CP1: Experiment 1 and 2 - Euler and Runge-Kutta

Report by: Niel Eryk De Torres

Student No.: 22337723

Grader: Oisin Creaner

17/09/2024 & 24/09/2024

Abstract

This report investigates numerical methods for solving ordinary differential equations (ODEs), focusing on the Euler Method, Improved Euler Method, and the Runge-Kutta Method (RK4). These methods were applied to first and second order systems, with results compared to exact analytical solutions. While the Euler Method showed significant accuracy issues with large step sizes, the Improved Euler Method offered better precision. The RK4 method consistently delivered highly accurate results across various scenarios. Simulations of physical systems like radioactive decay and harmonic oscillators were made using Python programs. These programs highlighted the strengths and limitations of each approach.

Introduction

Finding analytical solutions to differential equations is often difficult or even impossible, especially if the equation is nonlinear. Even when solutions are possible, they may involve complicated integrals that are impractical to evaluate exactly. In such cases, using numerical methods to provide approximate numerical solutions to the differential equation proves to be an effective alternative. In this report, the Euler Method, the Improved Euler Method, and the fourth order Runge-Kutta Method (RK4) for solving ordinary differential equations (ODEs) will be discussed. For this experiment, these methods were used to simulate systems governed by both first order and second order differential equations. The numerical solutions provided by these methods were compared to each other as well as the analytical solutions to highlight their differences and limitations.

Background and Theory

The Euler Method is a simple numerical approach to solving ODEs. This method uses the slope at an initial point to estimate the value of the dependant variable at the next point and then repeating the process using the new point as the initial point. The Euler Method can be written as follows:

$$x_{n+1} \approx x_n + h \cdot f(x_n,\, t_n) \tag{1}$$

where x_n is the current value of the dependant variable, x_{n+1} is the next value of the dependant variable, h is the step size representing the distance between successive points on the x-axis and $f(x_n,\, t_n)$ is the value of the derivative at the point $(x_n,\, t_n)$ which represent the function's slope at that point. This method requires very small values of h to be accurate. For large run times the calculated x_{n+1} becomes more inaccurate and chaotic due to the accumulations of errors used to calculate x_{n+1}.

The Improved Euler Method is a version of the Euler Method modified to improves its accuracy. This method uses an average of the slope at the initial point and the slope at the next point calculated using the original Euler Method to create a more accurate approximation. The Improved Euler Method can be written as:

$$x_{n+1} \approx x_n + \frac{1}{2} \cdot h \cdot f(x_n + h \cdot f(x_n, t_n), t_{n+1}) \tag{2}$$

where x_n, x_{n+1}, h and $f(x_n, t_n)$ are the same as described in Eqn. 1 and t_{n+1} is the next value of the independent variable (e.g. time).

The Runge-Kutta Method is a set of more complicated solutions for ODEs. The RK4 method specifically calculates four slopes within each time step and uses a weighted average to calculate the next value of the dependant variable. This method gives a much more accurate approximation than the previous methods which allows accuracy to be maintained even for larger h values. This is because errors in the RK4 Method are only on the order of h^5 while errors in the Euler and Improved Euler are on the order of h^2 and h^3 respectively.

For this experiment 12 different python programs were made to demonstrate the use of different numerical methods and test their limitations. Programs E1 to E4 made use of the Euler and Improved Euler Method while R1 to R8 made use of the RK4 Method.

Program E1: Radioactive Decay by the Euler method

A system of U-235 containing $N(t)$ nuclei at time t which is governed by the following ODE:

$$\frac{dN}{dt} = -\frac{N}{\tau} \tag{3}$$

where τ is the time constant for the decay.

This ODE can be solved analytically and gives the following solution:

$$N(t) = N_0 \exp\left(-\frac{t}{\tau}\right) \tag{4}$$

where N_0 is the number of nuclei present at time t = 0.

The Euler Method was used to give the following numerical solution for the system by combining Eqn. 1 and Eqn. 3:

$$N(t + \Delta t) \approx N(t) - \frac{N(t)\,\Delta t}{\tau} \tag{5}$$

where $N(t + \Delta t)$ represents the number of U-235 nuclei after a short time Δt has passed since the previous value of $N(t)$.

Program E2: A non-linear example

This program featured the following non-linear ODE which has no analytical solution:

$$\frac{dx}{dt} = t - x^2 \tag{6}$$

The Euler Method was used to give the following numerical solution by combining Eqn. 1 and Eqn. 6:

$$x_{n+1} = x_n + h \cdot (t - x^2) \tag{7}$$

Program E3: Coupled equations

This program featured a set of coupled first-order differential equations which represent a Simple Harmonic Oscillator (SHO) as shown below:

$$\frac{dx}{dt} = v \tag{8a}$$

$$\frac{dv}{dt} = -x \tag{8b}$$

where x is displacement, v is velocity and t is time.

The Euler Method was used to give the following numerical solutions for the system by combining Eqn. 1 with Eqn. 8a and Eqn. 8b:

$$x_{n+1} = x_n + h \cdot v \tag{9a}$$

$$v_{n+1} = v_n - h \cdot x \tag{9b}$$

The exact solutions of this system are:

$$x(t) = \sin(t) \tag{10a}$$

$$v(t) = \cos(t) \tag{10b}$$

Program E4: Modified Euler method

In this program the Improved Euler method was used to find a numerical solution for the SHO described in program E3. By combining Eqn. 2 with Eqn. 8a and Eqn. 8b the following was achieved:

$$x_{n+1} = x_n + \frac{h}{2} \cdot (y_n + (y_n - h \cdot x_n)) \tag{11a}$$

$$y_{n+1} = y_n - \frac{h}{2} \cdot (y_n + (x_n - h \cdot y_n)) \tag{11b}$$

Program R1: Using SciPy to solve ODEs

This program featured the following ODE:

$$\frac{dy}{dt} = -ay^3 + b\sin(t) \tag{12}$$

a controls the nonlinear damping term $-ay^3$ in Eq. 12 and b controls the amplitude of the external forcing term $b\sin(t)$. This means that a larger a causes the $y(t)$ to return to zero faster and larger b leads to higher oscillation in $y(t)$.

Program R2: RL Circuit

In this program the following ODE which describes the behaviour of current flowing through the circuit shown in Fig. 1. Is used:

$$\frac{dI}{dt} = \frac{V-RI}{L} \tag{13}$$

where I is current, V is voltage, R is resistance, L is inductance and t is time.

The exact analytical solution of Eqn. 12 is given by:

$$I(t) = \frac{V}{R}(1 - \exp\left(-\frac{Rt}{L}\right)) \tag{14}$$

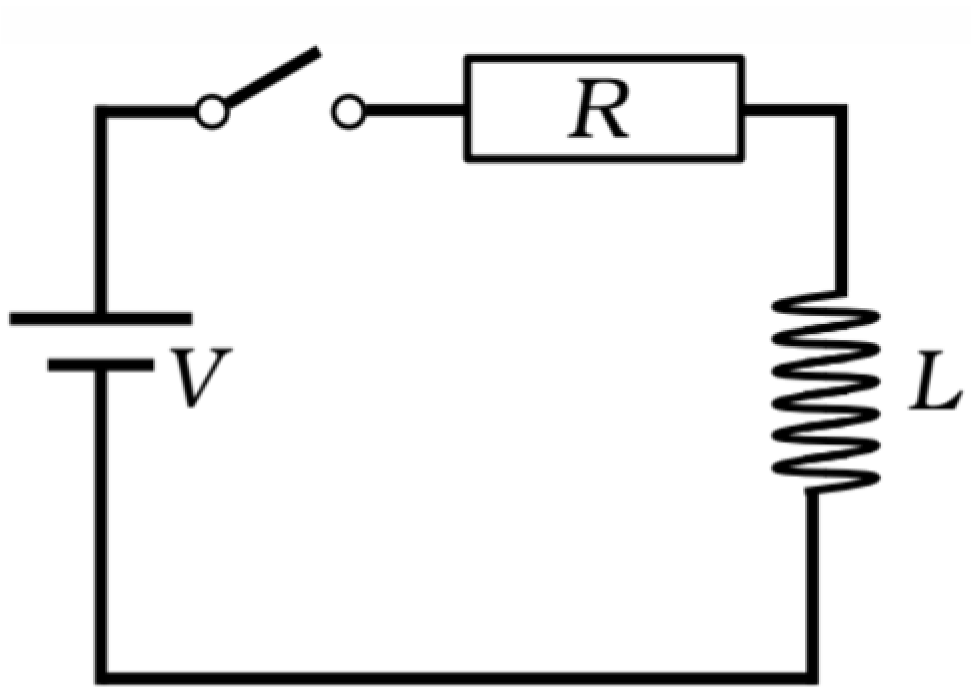

Figure 1: A diagram of an RL circuit where V is a voltage source, R is a resistor and L is an inductor. [1]

Program R4: Damped Harmonic Oscillator

This program features Eqn. 8a and the following ODE to represent a damped harmonic oscillator:

$$\frac{dv}{dt} = -bv - \omega_0^2 x \qquad (15)$$

where b is the damping coefficient and ω_0 is the resonant frequency of the system.

Program R6: Driven Oscillator

This program features Eqn. 8a and the following ODE to represent a driven oscillator:

$$\frac{dv}{dt} = -bv - \omega_0^2 x - A\sin(\omega_d t) \qquad (16)$$

where A is the amplitude of the driving oscillation and ω_d is the driving frequency of the external force.

Experimental Design and Procedure

Programs E1 to E4 are set up similarly to each other. Configurable variables like step size h and maximum time t_{max} along with initial conditions such as start time t_0 and starting position x_0 are declared at the start. These variables were set according to the Lab Instructions on Loop [2] unless otherwise stated. Arrays are then created to store values for the independent and dependant variables. A while loop is then used to calculate new values for the dependent and independent variable an equation derived from the Euler or Improved Euler method. These values are then stored in the previously created arrays. The values in the arrays are then used to generate graphs. The exact detail of what each of these programs did can also be found on Loop [2].

