**Applied Numerical Methods**

Train suspension spring-mass system

**Done by:** Rawan Elalfy 230382

Yasmeen Motaz 237676

Abdelrahman Emad 235338

**Module Code:** 24MTRN13C

Table of Contents

[Introduction 2](#_Toc185024361)

[Problem 3](#_Toc185024362)

[Solution 4](#_Toc185024363)

[MatLab code 5](#_Toc185024364)

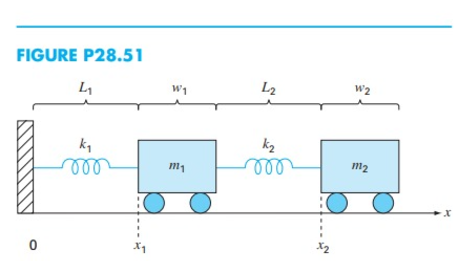
[Comparison 9](#_Toc185024365)

[Errors 11](#_Toc185024366)

[Conclusion 12](#_Toc185024367)

# Introduction

A train's suspension system is a practical example of a spring-mass system, a fundamental mechanical model used to describe oscillatory motion. The interaction between the train's body and its suspension—specifically how the wheels and chassis respond to forces like bumps, vibrations, and uneven tracks—is commonly represented using this model.

In this system, the suspension, designed to absorb shocks and minimize vibrations, is modeled as the "spring," while the train's body or individual carriages represent the "mass." When the spring is compressed or stretched, it exerts a restoring force, causing the mass to oscillate under the laws of motion. The diagram in Fig. (28.51) illustrates how the train's suspension system mitigates vibrations, enhancing passenger comfort, safety, and overall stability during travel.

# Problem

**A diagram of a machine

Description automatically generated**

# Solution

**Using 3 methods:**

**Euler’s Method:**

The **Euler method** is the simplest numerical method for solving ordinary differential equations (ODEs). It is a **first-order** method, meaning its accuracy increases linearly with the step size.

A black and blue text

Description automatically generated with medium confidence

**Heun’s method:**

A white background with black text

Description automatically generatedA math equations on a white background

Description automatically generatedThe **Heun method** is an improvement over the Euler method. It is a **second-order** method, meaning its error decreases quadratically as the step size decreases. It improves accuracy by averaging the slope at the start and end of the interval.

**Runge–Kutta:**

The **Runge-Kutta** methods are a family of higher-order methods for solving ODEs. The most used version is the **fourth-order** Runge-Kutta method (RK4), which provides a good balance of accuracy and computational efficiency. Using the fourth-order method:

A math equations and numbers

Description automatically generated with medium confidence

## MatLab code

clear all

% Parameters

% Use these values as a reference

% k1 = 5; k2 = 5; % Spring constants (N/m)

% m1 = 2; m2 = 2; % Masses (kg)

% L1 = 2; L2 = 2; % Unstretched spring lengths (m)

% w1 = 5; w2 = 5; % Widths of the masses (m)

% h = 0.01; %step size

k1 = input('Please enter spring constant 1:\n');

k2 = input('Please enter spring constant 2:\n');

m1 = input('Please enter mass 1:\n');

m2 = input('Please enter mass 2:\n');

L1 = input('Please enter the unstretched length of spring 1:\n');

L2 = input('Please enter the unstretched length of spring 2:\n');

w1 = input('Please enter the width of mass 1:\n');

w2 = input('Please enter the width of mass 2:\n');

h = input('Please enter the step size:\n');

% Initial conditions

i=1;

x1(i) = L1;

x2(i) = L1 + w1 + L2 + 6;

v1(i)= 0;

v2(i) = 0;

t0 = 0;

tf = 20;

t0 = t0:h:tf;

n\_steps = length(t0);

% Initialize arrays for results

x1\_euler(i) =0;

x2\_euler(i)=0;

v1\_euler(i) = 0;

v2\_euler(i) =0;

x1\_heun(i) = 0;

x2\_heun(i) =0;

v1\_heun(i) = 0;

v2\_heun(i) = 0;

x1\_rk4(i) = 0;

x2\_rk4(i)= 0;

v1\_rk4(i) =0;

v2\_rk4(i) =0;

