 0.12 | -0.6933016 | 0.12 | -0.6886458 |
| 0.13 | -0.67757936 | 0.13 | -0.6825423 | 0.13 | -0.6775583 |
| 0.14 | -0.66574130 | 0.14 | -0.6710148 | 0.14 | -0.665717 |
| 0.15 | -0.65316077 | 0.15 | -0.6587289 | 0.15 | -0.653133 |
| 0.16 | -0.63984974 | 0.16 | -0.6456956 | 0.16 | -0.6398183 |
| 0.17 | -0.62582097 | 0.17 | -0.6319263 | 0.17 | -0.6257856 |
| 0.18 | -0.61108805 | 0.18 | -0.6174336 | 0.18 | -0.6110486 |
| 0.19 | -0.59566544 | 0.19 | -0.6022308 | 0.19 | -0.5956216 |
| 0.2 | -0.57956838 | 0.2 | -0.5863319 | 0.2 | -0.5795201 |

| | | | | | |
|---|---|---|---|---|---|
| 0.21 | -0.56281299 | 0.21 | -0.5697521 | 0.21 | -0.5627599 |
| 0.22 | -0.54541621 | 0.22 | -0.552507 | 0.22 | -0.5453583 |
| 0.23 | -0.52739582 | 0.23 | -0.5346134 | 0.23 | -0.5273328 |
| 0.24 | -0.50877046 | 0.24 | -0.5160889 | 0.24 | -0.5087022 |
| 0.25 | -0.48955956 | 0.25 | -0.4969519 | 0.25 | -0.4894859 |
| 0.26 | -0.46978341 | 0.26 | -0.4772216 | 0.26 | -0.4697042 |
| 0.27 | -0.44946310 | 0.27 | -0.4569181 | 0.27 | -0.4493782 |
| 0.28 | -0.42862052 | 0.28 | -0.4360625 | 0.28 | -0.4285299 |
| 0.29 | -0.40727834 | 0.29 | -0.4146764 | 0.29 | -0.4071818 |
| 0.3 | -0.38546002 | 0.3 | -0.3927826 | 0.3 | -0.3853575 |
| 0.31 | -0.36318974 | 0.31 | -0.3704044 | 0.31 | -0.3630811 |
| 0.32 | -0.34049243 | 0.32 | -0.3475661 | 0.32 | -0.3403777 |
| 0.33 | -0.31739370 | 0.33 | -0.3242925 | 0.33 | -0.3172728 |
| 0.34 | -0.29391983 | 0.34 | -0.3006095 | 0.34 | -0.2937927 |
| 0.35 | -0.27009771 | 0.35 | -0.2765434 | 0.35 | -0.2699643 |
| 0.36 | -0.24595486 | 0.36 | -0.2521214 | 0.36 | -0.2458153 |
| 0.37 | -0.22151934 | 0.37 | -0.2273711 | 0.37 | -0.2213736 |
| 0.38 | -0.19681972 | 0.38 | -0.2023209 | 0.38 | -0.1966678 |
| 0.39 | -0.17188505 | 0.39 | -0.1769997 | 0.39 | -0.171727 |
| 0.4 | -0.14674477 | 0.4 | -0.1514369 | 0.4 | -0.1465808 |
| 0.41 | -0.12142874 | 0.41 | -0.1256625 | 0.41 | -0.1212589 |
| 0.42 | -0.09596711 | 0.42 | -0.0997066 | 0.42 | -0.0957915 |
| 0.43 | -0.07039030 | 0.43 | -0.0736002 | 0.43 | -0.0702091 |
| 0.44 | -0.04472895 | 0.44 | -0.047374 | 0.44 | -0.0445423 |
| 0.45 | -0.01901385 | 0.45 | -0.0210594 | 0.45 | -0.0188219 |