Programs R1 to R8 were completed and modified versions of programs provided on GitHub [3].

Program R1: Using SciPy to solve ODEs

The contents of the provided R1_template.py [3] are described as follows. First all necessary libraries are imported. A function called “nonlinear()” is defined as Eqn. 12. which requires inputs t and y. All the following code is written within a function called “main()”. This is to help with portability and reusability. Initial variables and arrays to store values are created. The “solve_ivp()” function from SciPy is then used to solve the ODE. This function takes the derivative function defined earlier, range of time values, initial state of the system, the specific RK Method and time points as arguments. The solution to the ODE is then stored within the “result” array. This array is read and used to generate a plot.

The original R1 template program was modified to add the variables a and b as inputs. This was done by adding them as inputs needed for the “nonlinear()” function as well as adding an args input into “solve_ivp()” which contained a and b. Multiple a and b inputs were ran through the program to produce Fig. 10.

Program R2: RL Circuit

This program is based on the previous one but the “nonlinear()” function is replaced with working versions of the ones outlined in R2_function_outlines. These functions contain Eq. 13 & Eqn. 14. A function called “h_n()” was added to convert inputs of time steps h into a number of steps n that the program can use. Multiple values of h were inputted into the program to generate different solutions which were plotted and compared to each other as shown in Fig. 11. The original values of $V = 10V, R = 50\Omega$ and $L = 100H$ were then changed to see how it would affect the curve as shown in Fig. 12.

Program R3: Harmonic Oscillator

This program was created by merging the provided R3_template.py and R3_phase_space.py [3] and placing both of their output graphs into a subplot together as shown in Fig. 13. The exact solutions Eqn. 10a & Eqn. 10b were also added to the graph to compare with the numerical solutions.

Program R4: Damped Harmonic Oscillator

This is based on the previous one but replaced the "simple_pendulum()" function with the one provided in R4_function.py [3]. The additional inputs of b and ω_0 also had to be added to the "solve_ivp()" function. The b and ω_0 inputs were changed to generate a time dependant and phase space plot for an under damped, critically damped and over damped harmonic oscillator. For the under damped system shown in Fig. 14, $b = 0.1$ and $\omega_0 = 1$. For the critically damped system shown in Fig. 15, $b = 2$ and $\omega_0 = 1$. For the over damped system shown in Fig. 16, $b = 4$ and $\omega_0 = 1$.

Another function called "is_at_rest()" was used to determine when the system returned to rest. It took x, v and a minimal threshold ϵ as inputs. When used in a loop It returns the time at which both x(t) and v(t) are less than ϵ which was set to 0.01.

Program R5: Lambda function

This program is almost identical to the previous one. The main difference is that uses a lambda function as the function inputted into solve_ivp() to add additional inputs to the function.

Program R6: Driven Oscillator

This program is based on the previous one but the function "damped_pendulum()" is replaced with one that makes use of Eqn. 16. Functions in the provided R6_plot.py [3] were used to implement multiple plots in one. The program used values A = 1.2, b = 0.2, ω_0 = 1.2 and various values of ω_d to generate Fig. 17.

Program R7: Shape of the resonance

This program is a modified version of R6 where code was changed to include the function "placeholder_amplitudes()" provided in R7_amplitudes.py [3] . The program used values A = 1.5, b = 0.3 and ω_0 = 2 to generate Fig. 18.

Program R8: Amplitude of the resonance

This program is a modified version of R7 which implements the changes demonstrated in the provided R8_excerpt.py [3]. The program used values A = 1, b = 0.01, ω_0 = 1 and various values of ω_d to generate Fig. 19.

Results and Discussion

Since all values are calculated for simulations and units are usually not provided, units will usually not be included in graphs.

Program E1: Radioactive Decay by the Euler method

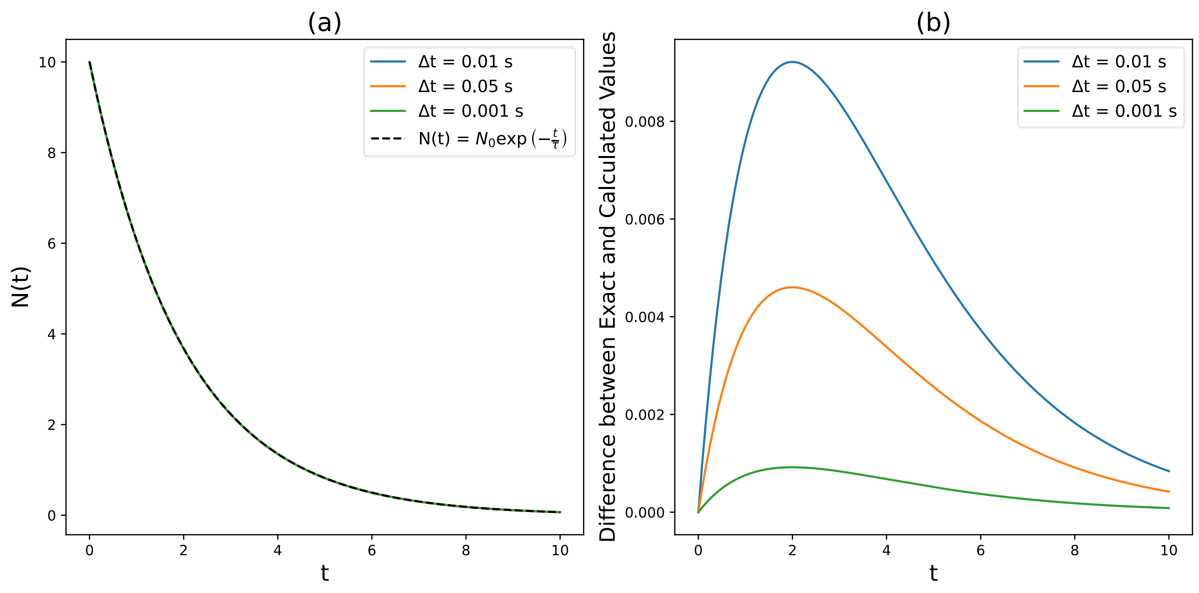

Fig. 2. (a) The graph compares numerical solutions from Eqn. 5 using different values of Δt with the exact solution from Eqn. 4. (b) This graph shows the difference between the exact and numerical solutions over time.

The fact that the different numerical solutions almost perfectly overlap with the exact solution in Fig. 2 (a) shows that the Euler Method is able to accurately approximate the exact solution when small Δt is used. Fig. 2. (b) shows that the difference between exact and numerical solution increases with larger Δt and that all differences are largest at around t = 2s. As expected the differences converge to zero as time progresses.

Program E2: A non-linear example

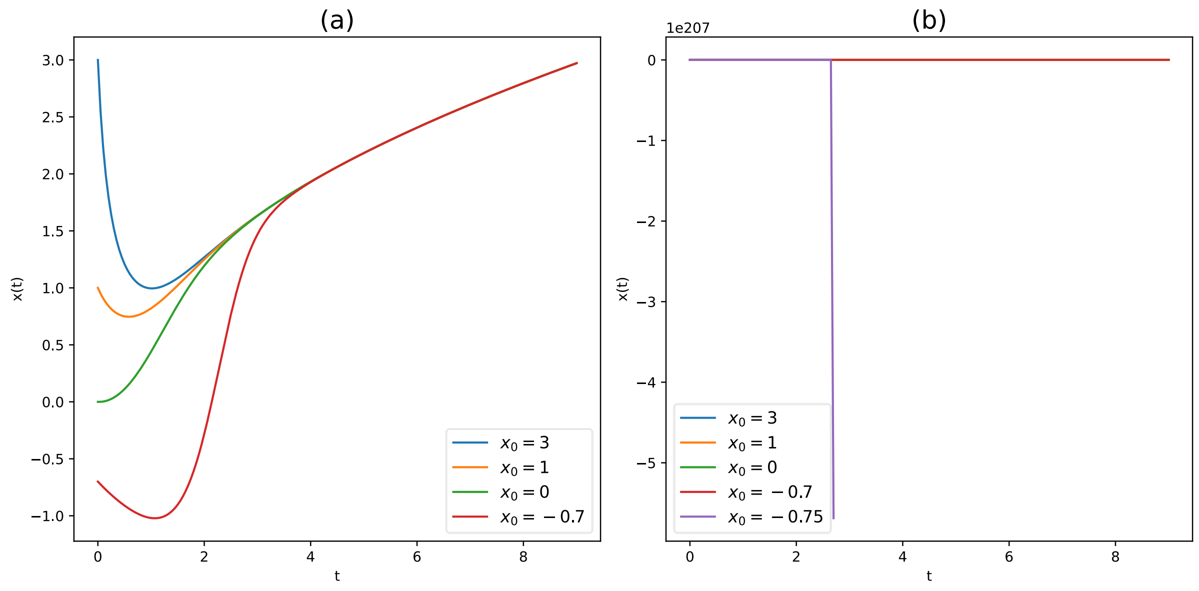

Fig. 3. (a) The graph shows the numerical solutions given by Eqn. 7 when using different initial x (x_0) values. (b) This graph shows the result of adding another solution with $x_0 = -0.75$.

The various solutions converge in Fig. 3 (a) due to the nature of Eqn. 7. When x_0 > -0.75 the t term becomes the main factor controlling how $x(t)$ behaves as t increases. The fact that the graph in Fig. 3 (b) appears to break is also due to the nature of Eqn. 7. When x_0 ≤ -0.75 the $-x^2$ term becomes the main factor controlling how $x(t)$ behaves as t increases. The $-x^2$ term acts like a "slowing force" so it being dominant causes the curve to not diverge with the others and behave strangely.