% Set initial values

x1\_euler(1) = x1(i);

x2\_euler(1) = x2(i);

v1\_euler(1) = v1(i);

v2\_euler(1) = v2(i);

x1\_heun(1) = x1(i);

x2\_heun(1) = x2(i);

v1\_heun(1) = v1(i);

v2\_heun(1) = v2(i);

x1\_rk4(1) = x1(i);

x2\_rk4(1) = x2(i);

v1\_rk4(1) = v1(i);

v2\_rk4(1) = v2(i);

% Function definitions for derivatives

dx1dt = @(v1) v1;

dx2dt = @(v2) v2;

dv1dt = @(x1, x2) (-k1/m1)\*(x1 - L1) + (k2/m1)\*(x2 - x1 - w1 - L2);

dv2dt = @(x1, x2) (-k2/m2)\*(x2 - x1 - w1 - L2);

% Euler's Method

for i = 1:n\_steps-1

x1\_euler(i+1) = x1\_euler(i) + h \* dx1dt(v1\_euler(i));

x2\_euler(i+1) = x2\_euler(i) + h \* dx2dt(v2\_euler(i));

v1\_euler(i+1) = v1\_euler(i) + h \* dv1dt(x1\_euler(i), x2\_euler(i));

v2\_euler(i+1) = v2\_euler(i) + h \* dv2dt(x1\_euler(i), x2\_euler(i));

Ea\_x1\_euler(i)=abs((x1\_euler(i+1)-x1\_euler(i))/x1\_euler(i+1)\*100);

Ea\_x2\_euler(i)=abs((x2\_euler(i+1)-x2\_euler(i))/x2\_euler(i+1)\*100);

Ea\_v1\_euler(i)=abs((v1\_euler(i+1)-v1\_euler(i))/v1\_euler(i+1)\*100);

Ea\_v2\_euler(i)=abs((v2\_euler(i+1)-v2\_euler(i))/v2\_euler(i+1)\*100);

end

% Heun's Method

for i = 1:n\_steps-1

% Predictor step

x1\_pred = x1\_heun(i) + h \* dx1dt(v1\_heun(i));

x2\_pred = x2\_heun(i) + h \* dx2dt(v2\_heun(i));

v1\_pred = v1\_heun(i) + h \* dv1dt(x1\_heun(i), x2\_heun(i));

v2\_pred = v2\_heun(i) + h \* dv2dt(x1\_heun(i), x2\_heun(i));

% Corrector step

x1\_heun(i+1) = x1\_heun(i) + (h/2) \* (dx1dt(v1\_heun(i)) + dx1dt(v1\_pred));

x2\_heun(i+1) = x2\_heun(i) + (h/2) \* (dx2dt(v2\_heun(i)) + dx2dt(v2\_pred));

v1\_heun(i+1) = v1\_heun(i) + (h/2) \* (dv1dt(x1\_heun(i), x2\_heun(i)) + dv1dt(x1\_pred, x2\_pred));

v2\_heun(i+1) = v2\_heun(i) + (h/2) \* (dv2dt(x1\_heun(i), x2\_heun(i)) + dv2dt(x1\_pred, x2\_pred));

Ea\_x1\_heun(i)=abs((x1\_heun(i+1)-x1\_heun(i))/x1\_heun(i+1)\*100);

Ea\_x2\_heun(i)=abs((x2\_heun(i+1)-x2\_heun(i))/x2\_heun(i+1)\*100);

Ea\_v1\_heun(i)=abs((v1\_heun(i+1)-v1\_heun(i))/v1\_heun(i+1)\*100);

