Jaypee Institute of Information Technology, Sector - 62, Noida

B.Tech CSE I Semester

Mathematics PBL Report

Harmonic Oscillators Using Differential Equations

Submitted to

Dr. Nisha Shukla

Submitted by

Harsh Sharma	2401030232
Karvy Singh	2401030234
Kunal Sharma	2401030236
Rudra Kumar Singh	2401030237
Mukul Aggarwal	2401030239

Letter of Transmittal

Dr. Nisha Shukla

Department of Mathematics

Subject: Submission of Report on "Harmonic Oscillators Using Differential Equations"

Dear Dr. Nisha Shukla,

We are pleased to submit our report titled *"Harmonic Oscillators Using Differential Equations"* as part of our coursework. This report explores the mathematical foundations of harmonic oscillators, focusing on differential equations and their applications in modeling simple harmonic motion, damped harmonic motion, and electrical analogs like RLC circuits.

We have endeavored to cover the theoretical aspects comprehensively and hope that this report meets your expectations.

Thank you for your guidance and the opportunity to work on this project.

Sincerely,

Harsh Sharma (2401030232)
Karvy Singh (2401030234)
Kunal Sharma (2401030236)
Rudra Kumar Singh (2401030237)
Mukul Aggarwal (2401030239)

Date: November 23, 2024

Contents

1 Introduction

Harmonic oscillators are a cornerstone in the study of differential equations and mathematical physics. They model systems where a restoring force is proportional to the displacement from equilibrium, leading to periodic motion. Understanding harmonic oscillators through differential equations allows for the analysis of a wide range of phenomena in engineering, physics, and applied mathematics. This report explores the mathematical underpinnings of harmonic oscillators, emphasizing differential equations, general solutions, and applications such as simple harmonic motion, damped harmonic motion, and RLC circuits.

2 Mathematical Foundation of Harmonic Oscillations

2.1 The Universal Oscillator Equation

The universal form of a second-order linear homogeneous differential equation describing harmonic oscillations is:

$$\frac{d^2x}{dt^2} + 2\beta\frac{dx}{dt} + \omega_0^2 x = 0$$

where:

- $x = x(t)$ is the displacement as a function of time.
- $\frac{dx}{dt}$ and $\frac{d^2x}{dt^2}$ denote the first and second derivatives of x with respect to time.
- $\beta \geq 0$ is the damping coefficient.
- $\omega_0 > 0$ is the natural angular frequency of the system.

This equation encapsulates the behavior of a wide variety of oscillatory systems, including mechanical and electrical oscillators.

2.2 General Solutions

The solution to the universal oscillator equation depends on the discriminant $D = \beta^2 - \omega_0^2$:

- **Case 1: Underdamped ($\beta^2 < \omega_0^2$)**

  The system exhibits oscillatory behavior with an exponentially decaying amplitude. The general solution is:

  $$x(t) = e^{-\beta t}\left(A\cos(\omega_d t) + B\sin(\omega_d t)\right)$$

  where $\omega_d = \sqrt{\omega_0^2 - \beta^2}$ is the damped angular frequency.

- **Case 2: Critically Damped ($\beta^2 = \omega_0^2$)**

  The system returns to equilibrium without oscillating. The general solution is:

  $$x(t) = (A + Bt)e^{-\beta t}$$

- **Case 3: Overdamped ($\beta^2 > \omega_0^2$)**

  The system returns to equilibrium without oscillating, and more slowly than in the critically damped case. The general solution is:

$$x(t) = e^{-\beta t}\left(Ce^{\gamma t} + De^{-\gamma t}\right)$$

  where $\gamma = \sqrt{\beta^2 - \omega_0^2}$.