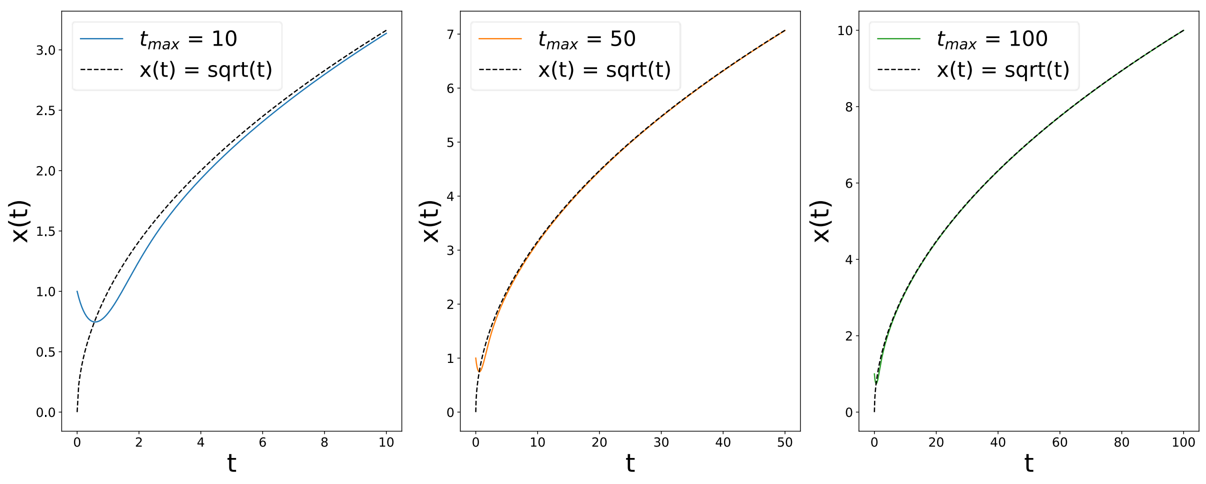

Fig. 4. These graphs compare the same numerical solution given by Eqn. 7 against different values of t_{max} with the curve $x = \sqrt{t}$ as the dashed line.

The graphs in Fig. 4 show the Euler Method working as expected for these values of t_{max}.

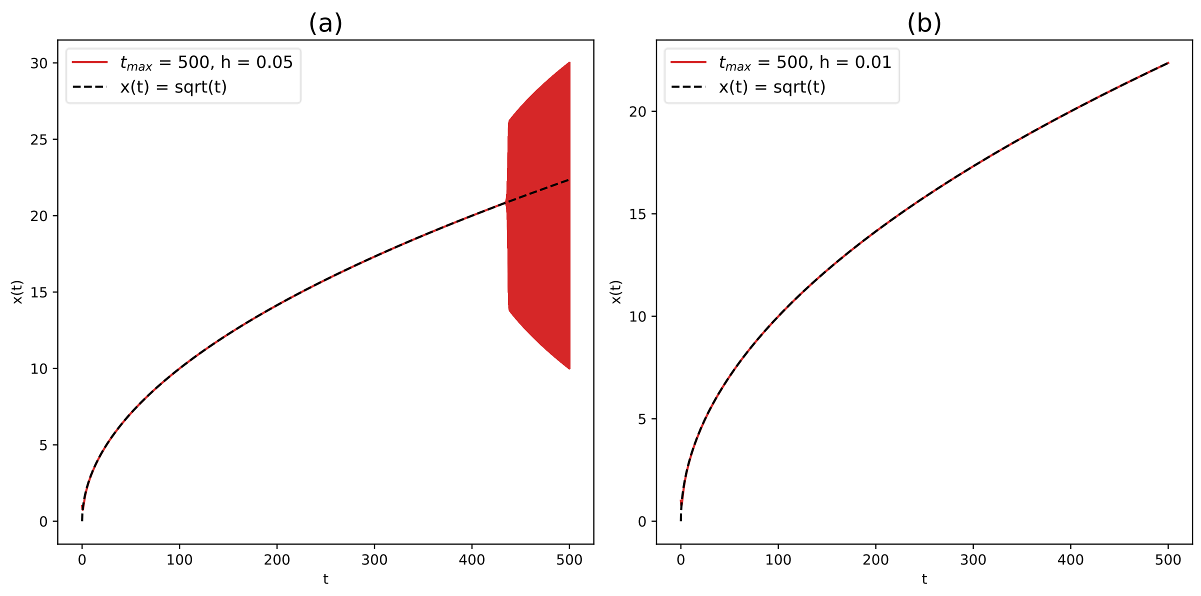

Fig. 4. (a) The graph plots the numerical solution given by Eqn. 7 against a larger value of t_{max}. (b) This graph shows the result of decreasing the step size to $h = 0.01$.

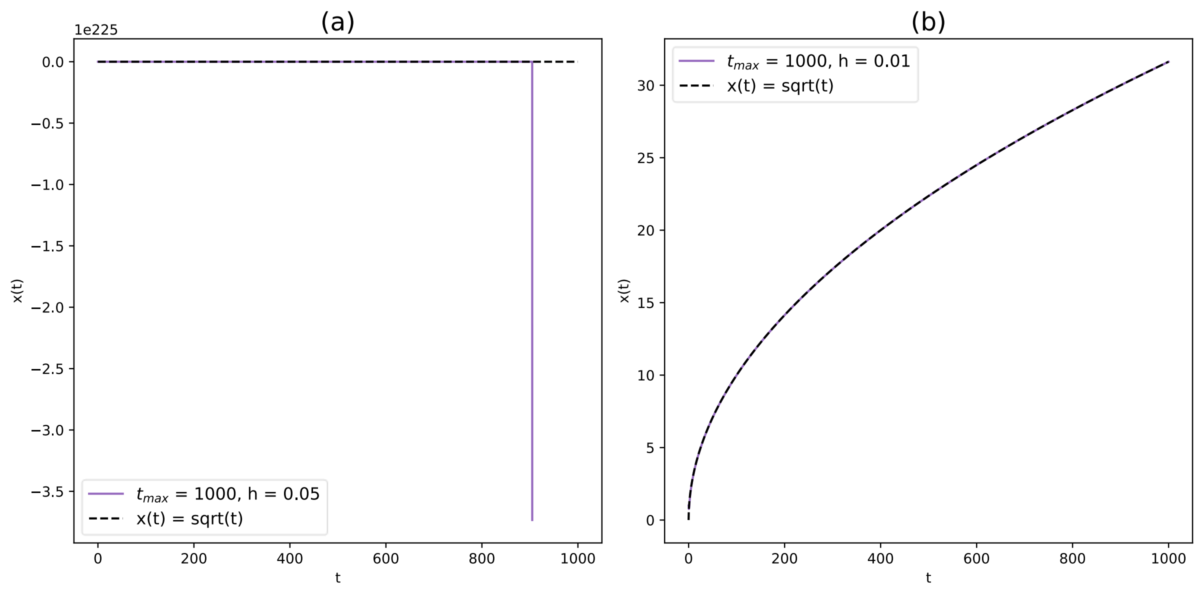

Fig. 5. (a) The graph plots the numerical solution given by Eqn. 7 against an even larger value of t_{max}. (b) This graph shows the result of decreasing the step size to $h = 0.01$

Both Fig. 4 (a) and Fig. 5 (b) both demonstrate the Euler Method breaking down at larger t_{max} due to accumulated error being used to calculate successive points. Fig. 4 (b) and Fig. 5 (b) demonstrate how reducing the step size h can help prevent the Euler Method breaking down for larger t_{max} values.

Appropriate limits for a given application are determined by the nature of the application, types of real-world error and practical constraints. Safety-critical applications have an extremely low tolerance for error due to the more severe consequences of error which means limits must be tighter. Limits should consider measurement, systematic and random uncertainty when being set. Physical properties like friction must be taken into account when limits are set.

Program E3: Coupled equations

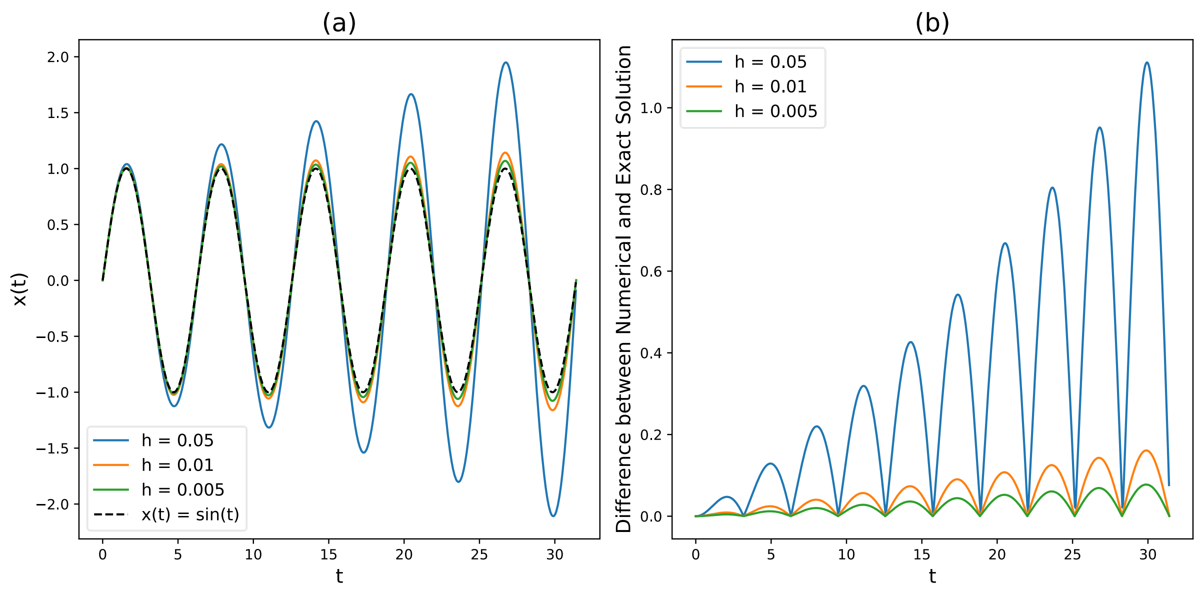

Fig. 6. (a) Multiple numerical solutions from using Eqn. 9a with different values for h plotted against time with the exact solution Eqn. 10a as the dashed line. (b) The absolute difference between numerical and exact solutions plotted against time.