| | | | | | |
|---|---|---|---|---|---|
| 0.46 | 0.00672410 | 0.46 | 0.00531207 | 0.46 | 0.00692106 |
| 0.47 | 0.03245397 | 0.47 | 0.03170888 | 0.47 | 0.03265573 |
| 0.48 | 0.05814483 | 0.48 | 0.05809931 | 0.48 | 0.05835115 |
| 0.49 | 0.08376585 | 0.49 | 0.08445162 | 0.49 | 0.08397645 |
| 0.5 | 0.10928629 | 0.5 | 0.11073413 | 0.5 | 0.10950089 |
| 0.51 | 0.13467563 | 0.51 | 0.13691522 | 0.51 | 0.13489391 |
| 0.52 | 0.15990358 | 0.52 | 0.16296347 | 0.52 | 0.16012522 |
| 0.53 | 0.18494013 | 0.53 | 0.18884762 | 0.53 | 0.1851648 |
| 0.54 | 0.20975565 | 0.54 | 0.21453673 | 0.54 | 0.20998299 |
| 0.55 | 0.23432088 | 0.55 | 0.24000015 | 0.55 | 0.23455052 |
| 0.56 | 0.25860702 | 0.56 | 0.26520763 | 0.56 | 0.25883858 |
| 0.57 | 0.28258576 | 0.57 | 0.29012933 | 0.57 | 0.28281884 |
| 0.58 | 0.30622933 | 0.58 | 0.31473593 | 0.58 | 0.30646352 |
| 0.59 | 0.32951052 | 0.59 | 0.33899861 | 0.59 | 0.32974541 |
| 0.6 | 0.35240276 | 0.6 | 0.36288913 | 0.6 | 0.35263792 |
| 0.61 | 0.37488011 | 0.61 | 0.38637986 | 0.61 | 0.37511511 |
| 0.62 | 0.39691732 | 0.62 | 0.40944383 | 0.62 | 0.39715172 |
| 0.63 | 0.41848984 | 0.63 | 0.43205477 | 0.63 | 0.41872321 |
| 0.64 | 0.43957387 | 0.64 | 0.45418711 | 0.64 | 0.43980575 |
| 0.65 | 0.46014637 | 0.65 | 0.47581605 | 0.65 | 0.46037631 |
| 0.66 | 0.48018505 | 0.66 | 0.49691753 | 0.66 | 0.48041261 |
| 0.67 | 0.49966845 | 0.67 | 0.51746833 | 0.67 | 0.49989317 |
| 0.68 | 0.51857591 | 0.68 | 0.53744602 | 0.68 | 0.51879733 |
| 0.69 | 0.53688756 | 0.69 | 0.556829 | 0.69 | 0.53710524 |
| 0.7 | 0.55458440 | 0.7 | 0.57559651 | 0.7 | 0.55479789 |

| | | | | | |
|---|---|---|---|---|---|
| 0.71 | 0.57164824 | 0.71 | 0.59372867 | 0.71 | 0.5718571 |
| 0.72 | 0.58806173 | 0.72 | 0.61120642 | 0.72 | 0.58826551 |
| 0.73 | 0.60380838 | 0.73 | 0.62801159 | 0.73 | 0.60400664 |
| 0.74 | 0.61887250 | 0.74 | 0.64412686 | 0.74 | 0.61906481 |
| 0.75 | 0.63323927 | 0.75 | 0.6595358 | 0.75 | 0.63342521 |
| 0.76 | 0.64689469 | 0.76 | 0.67422281 | 0.76 | 0.64707385 |
| 0.77 | 0.65982560 | 0.77 | 0.68817317 | 0.77 | 0.65999755 |
| 0.78 | 0.67201965 | 0.78 | 0.701373 | 0.78 | 0.672184 |
| 0.79 | 0.68346531 | 0.79 | 0.71380928 | 0.79 | 0.68362166 |
| 0.8 | 0.69415186 | 0.8 | 0.72546982 | 0.8 | 0.69429983 |
| 0.81 | 0.70406939 | 0.81 | 0.73634325 | 0.81 | 0.7042086 |
| 0.82 | 0.71320877 | 0.82 | 0.74641903 | 0.82 | 0.71333884 |
| 0.83 | 0.72156165 | 0.83 | 0.75568744 | 0.83 | 0.72168223 |
| 0.84 | 0.72912047 | 0.84 | 0.76413952 | 0.84 | 0.72923121 |
| 0.85 | 0.73587843 | 0.85 | 0.77176715 | 0.85 | 0.73597898 |
| 0.86 | 0.74182948 | 0.86 | 0.77856295 | 0.86 | 0.74191952 |
| 0.87 | 0.74696834 | 0.87 | 0.78452033 | 0.87 | 0.74704755 |
| 0.88 | 0.75129046 | 0.88 | 0.78963345 | 0.88 | 0.75135852 |
| 0.89 | 0.75479203 | 0.89 | 0.79389723 | 0.89 | 0.75484865 |
| 0.9 | 0.75746999 | 0.9 | 0.79730733 | 0.9 | 0.75751488 |
| 0.91 | 0.75932200 | 0.91 | 0.79986015 | 0.91 | 0.75935487 |