2.3 Eigenvalues and Characteristic Equations

To solve the universal oscillator equation, we use the characteristic equation derived by assuming solutions of the form $x(t) = e^{rt}$:

$$r^2 + 2\beta r + \omega_0^2 = 0$$

Solving this quadratic equation yields the eigenvalues r, which determine the behavior of the solution:

$$r = -\beta \pm \sqrt{\beta^2 - \omega_0^2}$$

The nature of the roots (real and distinct, real and repeated, or complex conjugates) corresponds to the overdamped, critically damped, and underdamped cases, respectively.

3 Simple Harmonic Motion

3.1 Formulation of SHM as a Differential Equation

Simple Harmonic Motion (SHM) is a special case of the universal oscillator equation with no damping ($\beta = 0$):

$$\frac{d^2x}{dt^2} + \omega_0^2 x = 0$$

This second-order linear homogeneous differential equation describes systems where the restoring force is directly proportional to the displacement and acts in the opposite direction.

3.2 Solutions to the SHM Equation

The characteristic equation is:

$$r^2 + \omega_0^2 = 0$$

with roots:

$$r = \pm i\omega_0$$

The general solution is:

$$x(t) = A\cos(\omega_0 t) + B\sin(\omega_0 t)$$

Alternatively, using a single trigonometric function:

$$x(t) = X_0 \cos(\omega_0 t + \phi)$$

where:

- $X_0 = \sqrt{A^2 + B^2}$ is the amplitude.
- $\phi = \tan^{-1}\left(\frac{B}{A}\right)$ is the phase angle.

3.3 Mathematical Examples of SHM

Example 1: Solving the SHM Equation

Consider the differential equation:

$$\frac{d^2x}{dt^2} + 9x = 0$$

- The natural frequency is $\omega_0 = 3$ rad/s.
- The general solution is $x(t) = A\cos(3t) + B\sin(3t)$.

Given initial conditions $x(0) = 2$ and $\frac{dx}{dt}(0) = 0$:

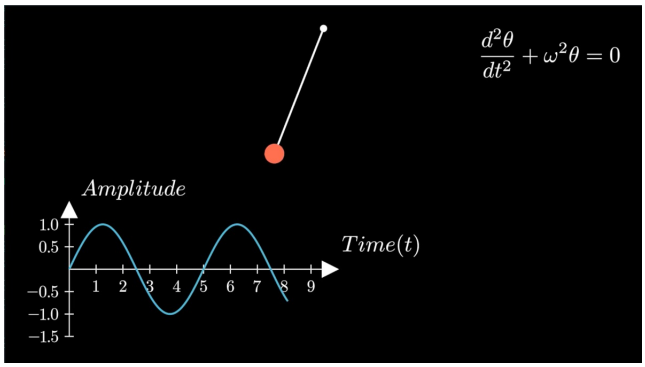
$$x(0) = A = 2$$
$$\frac{dx}{dt}(0) = -3A\sin(0) + 3B\cos(0) = 3B = 0 \implies B = 0$$

Thus, the particular solution is:

$$x(t) = 2\cos(3t)$$

3.4 Simulation

As we saw above, Simple Harmonic Motion is represented by a 2nd order differential equation, which on rendering gives a sine graph between amplitude and time.



Demo of SHM made using python

4 Damped Harmonic Motion

4.1 Formulation of Damped Harmonic Oscillator

Including damping, the differential equation becomes:

$$\frac{d^2x}{dt^2} + 2\beta\frac{dx}{dt} + \omega_0^2 x = 0$$

This equation models systems where energy is lost over time due to non-conservative forces like friction or air resistance.

4.2 Classification Based on Damping Ratio

Define the damping ratio $\zeta = \dfrac{\beta}{\omega_0}$:

- **Underdamped** ($\zeta < 1$): Oscillatory motion with exponential decay.
- **Critically Damped** ($\zeta = 1$): Non-oscillatory motion returning to equilibrium as quickly as possible.
- **Overdamped** ($\zeta > 1$): Non-oscillatory motion returning to equilibrium slower than the critically damped case.