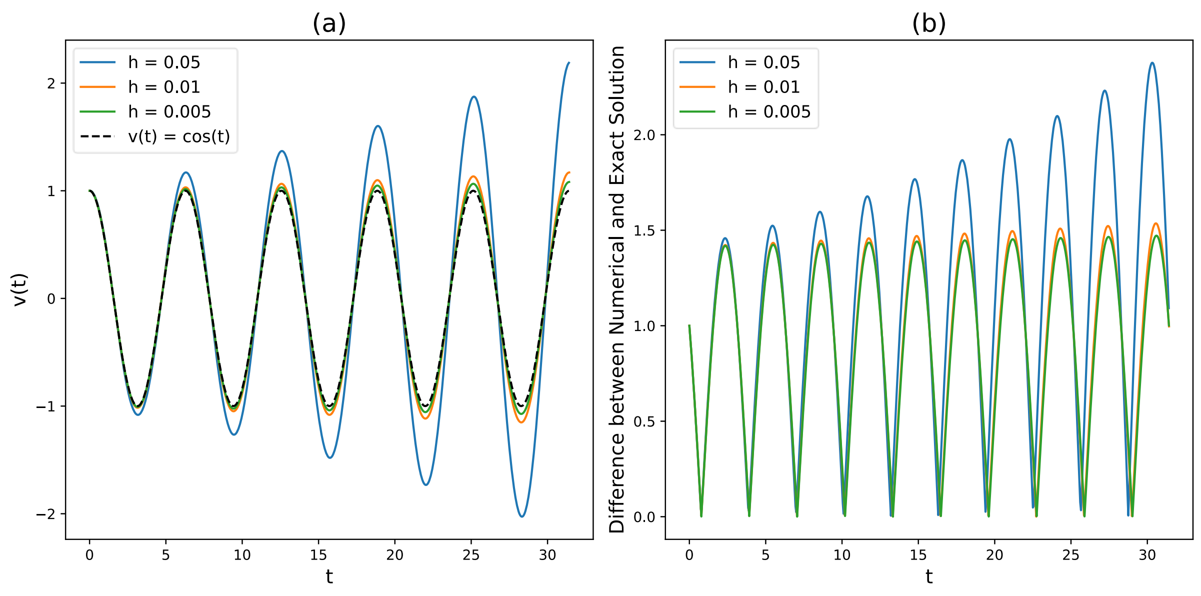

Fig. 7. (a) Multiple numerical solutions from using Eqn. 9b with different values for h plotted against time with the exact solution Eqn. 10b as the dashed line. (b) The absolute difference between numerical and exact solutions plotted against time.

Both Fig. 6 and Fig. 7 display how the numerical solution from the Euler Method diverges increasingly from the actual solution as time progresses. They also once again demonstrate how this deviation can be reduced by reducing the size of step size h. The solutions shown in these figures can be improved by further decreasing the size of h.

Program E4: Modified Euler method

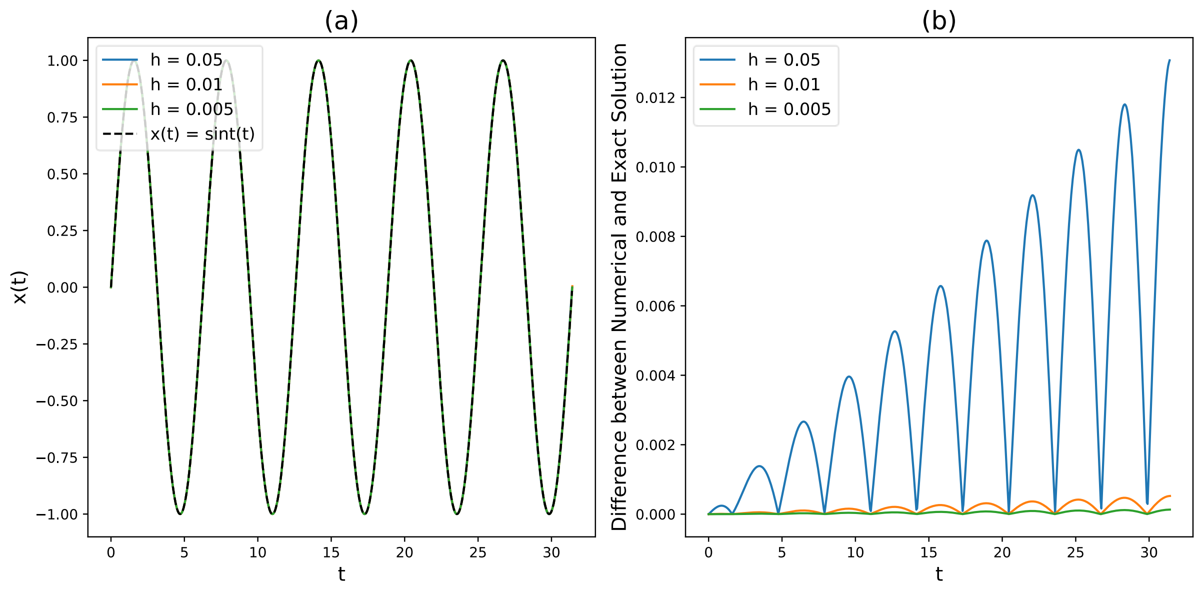

Fig. 8. (a) Multiple numerical solutions from using Eqn. 11a with different values for h plotted against time with the exact solution Eqn. 10a as the dashed line. (b) The absolute difference between numerical and exact solutions plotted against time.

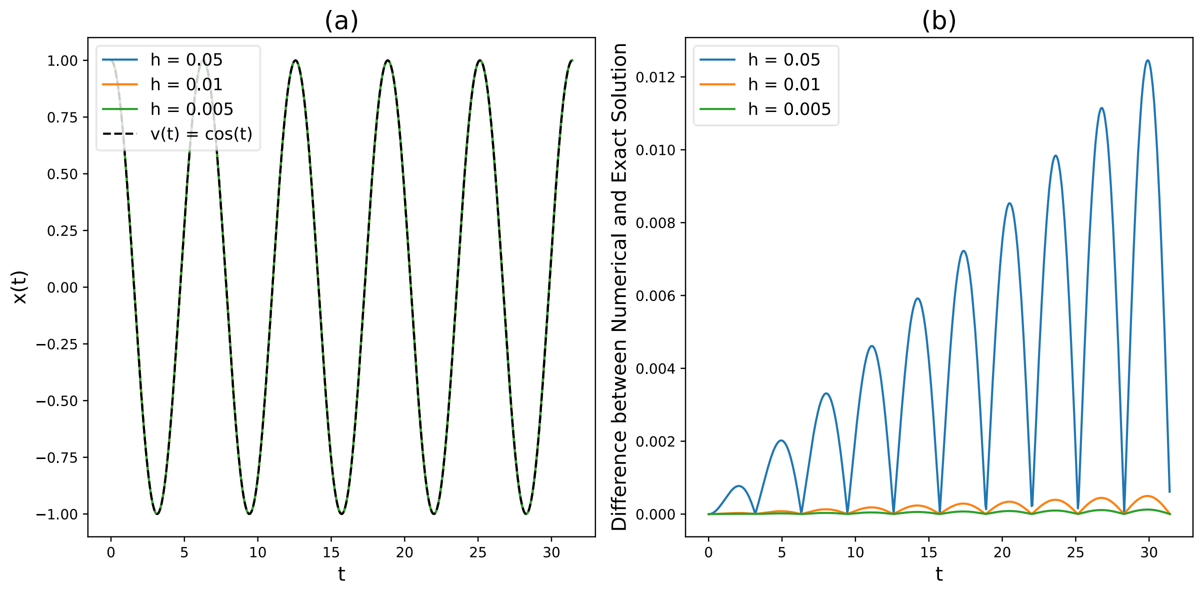

Fig. 9. (a) Multiple numerical solutions from using Eqn. 11b with different values for h plotted against time with the exact solution Eqn. 10b as the dashed line. (b) The absolute difference between numerical and exact solutions plotted against time.

The numerical solutions given by the Improved Euler Method are a large improvement from Euler Method. This is shown by the near perfect agreement between numerical and actual solutions in Fig. 8 (a) and Fig. 9 (a). Although increasing deviation from the exact solution as time increases is still present as seen in Fig. 8 (b) and Fig. 9 (b), it is orders of magnitude smaller than what it was for the Euler method. Decreasing step size h can still be used to decrease the difference between numerical and actual solutions.

Program R1: Using SciPy to solve ODEs

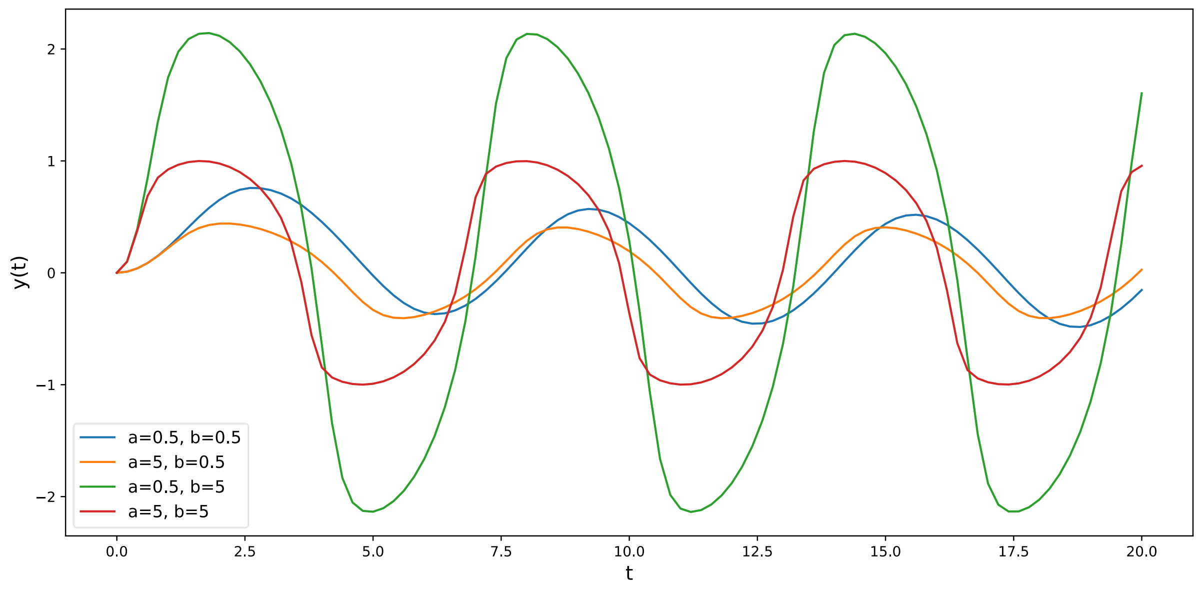

Fig. 10. Multiple numerical solutions for Eqn. 12 when using different a and b values. These solutions were found using RK4 Method.