Ea\_v2\_heun(i)=abs((v2\_heun(i+1)-v2\_heun(i))/v2\_heun(i+1)\*100);

end

% Runge-Kutta Method (4th order)

for i = 1:n\_steps-1

k1x1 = h \* dx1dt(v1\_rk4(i));

k1x2 = h \* dx2dt(v2\_rk4(i));

k1v1 = h \* dv1dt(x1\_rk4(i), x2\_rk4(i));

k1v2 = h \* dv2dt(x1\_rk4(i), x2\_rk4(i));

k2x1 = h \* dx1dt(v1\_rk4(i) + k1v1/2);

k2x2 = h \* dx2dt(v2\_rk4(i) + k1v2/2);

k2v1 = h \* dv1dt(x1\_rk4(i) + k1x1/2, x2\_rk4(i) + k1x2/2);

k2v2 = h \* dv2dt(x1\_rk4(i) + k1x1/2, x2\_rk4(i) + k1x2/2);

k3x1 = h \* dx1dt(v1\_rk4(i) + k2v1/2);

k3x2 = h \* dx2dt(v2\_rk4(i) + k2v2/2);

k3v1 = h \* dv1dt(x1\_rk4(i) + k2x1/2, x2\_rk4(i) + k2x2/2);

k3v2 = h \* dv2dt(x1\_rk4(i) + k2x1/2, x2\_rk4(i) + k2x2/2);

k4x1 = h \* dx1dt(v1\_rk4(i) + k3v1);

k4x2 = h \* dx2dt(v2\_rk4(i) + k3v2);

k4v1 = h \* dv1dt(x1\_rk4(i) + k3x1, x2\_rk4(i) + k3x2);

k4v2 = h \* dv2dt(x1\_rk4(i) + k3x1, x2\_rk4(i) + k3x2);

% Update values

x1\_rk4(i+1) = x1\_rk4(i) + (k1x1 + 2\*k2x1 + 2\*k3x1 + k4x1)/6;

x2\_rk4(i+1) = x2\_rk4(i) + (k1x2 + 2\*k2x2 + 2\*k3x2 + k4x2)/6;

v1\_rk4(i+1) = v1\_rk4(i) + (k1v1 + 2\*k2v1 + 2\*k3v1 + k4v1)/6;

v2\_rk4(i+1) = v2\_rk4(i) + (k1v2 + 2\*k2v2 + 2\*k3v2 + k4v2)/6;

Ea\_x1\_rk4(i)=abs((x1\_rk4(i+1)-x1\_rk4(i))/x1\_rk4(i+1)\*100);

Ea\_x2\_rk4(i)=abs((x2\_rk4(i+1)-x2\_rk4(i))/x2\_rk4(i+1)\*100);

Ea\_v1\_rk4(i)=abs((v1\_rk4(i+1)-v1\_rk4(i))/v1\_rk4(i+1)\*100);

Ea\_v2\_rk4(i)=abs((v2\_rk4(i+1)-v2\_rk4(i))/v2\_rk4(i+1)\*100);

end

% Display results

disp('Comparison of Methods: Final Displacements and Velocities');

disp('-------------------------------------------------------');

disp('Method | x1 (m) | x2 (m) | v1 (m/s) | v2 (m/s)');

disp('-------------------------------------------------------');

fprintf('Euler | %.4f | %.4f | %.4f | %.4f\n', ...

x1\_euler(end), x2\_euler(end), v1\_euler(end), v2\_euler(end));

fprintf('Heun | %.4f | %.4f | %.4f | %.4f\n', ...

x1\_heun(end), x2\_heun(end), v1\_heun(end), v2\_heun(end));

fprintf('Runge-Kutta | %.4f | %.4f | %.4f | %.4f\n', ...

x1\_rk4(end), x2\_rk4(end), v1\_rk4(end), v2\_rk4(end));

disp('-------------------------------------------------------');

% Display the approximate errors directly for each method

disp('----------------------------------------------------------');

disp('Approximate Errors for Each Method (at Final Time Step)');

disp('----------------------------------------------------------');

disp('Method | Ea(x1) | Ea(x2) | Ea(v1) | Ea(v2)');

disp('----------------------------------------------------------');

% Euler Method errors

fprintf('Euler | %.4f%% | %.4f%% | %.4f%% | %.4f%%\n', ...

Ea\_x1\_euler(end), Ea\_x2\_euler(end), Ea\_v1\_euler(end), Ea\_v2\_euler(end));

% Heun Method errors

fprintf('Heun | %.4f%% | %.4f%% | %.4f%% | %.4f%%\n', ...