4.3 Analytical Solutions

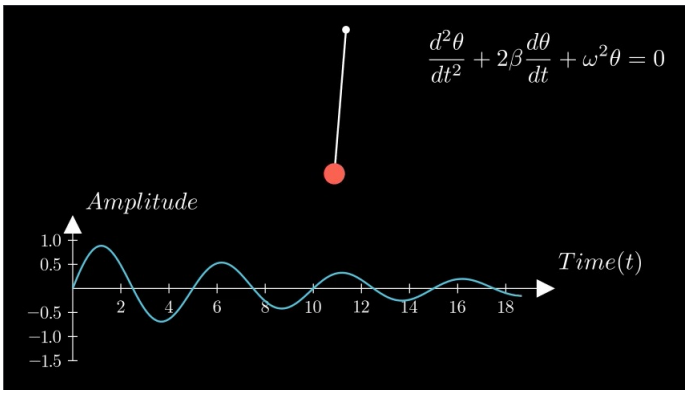
Refer to Section 2.2 for the general solutions in each damping case. Solving specific problems involves applying initial conditions to determine the constants.

Example: Underdamped Oscillator
Given $\frac{d^2x}{dt^2} + 4\frac{dx}{dt} + 5x = 0$:

- $\beta = 2$, $\omega_0 = \sqrt{5}$.
- Damping ratio $\zeta = \dfrac{2}{\sqrt{5}} < 1$ (underdamped).
- Damped frequency $\omega_d = \sqrt{\omega_0^2 - \beta^2} = 1$.
- General solution: $x(t) = e^{-2t}(A\cos t + B\sin t)$.

4.4 Simulation

As we saw above, Damping of Oscillations is represented by a 2nd order differential equation, and rendering it gives us such graphical pattern between amplitude and time.



Demo of Damped Oscillations made using python

5 Electrical Analog: RLC Circuit

5.1 Mathematical Modeling of RLC Circuits

An RLC series circuit consisting of a resistor (R), inductor (L), and capacitor (C) can be modeled by a second-order linear differential equation. Applying Kirchhoff's voltage law:

$$L\frac{d^2q}{dt^2} + R\frac{dq}{dt} + \frac{1}{C}q = 0$$

where $q(t)$ is the charge on the capacitor at time t.

5.2 Differential Equation for RLC Circuits

Rewriting the equation:

$$\frac{d^2q}{dt^2} + 2\alpha\frac{dq}{dt} + \omega_0^2 q = 0$$

with:

- $\alpha = \dfrac{R}{2L}$ (damping coefficient).
- $\omega_0 = \dfrac{1}{\sqrt{LC}}$ (natural angular frequency).

This equation is analogous to the universal oscillator equation.

5.3 Solution Techniques and Examples

Example: Solving an RLC Circuit Differential Equation

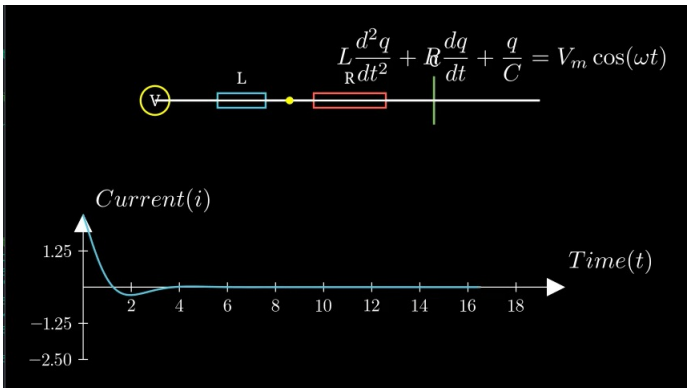
Given $L = 1\,\text{H}$, $R = 2\,\Omega$, $C = 0.25\,\text{F}$:

- $\alpha = \dfrac{2}{2 \times 1} = 1\,\mathrm{s}^{-1}$.
- $\omega_0 = \dfrac{1}{\sqrt{1 \times 0.25}} = 2\,\mathrm{s}^{-1}$.
- Damping ratio $\zeta = \dfrac{1}{2} < 1$ (underdamped).
- Damped frequency $\omega_d = \sqrt{4 - 1} = \sqrt{3}\,\mathrm{s}^{-1}$.
- General solution: $q(t) = e^{-t}\left(A\cos\left(\sqrt{3}\,t\right) + B\sin\left(\sqrt{3}\,t\right)\right)$.