Fig. 10. Shows that when a is higher $y(t)$ returns to zero faster and that when b is higher $y(t)$'s amplitude is higher. This is in line with expectations.

Program R2: RL circuit

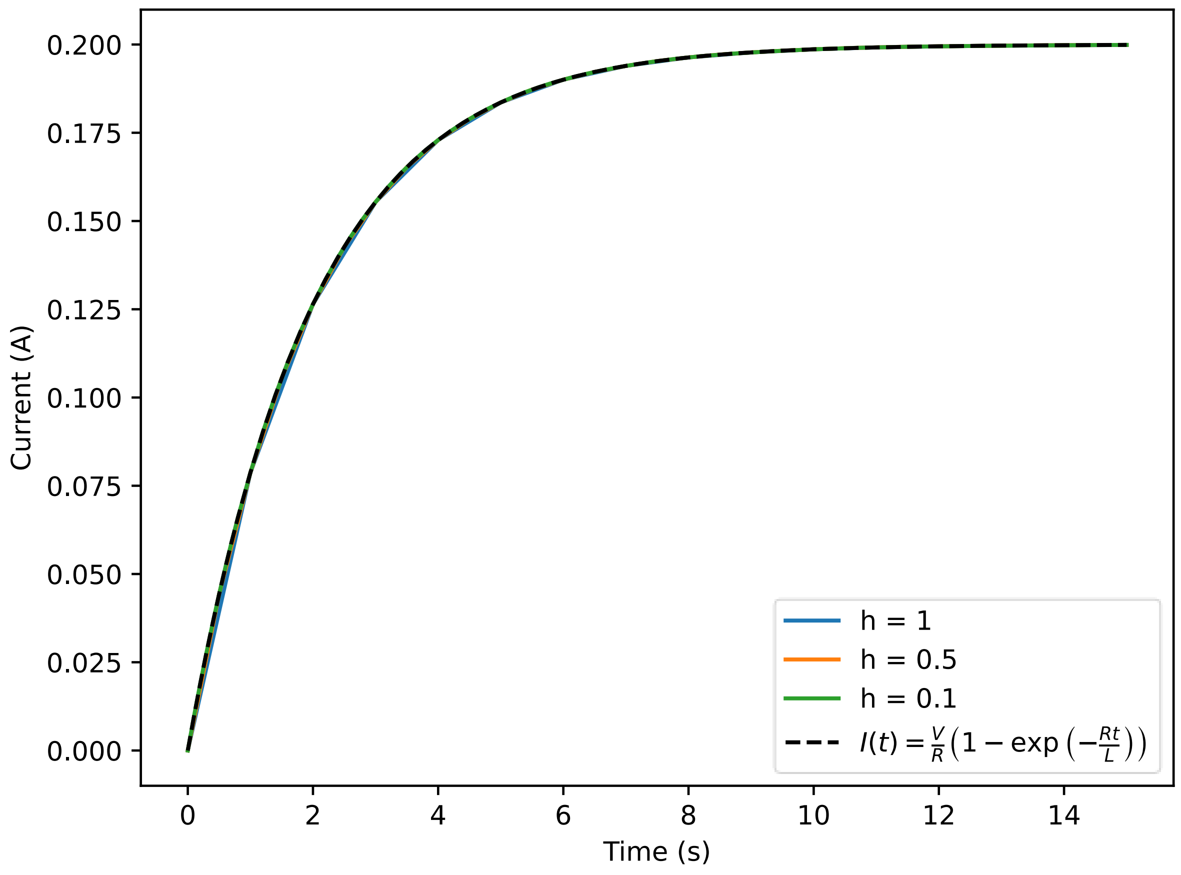

Fig. 11. Multiple numerical solutions for Eqn. 13 found using the RK4 Method when using different time step (h) values. The exact solution of Eqn. 14 is shown as the dashed line.

The fact that the numerical solutions and the exact solution in Fig. 11 are indistinguishable despite the relatively high step size h is a clear example of how much of an improvement the RK4 Method is compared to the Euler Method.

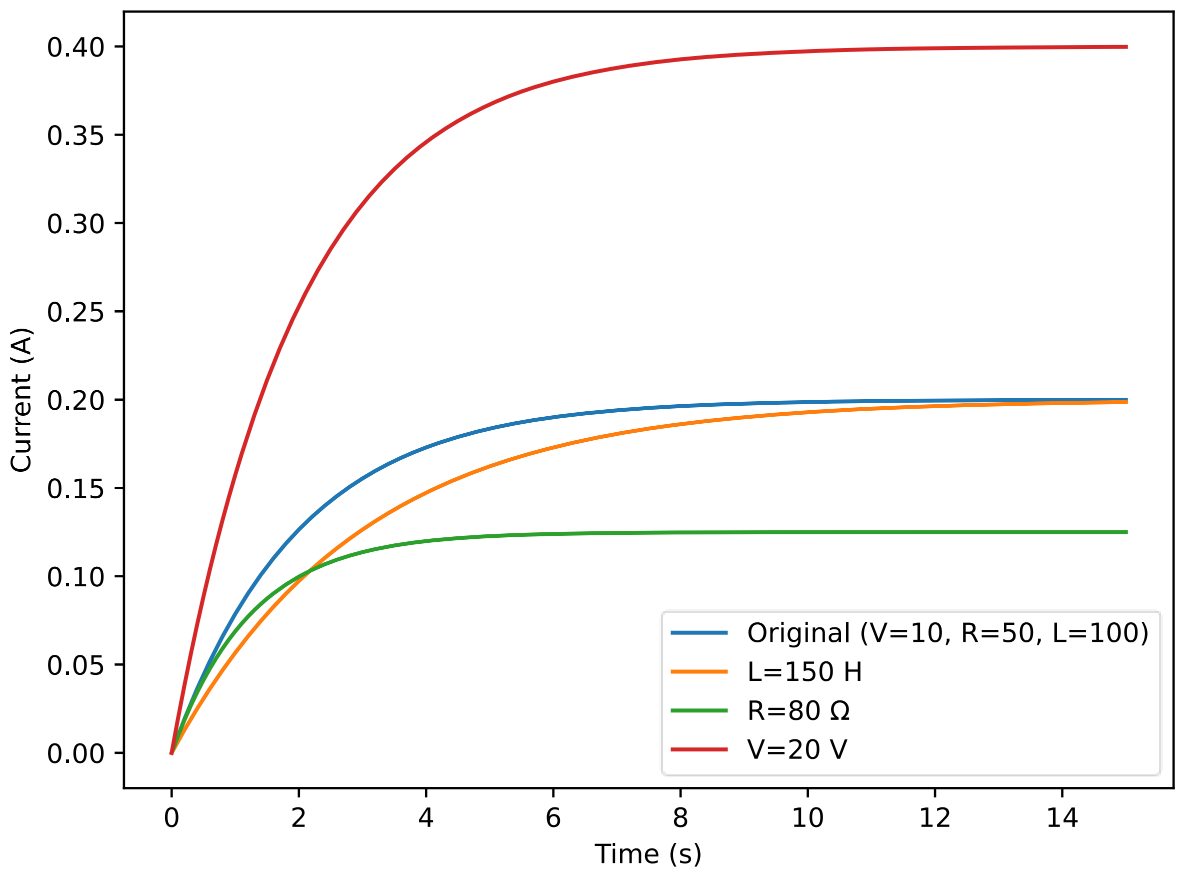

Fig. 12. This graph shows how the curve changes when the Voltage, Resistance or Inductance is increased.

Increasing R slows down the rate at which the current reaches its steady-state value since there is more opposition to the current's flow. Increasing V leads to a higher steady-state current which is in line with Ohm's Law. Increasing L increases the time taken for the current to reach its steady state. This is because the inductor resists change in the current more strongly.

Program R3: Harmonic Oscillator

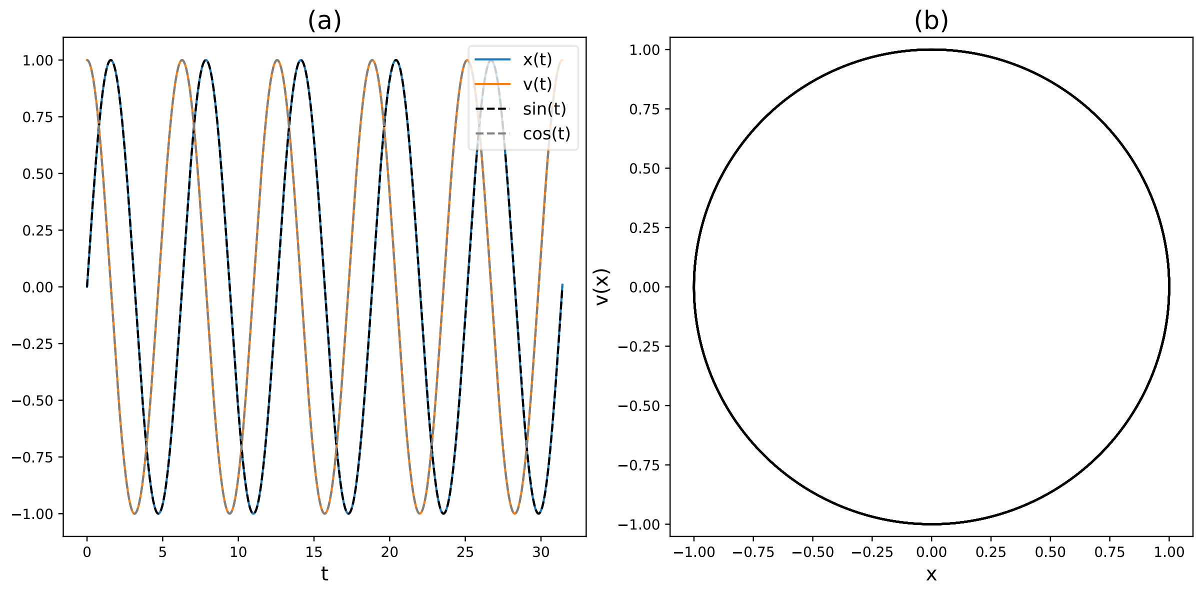

Fig. 13. (a) Numerical solutions for Eqn. 8a & 8b found using the RK4 Method compared to exact solutions Eqn. 10a & 10b. (b) The phase space plot of the SHO.