Ea\_x1\_heun(end), Ea\_x2\_heun(end), Ea\_v1\_heun(end), Ea\_v2\_heun(end));

% Runge-Kutta Method errors

fprintf('Runge-Kutta | %.4f%% | %.4f%% | %.4f%% | %.4f%%\n', ...

Ea\_x1\_rk4(end), Ea\_x2\_rk4(end), Ea\_v1\_rk4(end), Ea\_v2\_rk4(end));

disp('----------------------------------------------------------');

% Plot results

% Heun's graph overlaps with Runge-Kutta due to very close results

figure;

subplot(2, 1, 1);

plot(t0, x1\_euler, 'r', t0, x1\_heun, 'g', t0, x1\_rk4, 'b');

xlabel('Time (s)'); ylabel('x1 (m)');

legend('Euler', 'Heun', 'RK4');

title('Displacement of Mass 1');

grid on;

subplot(2, 1, 2);

plot(t0, x2\_euler, 'r', t0, x2\_heun, 'g', t0, x2\_rk4, 'b');

xlabel('Time (s)'); ylabel('x2 (m)');

legend('Euler', 'Heun', 'RK4');

title('Displacement of Mass 2');

grid on;

figure;

subplot(2, 1, 1);

plot(t0, v1\_euler, 'r', t0, v1\_heun, 'g', t0, v1\_rk4, 'b');

xlabel('Time (s)'); ylabel('v1 (m/s)');

legend('Euler', 'Heun', 'RK4');

title('Velocity of Mass 1');

grid on;

subplot(2, 1, 2);

plot(t0, v2\_euler, 'r', t0, v2\_heun, 'g', t0, v2\_rk4, 'b');

xlabel('Time (s)'); ylabel('v2 (m/s)');

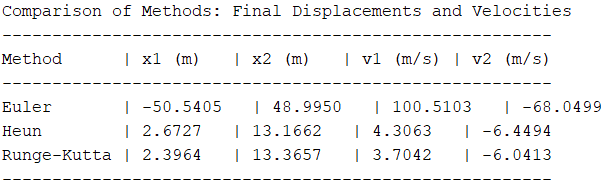
legend('Euler', 'Heun', 'RK4');

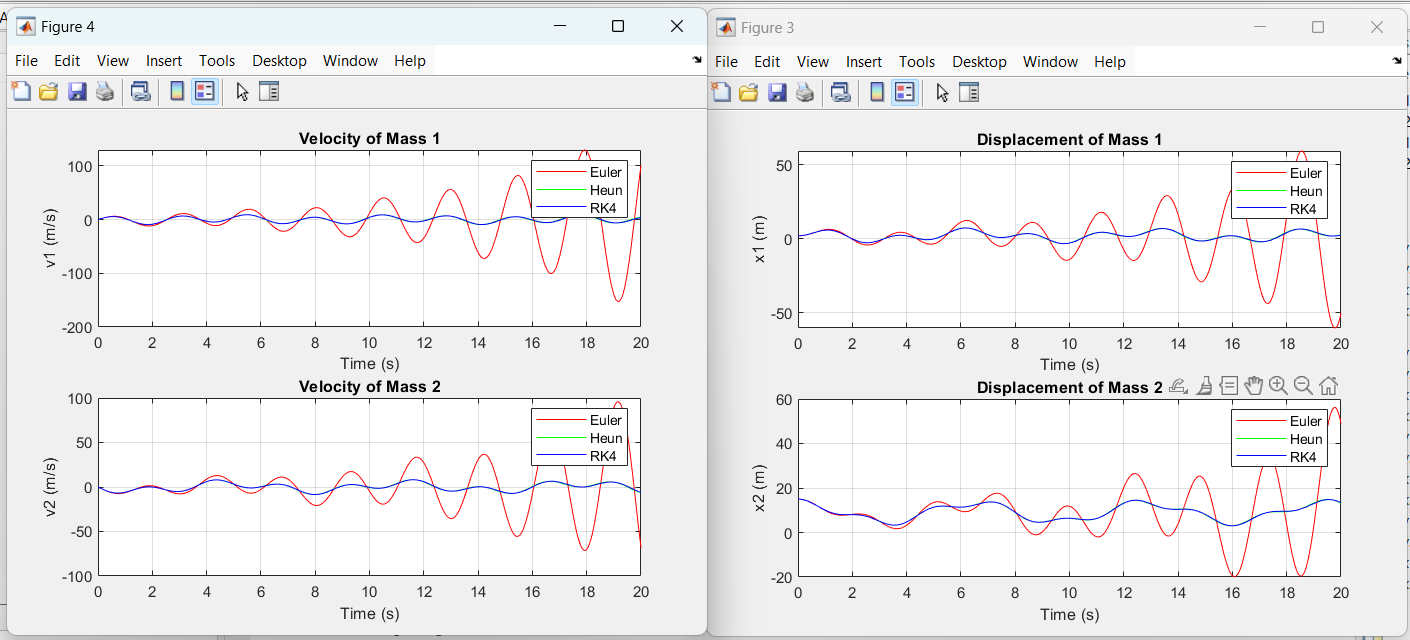
title('Velocity of Mass 2');