Applying initial conditions allows for solving for constants A and B.

5.4 Simulation

As we saw above, damping of current in RLC circuit is the most common phenomenon studied in higher electronics, it is also represented by a 2nd order differential equation which on rendering gives such graph.



Demo of current damping in RLC circuits made using python

6 Conclusion

The study of harmonic oscillators through differential equations provides profound insights into the behavior of dynamic systems. By focusing on the mathematical formulations, solutions, and classifications of these differential equations, we gain a deeper understanding applicable across various fields such as mechanics, electronics, and beyond. The universal oscillator equation serves as a foundational tool in analyzing and predicting system responses under different damping conditions.

7 Source Code

Throughout the report, we have attached screenshots of simulations. These simulations were made using python and the Manim library. We have attached the source code of the simulations below:

7.1 Simple Harmonic Motion

```
from manim import *

class PendulumWithSineGraph(Scene):
    def construct(self):
        # Define pendulum components
        pendulum_pivot = UP * 3.5  # Shift pivot upwards
        pendulum_length = 3
        pendulum_radius = 0.2
        pendulum_angle_amplitude = PI / 6  # max angle (30 degrees)
        omega = 2 * PI / 5  # angular frequency (period = 5 seconds)

        # Time tracking
        self.time_tracker = ValueTracker(0)

        # Pendulum parts
        pivot_dot = Dot(pendulum_pivot, color=WHITE)
        pendulum_line = Line(pendulum_pivot, pendulum_pivot + DOWN *
        ↪  pendulum_length)
        pendulum_ball =
        ↪  Circle(radius=pendulum_radius).move_to(pendulum_line.get_end())
        pendulum_ball.set_fill(RED, opacity=1)

        pendulum_group = VGroup(pendulum_line, pendulum_ball)

        # Create dynamic sine graph
        axes = Axes(
            x_range=[0, 10, 1],
            y_range=[-1.5, 1.5, 0.5],
            x_length=6,
            y_length=3,
            axis_config={"include_numbers": True},
        ).to_corner(DOWN + LEFT)

        sine_label = axes.get_axis_labels(x_label="Time (t)",
        ↪  y_label="Amplitude")
        sine_graph = always_redraw(lambda: axes.plot(
            lambda t: np.sin(omega * t),
            x_range=[0, self.time_tracker.get_value()],
            color=BLUE
        ))
```

```
        # SHM equation text
        shm_eq = MathTex(r"\frac{d^2\theta}{dt^2} + \omega^2 \theta = 0")
        shm_eq.to_corner(UP + RIGHT)

        # Pendulum motion updater
        def update_pendulum(group):
            time = self.time_tracker.get_value()
            angle = pendulum_angle_amplitude * np.sin(omega * time)
            pendulum_line.put_start_and_end_on(
                pendulum_pivot,
                pendulum_pivot + pendulum_length *
                ↪  np.array([np.sin(angle), -np.cos(angle), 0])
            )
            pendulum_ball.move_to(pendulum_line.get_end())

        pendulum_group.add_updater(update_pendulum)

        # Add components to the scene
        self.add(pivot_dot, pendulum_group, axes, sine_label, shm_eq,
        ↪  sine_graph)

        # Animate the time tracker and the scene
        self.play(self.time_tracker.animate.set_value(10), run_time=10,
        ↪  rate_func=linear)

        # Stop pendulum motion after the animation
        pendulum_group.remove_updater(update_pendulum)
```