The numerical solutions derived from the RK4 method are even more accurate than the Improved Euler method and does not require a low h to maintain its accuracy. The RK4 method is preferred due to its increased accuracy for any value of h.

Program R4: Damped Harmonic Oscillator

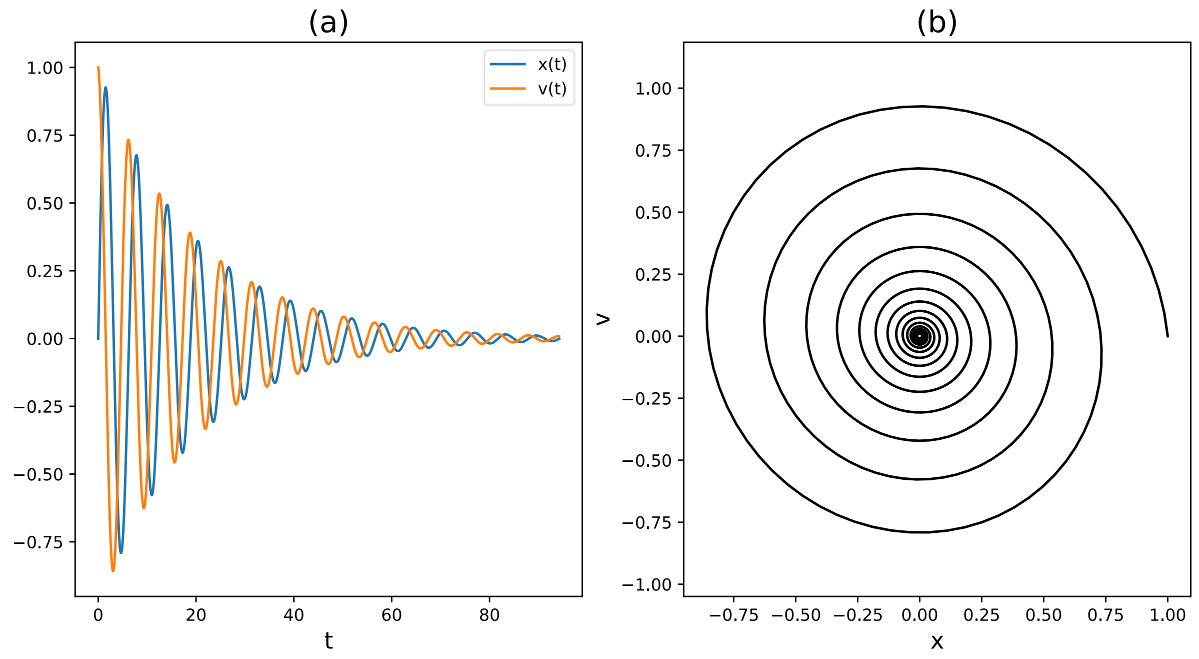

Fig. 14. (a) Numerical solutions for Eqn. 8a & 15 found using the RK4 Method when the system is under damped. (b) The phase space plot of an under damped harmonic oscillator.

The system in Fig. 14 comes to rest at t = 85.67. The accuracy of this time could be increased by reducing the size of the minimal threshold to determine if the sytem is at rest ϵ. As expected the under damped sytem oscillates as it returns to equilibrum.

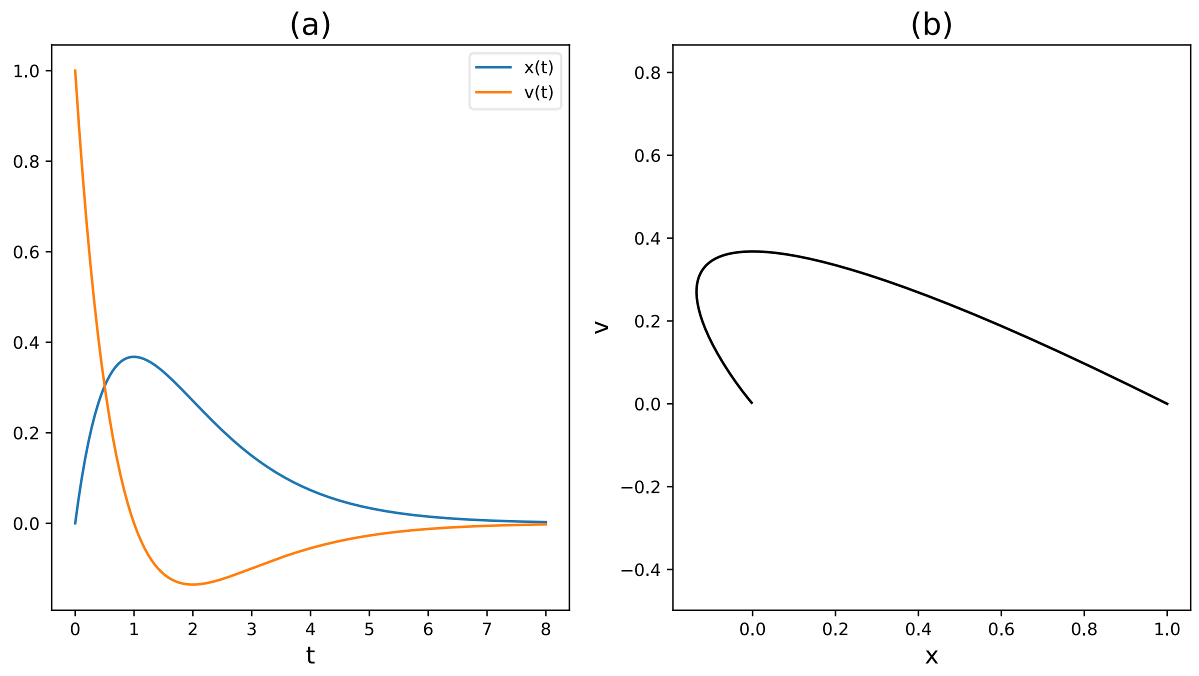

Fig. 15. (a) Numerical solutions for Eqn. 8a & 15 found using the RK4 Method when the system is critically damped. (b) The phase space plot of a critically damped harmonic oscillator.

The system in Fig. 15 comes to rest at t = 6.5. The accuracy of this time could be increased by reducing the size of the minimal threshold to determine if the sytem is at rest ϵ. As expected the critically damped sytem doesn't oscillates as it returns to equilibrum very quickly.

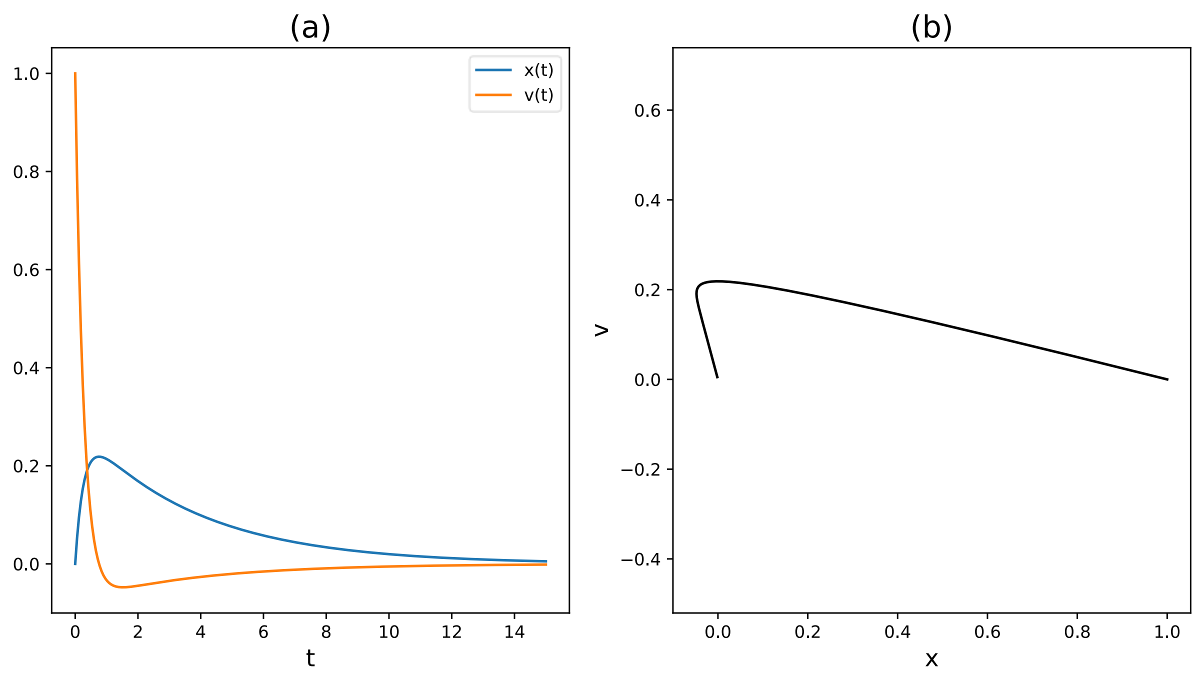

Fig. 16. (a) Numerical solutions for Eqn. 8a & 15 found using the RK4 Method plotted against t when the system is over damped. (b) The phase space plot of an over damped harmonic oscillator.