grid on;

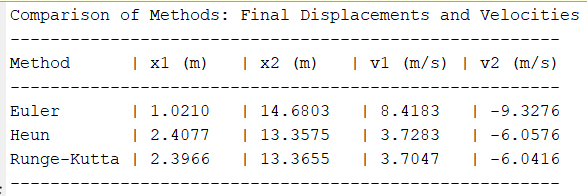
# Comparison

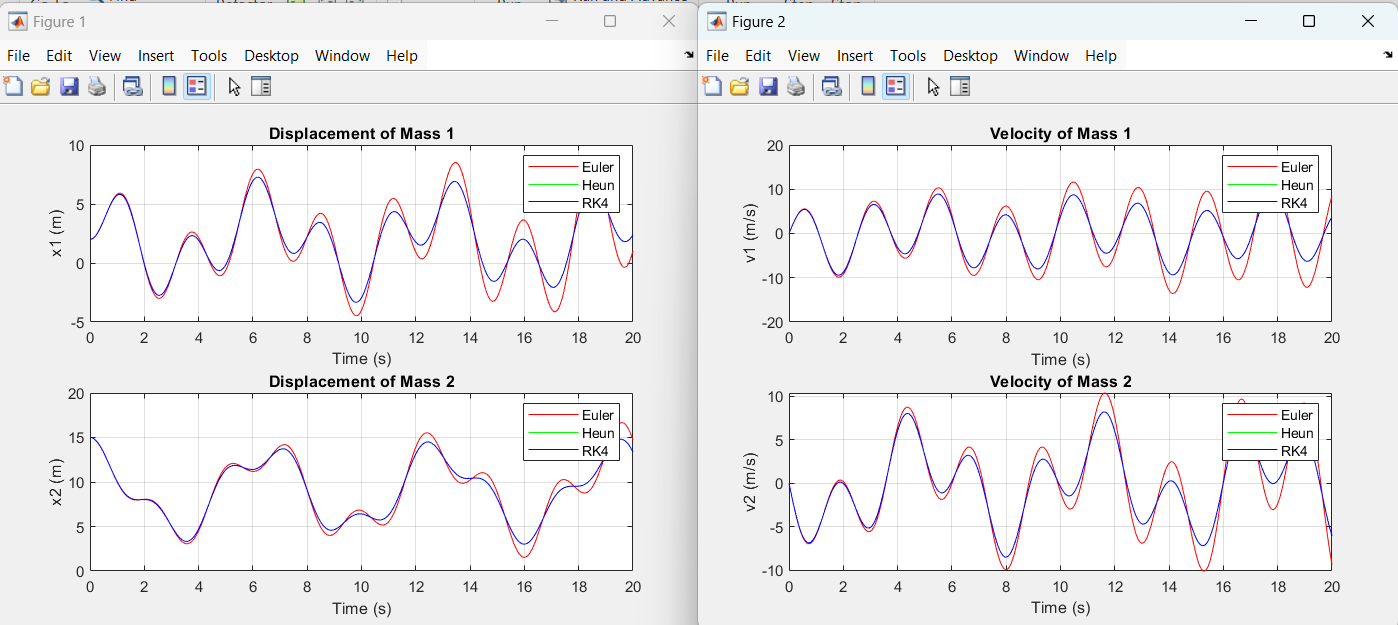
**-Using h=0.05**





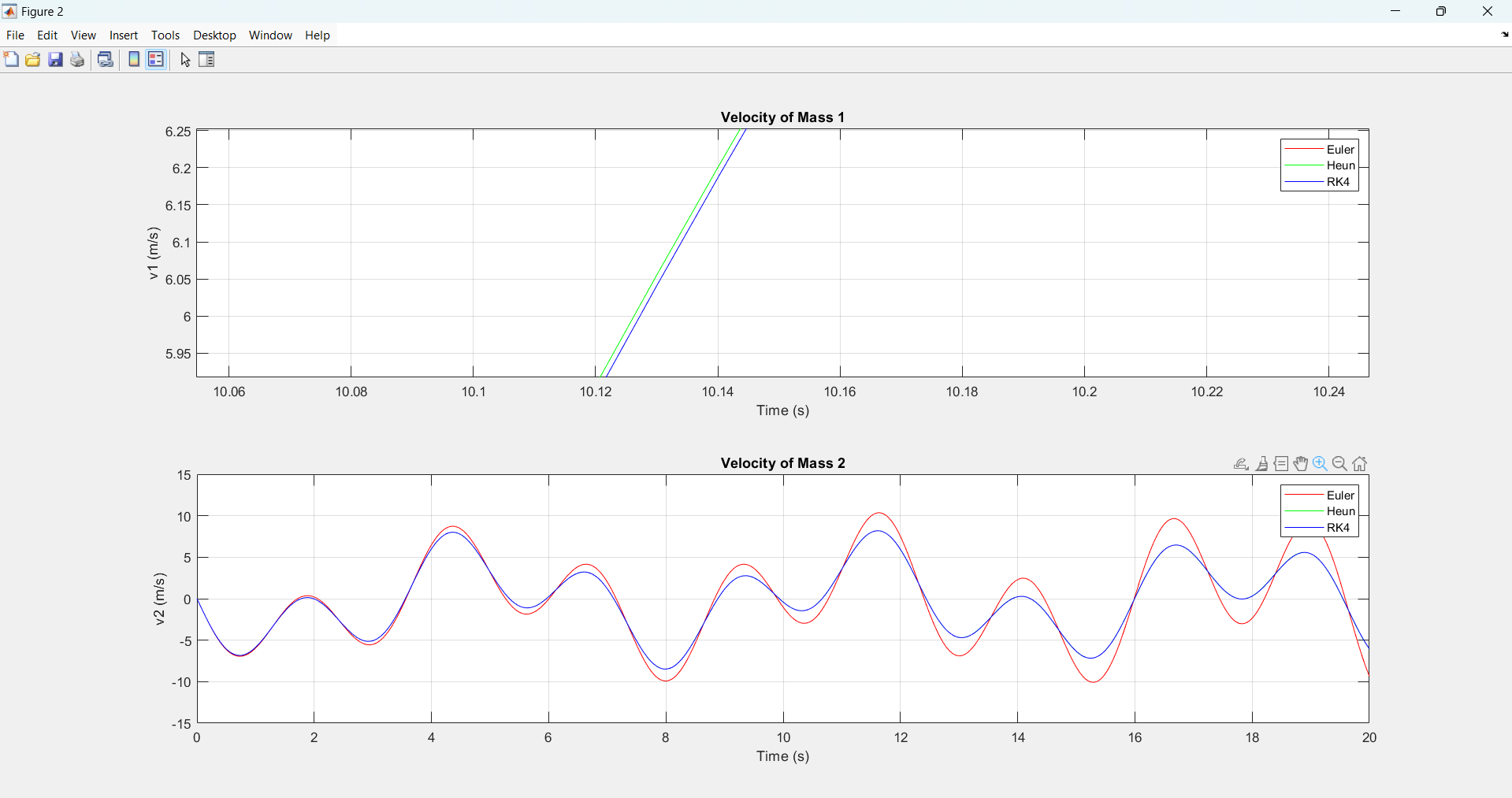
**-Using h=0.01**





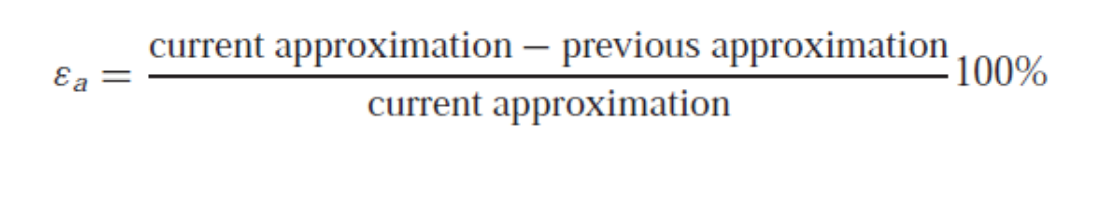
Comment:

* The smaller the step size, the closer the results are from each other.
* Heun’s method and Runge-Kutta’s method overlap each other on the figures as their results are too close from each other.



# Errors

To calculate the approximate error for each method:



Approximate error for each parameter in tabular form:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | X1 | X2 | V1 | V2 |
| Euler | 8.05% | 0.62% | 2.34% | 1.83% |
| Heun | 1.53% | 0.45% | 2.42% | 1.65% |
| Runge-kutta | 1.53% | 0.45% | 2.42% | 1.66% |

**Analysis of Results:**

1. **Euler’s Method**:

The errors obtained with Euler’s method are significantly higher compared to the other methods. For example, the error in the displacement of mass 1 (X1) is 8.05%, indicating a substantial deviation from the true value. Similarly, the errors in velocities (V1 and V2) are around 2-3%, suggesting that Euler’s method is less accurate in predicting the velocities over time.

This can be attributed to Euler’s method being a first-order numerical technique. While it is computationally simple and efficient, its accuracy suffers, particularly for larger time steps or systems with more complex behavior.