7.2 Damped Harmonic Motion

```
from manim import *

class DampedPendulumWithGraph(Scene):
    def construct(self):
        # Define pendulum components
        pendulum_pivot = UP * 3.5  # Shift pivot upwards
        pendulum_length = 3
        pendulum_radius = 0.2
        pendulum_angle_amplitude = PI / 6  # Max angle (30 degrees)
        omega = 2 * PI / 5  # Angular frequency (period = 5 seconds)
        damping_coefficient = 0.1  # Damping factor

        # Time tracking
        self.time_tracker = ValueTracker(0)

        # Pendulum parts
```

```
        pivot_dot = Dot(pendulum_pivot, color=WHITE)
        pendulum_line = Line(pendulum_pivot, pendulum_pivot + DOWN *
        ↪  pendulum_length)
        pendulum_ball =
        ↪  Circle(radius=pendulum_radius).move_to(pendulum_line.get_end())
        pendulum_ball.set_fill(RED, opacity=1)

        pendulum_group = VGroup(pendulum_line, pendulum_ball)

        # Create dynamic damped sine graph
        axes = Axes(
            x_range=[0, 20, 2],  # Extend x-range to cover 20 seconds
            y_range=[-1.5, 1.5, 0.5],
            x_length=10,
            y_length=3,
            axis_config={"include_numbers": True},
        ).to_corner(DOWN + LEFT)

        sine_label = axes.get_axis_labels(x_label="Time (t)",
        ↪  y_label="Amplitude")
        sine_graph = always_redraw(lambda: axes.plot(
            lambda t: np.exp(-damping_coefficient * t) * np.sin(omega *
            ↪  t),
            x_range=[0, self.time_tracker.get_value()],
            color=BLUE
        ))

        # Damped SHM equation text
        damped_shm_eq = MathTex(r"\frac{d^2\theta}{dt^2} +
        ↪  2\beta\frac{d\theta}{dt} + \omega^2 \theta = 0")
        damped_shm_eq.to_corner(UP + RIGHT)

        # Pendulum motion updater
        def update_pendulum(group):
            time = self.time_tracker.get_value()
            # Damped angle equation
            angle = pendulum_angle_amplitude *
            ↪  np.exp(-damping_coefficient * time) * np.sin(omega *
            ↪  time)
            pendulum_line.put_start_and_end_on(
                pendulum_pivot,
                pendulum_pivot + pendulum_length *
                ↪  np.array([np.sin(angle), -np.cos(angle), 0])
            )
            pendulum_ball.move_to(pendulum_line.get_end())

        pendulum_group.add_updater(update_pendulum)

```

```
        # Add components to the scene
        self.add(pivot_dot, pendulum_group, axes, sine_label,
        ↪  damped_shm_eq, sine_graph)

        # Animate the time tracker and the scene
        self.play(self.time_tracker.animate.set_value(20), run_time=20,
        ↪  rate_func=linear)  # Extend animation duration

        # Stop pendulum motion after the animation
        pendulum_group.remove_updater(update_pendulum)