The system in Fig. 16 comes to rest at t = 12.63. The accuracy of this time could be increased by reducing the size of the minimal threshold to determine if the sytem is at rest ϵ. As expected the over damped sytem doesn’t oscillates as it returns to equilibrum slower than when critically damped.

Program R5: Lambda function

Lambda functions are much more concise but also limited due to being restricted to a single expression compared to a traditional function. Lambda functions don’t have to be named so they are useful for quick functions that are only needed once but this also makes them harder to debug. Lambda functions are useful for being used as arguments for higher order functions like in the “solve_ivp()” function used in R5.

Program R6: Driven Oscillator

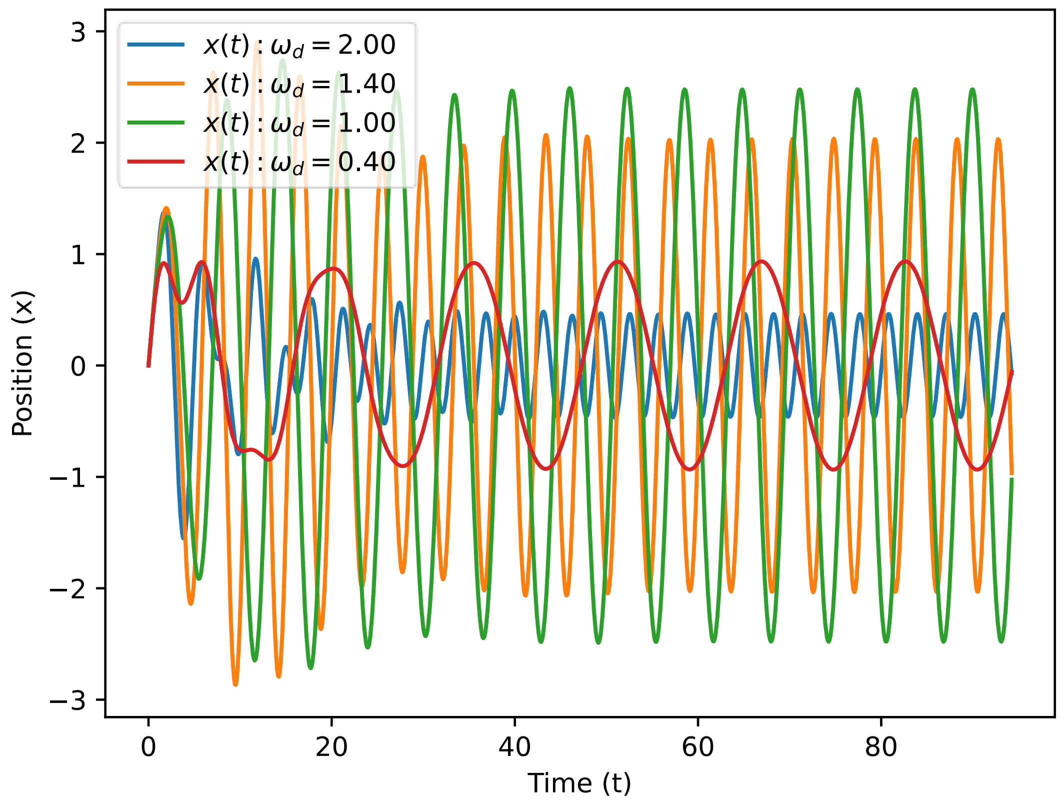

Fig. 17. Solutions of a damped driven harmonic oscillator described by Eqn. 8a & 16 with varying ω_d.

The solutions that use a ω_d value closer to the value of ω_0 = 1.2 display a much higher amplitude than values further from it. This is because of resonance, as ω_d gets closer in value to ω_0 the external energy acting on the system is more efficiently transferred to the system. When ω_d is low the amplitude smaller since the external force is too weak to effectively drive stronger oscillation. When ω_d is relatively high the driving force is too fast for the system to respond effectively which leads to damping that causes lower amplitude.

Program R7: Shape of the resonance

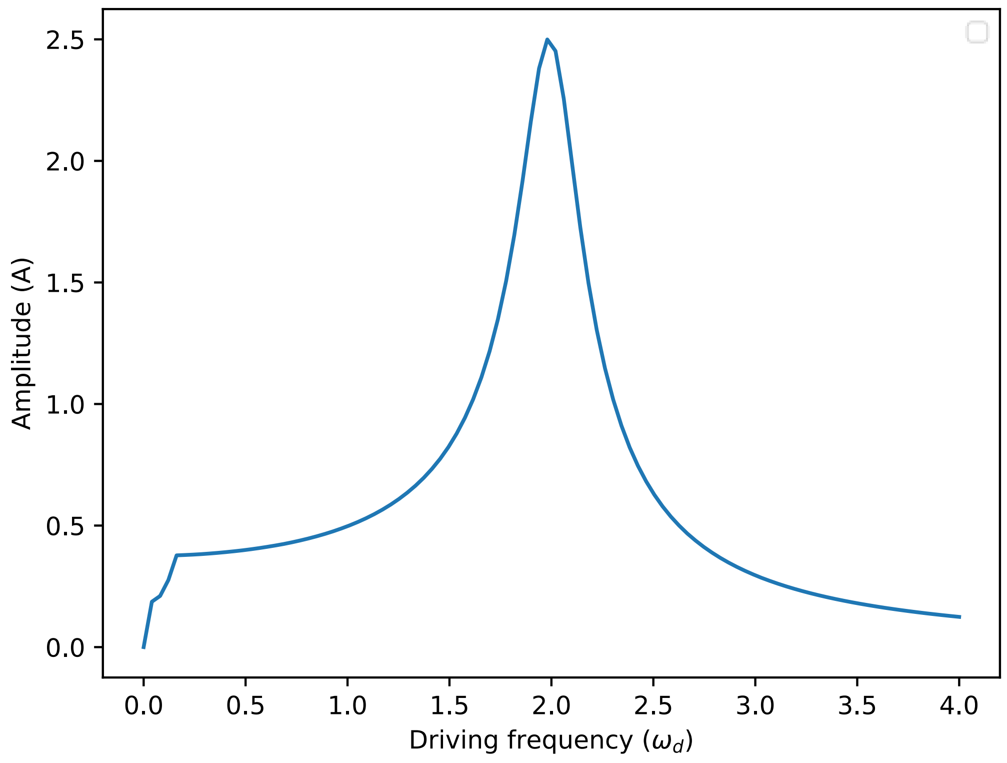

Fig. 18. A plot of amplitude against driving frequency for an under damped system: A = 1.5, b = 0.3 and ω_0 = 2.

The peak of the amplitude is directly situated at when ω_0 = ω_d which is when resonance is at its highest.

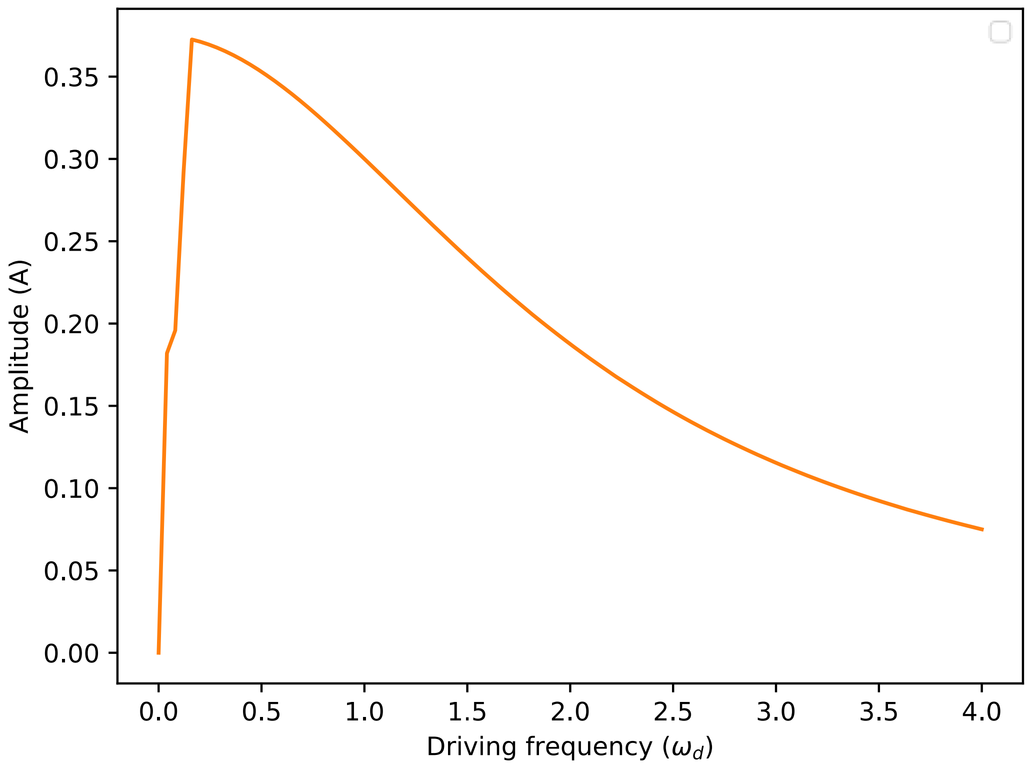

Fig. 19. A plot of amplitude against driving frequency for a critically damped system: A = 1.5, b = 4 and ω_0 = 2.