1. **Heun’s Method**:

Heun’s method shows marked improvement over Euler’s. The error in X1 is reduced to 1.53%, and the error in X2 is even lower at 0.45%. These reductions in error highlight Heun’s ability to provide more accurate predictions for the displacement of both masses.

The errors in velocities are also noticeably smaller, although still higher than those for displacement. Heun’s method is a second-order technique, meaning it refines the Euler method by incorporating a better approximation of the system's dynamics. This results in significantly improved accuracy, particularly in systems like this one.

1. **Runge-Kutta (4th Order)**:

The results for Runge-Kutta are almost identical to those of Heun’s method, with errors in X1 and X2 being 1.53% and 0.45%, respectively. The velocities show very similar errors as well.

# Conclusion

In this project, we analyzed the dynamics of a two-mass spring-damper system, simulating the displacements and velocities of the masses under varying numerical integration methods: Euler, Heun, and Runge-Kutta. Using the defined system parameters, we evaluated the performance of these methods in terms of accuracy and stability over a simulation time of 20 seconds.

Initially, we attempted to solve the system analytically by deriving closed-form solutions for the differential equations. However, the coupled nature of the equations and the complexity of the system made analytical solutions impractical. This highlighted the necessity of employing numerical methods to approximate the system's behavior effectively.

From the displacement and velocity plots, it is evident that the RK4 method provides the most accurate and stable results, closely approximating the expected physical behavior of the system. The Heun method also shows good accuracy but deviates slightly in comparison, particularly for the final velocities. The Euler method, being the simplest, exhibits significant deviations, especially in predicting the system's velocity, indicating numerical errors accumulating over time due to its lower-order approximation.

The tabulated results further emphasize the differences in accuracy among the methods. Runge-Kutta consistently yields results closer to the Heun method, reinforcing its reliability for applications requiring high precision, such as train suspension systems.

This study demonstrates that while simpler numerical methods like Euler can provide insights into system dynamics, higher-order methods such as Runge-Kutta are essential for accurately capturing the complex interactions in a spring-mass system. These findings are significant for optimizing suspension systems in transportation engineering, where accurate modeling of displacement and velocity responses is critical for safety and performance.