```

7.3 RLC Circuits: Damped Currents

```
from manim import *

class LRCCircuitWithOscillation(Scene):
    def construct(self):
        # Circuit parameters
        L = 1.0  # Inductance in Henrys
        R = 2.0  # Resistance in Ohms
        C = 0.5  # Capacitance in Farads
        omega = 2 * PI / 5  # Angular frequency of AC source
        Vm = 5.0  # Maximum voltage of the source
        beta = R / (2 * L)  # Damping coefficient

        # Derived parameters
        Im = Vm / R  # Approximation of maximum current amplitude

        # Time tracking
        self.time_tracker = ValueTracker(0)

        # --- Circuit Diagram ---
        ac_source = Circle(radius=0.3, color=YELLOW).move_to(LEFT * 4)
        ac_label = Text("V",
        ↪  font_size=24).move_to(ac_source.get_center())

        inductor = Rectangle(height=0.3, width=1,
        ↪  color=BLUE).next_to(ac_source, RIGHT, buff=1)
        inductor_label = Text("L", font_size=24).next_to(inductor, UP,
        ↪  buff=0.2)

        resistor = Rectangle(height=0.3, width=1.5,
        ↪  color=RED).next_to(inductor, RIGHT, buff=1)
        resistor_label = Text("R", font_size=24).next_to(resistor, UP,
        ↪  buff=0.2)

        capacitor = VGroup(
```

```
            Line(ORIGIN, UP * 0.5, color=GREEN),
            Line(UP * 0.5, DOWN * 0.5, color=GREEN),
            Line(DOWN * 0.5, ORIGIN, color=GREEN)
        ).next_to(resistor, RIGHT, buff=1)
        capacitor_label = Text("C", font_size=24).next_to(capacitor, UP,
        ↪  buff=0.2)

        wire1 = Line(ac_source.get_center(), inductor.get_left(),
        ↪  color=WHITE)
        wire2 = Line(inductor.get_right(), resistor.get_left(),
        ↪  color=WHITE)
        wire3 = Line(resistor.get_right(), capacitor.get_left(),
        ↪  color=WHITE)
        wire4 = Line(capacitor.get_right(), RIGHT * 4, color=WHITE)
        wire5 = Line(RIGHT * 4, ac_source.get_center(), color=WHITE)

        circuit = VGroup(ac_source, ac_label, inductor, inductor_label,
        ↪  resistor, resistor_label,
                         capacitor, capacitor_label, wire1, wire2, wire3,
                         ↪  wire4, wire5).shift(UP * 2)

        # --- Oscillating Current Indicator ---
        current_indicator = Dot(color=YELLOW).move_to(wire2.get_center())

        def update_current_indicator(indicator):
            time = self.time_tracker.get_value()
            amplitude = Im * np.exp(-beta * time)
            displacement = amplitude * np.cos(omega * time)
            new_x = wire2.get_center()[0] + displacement * 0.5  # Scale
            ↪  for motion
            indicator.move_to([new_x, wire2.get_center()[1], 0])

        current_indicator.add_updater(update_current_indicator)

        # --- Dynamic Current Graph ---
        axes = Axes(
            x_range=[0, 20, 2],
            y_range=[-Im, Im, Im / 2],
            x_length=10,
            y_length=3,
            axis_config={"include_numbers": True},
        ).to_corner(DOWN + LEFT)

        graph_label = axes.get_axis_labels(x_label="Time (t)",
        ↪  y_label="Current (i)")

        def lrc_current(t):
            return Im * np.exp(-beta * t) * np.cos(omega * t)
```

```
70
71          current_graph = always_redraw(lambda: axes.plot(
72              lambda t: lrc_current(t),
73              x_range=[0, self.time_tracker.get_value()],
74              color=BLUE
75          ))
76
77          # LRC equation text
78          lrc_eq = MathTex(r"L\frac{d^2q}{dt^2} + R\frac{dq}{dt} +
            ↪  \frac{q}{C} = V_m \cos(\omega t)")
79          lrc_eq.to_corner(UP + RIGHT)
80
81          # Add all components to the scene
82          self.add(circuit, axes, graph_label, current_graph, lrc_eq,
            ↪  current_indicator)
83
84          # Animate the time tracker and the scene
85          self.play(self.time_tracker.animate.set_value(20), run_time=20,
            ↪  rate_func=linear)
86
87          # Stop updating and wait at the end
88          current_indicator.clear_updaters()
89          self.wait()
90
91
```

8 References

- Boyce, W. E., & DiPrima, R. C. (2017). *Elementary Differential Equations and Boundary Value Problems*. Wiley.
- Simmons, G. F. (2016). *Differential Equations with Applications and Historical Notes*. CRC Press.
- Braun, M. (1993). *Differential Equations and Their Applications*. Springer.