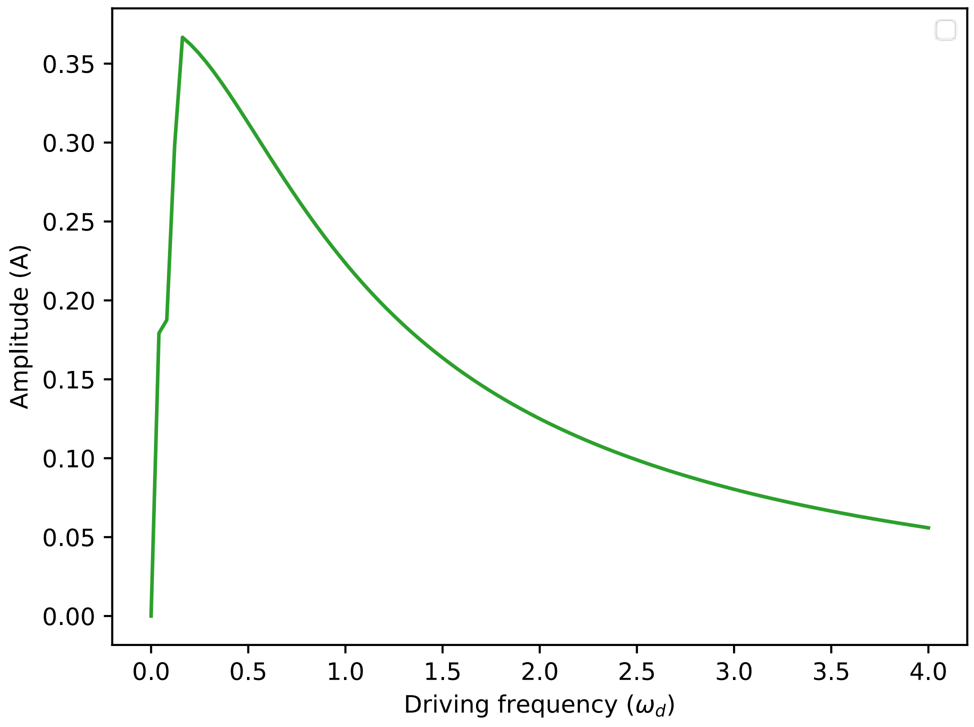

Fig. 20. A plot of amplitude against driving frequency for an over damped system: A = 1.5, b = 6 and ω_0 = 2.

Neither Fig. 19 or Fig. 20 display resonance due to the high damping preventing oscillations.

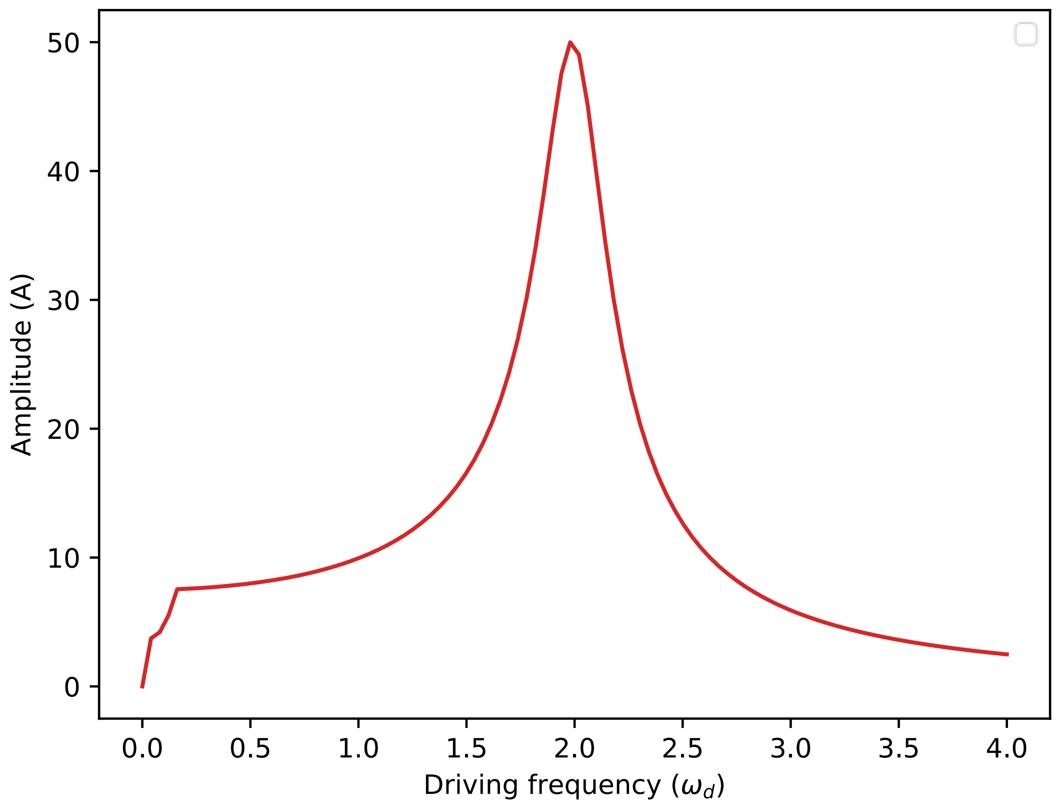

Fig. 21. A plot of amplitude against driving frequency when amplitude is high: A = 1.5, b = 6 and ω_0 = 2.

Increasing the driving force's amplitude simply increases the overall amplitude that the system oscillates at.

Program R8: Amplitude of the resonance

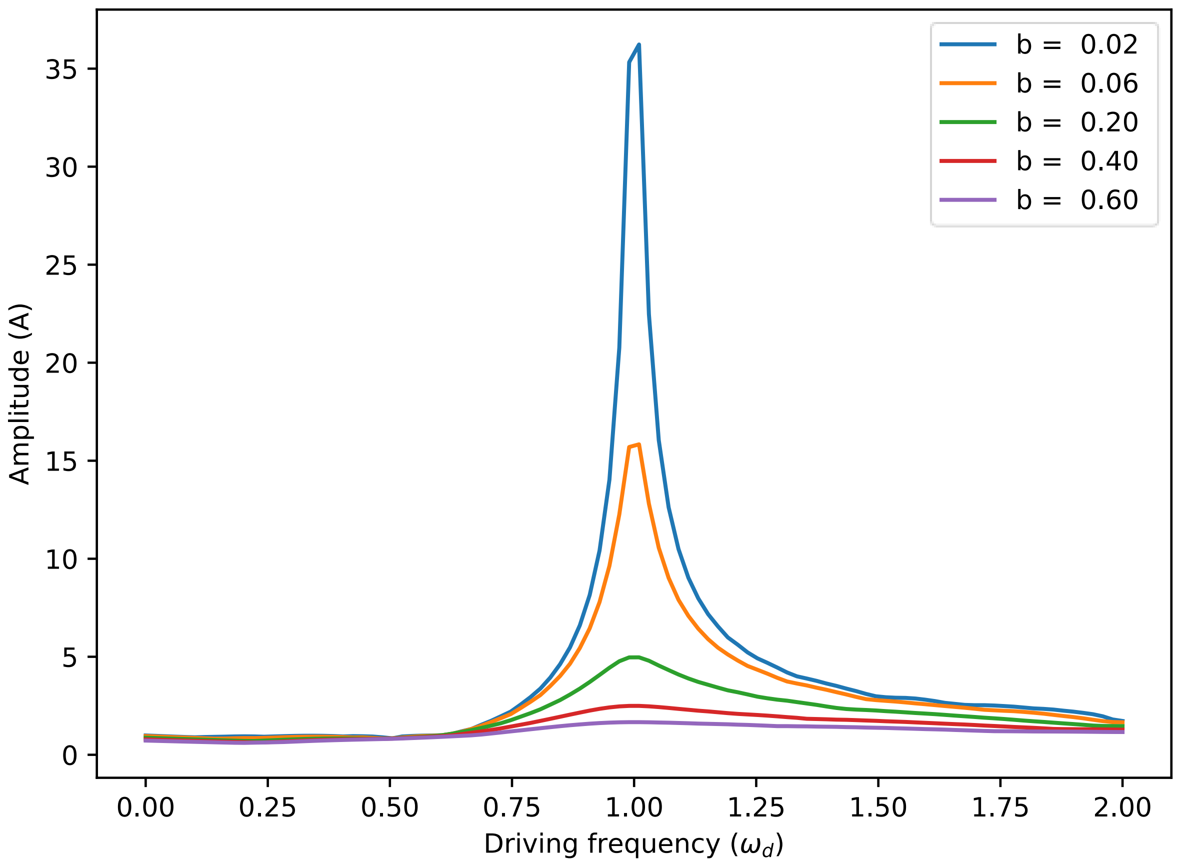

Fig. 22. A plot of amplitudes against driving frequencies using A = 1, ω_0 = 1 and different values of b.

When b is lower it reaches much higher amplitudes when driven close to ω_0 but it drops of rapidly as it gets further. Its resonance is said to be “sharp”. When b is increases the peak becomes broader with a less dramatic fall-off as its driven away from ω_0. Their resonance is said to be “broadened”.

As the program was being modified from R1 to R8 it was important to keep track of which parts of the program were being used. By removing redundant functions and other parts of code the programs became easier to understand and debug.

Conclusion

The numerical methods examined in this report provide useful alternatives to analytical solutions for ODEs, especially when dealing with nonlinear systems or when exact solutions are impractical. The Euler Method suffers from significant errors for large step sizes, while the Improved Euler Method offers a notable improvement in accuracy by averaging slopes. The RK4 method proved to be the most accurate, with the ability to handle larger step sizes while maintaining precision, making it highly advantageous for simulating complex dynamic systems such as harmonic oscillators and damped systems. For all methods decreasing the step size improved accuracy but this came at the cost of increased computational time. Therefore, the RK4 method is the preferred choice for most applications when precision is required.

References

[1] Runge Kutta Method Lab Instructions:
https://docs.google.com/document/d/1cTBW6-qEAg40SqrNtc20BFRbNBC0XGgj_t5His9Sop4/edit?usp=sharing

[2] Euler Method Lab Instructions:
https://docs.google.com/document/d/1SvP9KsgSpeeBHf9gKj8vsK_vqwVxFusoszuSNWmg1rs/edit?usp=sharing

[3] Repository of used templates and functions:
https://github.com/creanero/DCU_PHY1055/blob/main/Runge-Kutta/R1_template.py