**Table C2: Numerical solutions, h=0.05**

| **RK4** | | **Euler** | | **Improved Euler** | |
|---|---|---|---|---|---|
| **Time $t$ (s)** | **Angle $\theta$ (rad)** | **Time $t$ (s)** | **Angle $\theta$ (rad)** | **Time $t$ (s)** | **Angle $\theta$ (rad)** |
| *0* | -0.7599145 | 0 | -0.7599145 | 0 | -0.7599145 |
| 0.05 | -0.7443851 | 0.05 | -0.7546991 | 0.05 | -0.7443473 |
| 0.1 | -0.7085316 | 0.1 | -0.72878 | 0.1 | -0.708304 |
| 0.15 | -0.6531615 | 0.15 | -0.682271 | 0.15 | -0.6525928 |
| 0.2 | -0.5795697 | 0.2 | -0.6157465 | 0.2 | -0.5785144 |
| 0.25 | -0.4895617 | 0.25 | -0.5302704 | 0.25 | -0.4878879 |
| 0.3 | -0.3854632 | 0.3 | -0.4274355 | 0.3 | -0.3830639 |
| 0.35 | -0.2701021 | 0.35 | -0.3093997 | 0.35 | -0.2669083 |
| 0.4 | -0.1467504 | 0.4 | -0.1789049 | 0.4 | -0.1427443 |
| 0.45 | -0.0190208 | 0.45 | -0.0392587 | 0.45 | -0.0142457 |
| 0.5 | 0.10927825 | 0.5 | 0.10573593 | 0.5 | 0.11471314 |
| 0.55 | 0.234312 | 0.55 | 0.25191015 | 0.55 | 0.24023502 |
| 0.6 | 0.35239341 | 0.6 | 0.39491238 | 0.6 | 0.35858075 |
| 0.65 | 0.46013697 | 0.65 | 0.53042323 | 0.65 | 0.46632848 |
| 0.7 | 0.55457541 | 0.7 | 0.65437104 | 0.7 | 0.56049242 |
| 0.75 | 0.63323111 | 0.75 | 0.76311401 | 0.75 | 0.63859313 |
| 0.8 | 0.69414495 | 0.8 | 0.85356359 | 0.8 | 0.69868287 |
| 0.85 | 0.73587311 | 0.85 | 0.92323981 | 0.85 | 0.73933817 |
| 0.9 | 0.75746658 | 0.9 | 0.97026563 | 0.9 | 0.75963548 |
| 0.95 | 0.75844769 | 0.95 | 0.99332067 | 0.95 | 0.7591253 |

**Table C3: Numerical solutions, h=0.1**

| **RK4** | | **Euler** | | **Improved Euler** | |
|---|---|---|---|---|---|
| **Time *t* (s)** | **Angle $\theta$ (rad)** | **Time *t* (s)** | **Angle $\theta$ (rad)** | **Time *t* (s)** | **Angle $\theta$ (rad)** |
| *0* | -0.7599145 | 0 | -0.7599145 | 0 | -0.7599145 |
| 0.1 | -0.7085332 | 0.1 | -0.7494838 | 0.1 | -0.7080762 |
| 0.2 | -0.5795852 | 0.2 | -0.6562379 | 0.2 | -0.576193 |
| 0.3 | -0.3855063 | 0.3 | -0.4810905 | 0.3 | -0.3768643 |
| 0.4 | -0.1468323 | 0.4 | -0.2325916 | 0.4 | -0.1314753 |
| 0.5 | 0.10915615 | 0.5 | 0.07153885 | 0.5 | 0.13099036 |
| 0.6 | 0.35224408 | 0.6 | 0.40338021 | 0.6 | 0.37822317 |
| 0.7 | 0.5544226 | 0.7 | 0.72662845 | 0.7 | 0.58062425 |
| 0.8 | 0.69401474 | 0.8 | 1.00268658 | 0.8 | 0.71597141 |
| 0.9 | 0.75738157 | 0.9 | 1.19887573 | 0.9 | 0.77093808 |
| 1 | 0.73877291 | 1 | 1.29372861 | 1 | 0.74046606 |

## Appendix D — Energy conservation calculation code for pendulum

```
% Define the parameters
g = 9.81; % acceleration due to gravity in m/s^2
L = 0.816; % length of the pendulum in meters
t_end = 20; % end time
h = 0.1; % step size
t = 0:h:t_end; % time vector
m = 0.2

% Initial conditions
theta_0 = -0.75991450687; % initial angle in radians
x_0 = 0.104307386618; % initial angular velocity

theta = zeros(1, length(t));
x = zeros(1, length(t));
total_energy = zeros(1, length(t));

theta(1) = theta_0;
x(1) = x_0;

% Calculate initial total energy
kinetic_energy = 0.5 * m * (L * x(1))^2; % Kinetic energy = 0.5 * m * v^2 (mass m is
considered 1 for simplicity)
potential_energy = m * g * L * (1 - cos(theta(1))); % Potential energy = m * g * h (mass m
is considered 1 for simplicity)
total_energy(1) = kinetic_energy + potential_energy;

% Define the system of ODEs
f1 = @(theta, x) x; % dtheta/dt = x
f2 = @(theta, x) - (g / L) * sin(theta); % dx/dt = - (g / L) * sin(theta)

% RK4 method
for i = 1:length(t) - 1
    k1_theta = f1(theta(i), x(i));
    k1_x = f2(theta(i), x(i));

    k2_theta = f1(theta(i) + 0.5 * h * k1_theta, x(i) + 0.5 * h * k1_x);
    k2_x = f2(theta(i) + 0.5 * h * k1_theta, x(i) + 0.5 * h * k1_x);

    k3_theta = f1(theta(i) + 0.5 * h * k2_theta, x(i) + 0.5 * h * k2_x);
    k3_x = f2(theta(i) + 0.5 * h * k2_theta, x(i) + 0.5 * h * k2_x);

    k4_theta = f1(theta(i) + h * k3_theta, x(i) + h * k3_x);
    k4_x = f2(theta(i) + h * k3_theta, x(i) + h * k3_x);

    theta(i + 1) = theta(i) + (h / 6) * (k1_theta + 2 * k2_theta + 2 * k3_theta + k4_theta);
    x(i + 1) = x(i) + (h / 6) * (k1_x + 2 * k2_x + 2 * k3_x + k4_x);

    % Calculate total energy at each step
    kinetic_energy = 0.5 * m * (L * x(i + 1))^2; % Kinetic energy = 0.5 * m * v^2 (mass m
is considered 1 for simplicity)
```

```
    potential_energy = m * g * L * (1 - cos(theta(i + 1))); % Potential energy = m * g * h
(mass m is considered 1 for simplicity)
    total_energy(i + 1) = kinetic_energy + potential_energy;
end

results = [t', theta', total_energy'];

output_file = 'pendulum_simulation_with_energy.csv';

% Export the results to a CSV file
csvwrite(output_file, results);

disp(['Results have been written to ', output_file]);

figure;
plot(t, total_energy, 'linewidth', 1);
xlabel('Time (s)', 'fontsize', 12);
ylabel('Total Energy (Joules)', 'fontsize', 12);
title('Total Energy over Time', 'fontsize', 14);
set(gca, 'fontsize', 10);
grid on;


set(gcf, 'Position', [100, 100, 1200, 600]);  % [left, bottom, width, height]

print('total_energy_plot.png', '-dpng', '-r300');  % 300 dpi for high resolution
```