|  |  |
| --- | --- |
| Ex. No.: 1 | **Analog (op amp based) Simulation of Linear Differential Equation** |
| Date: |

**Aim**

To perform analog simulation of linear differential equation using opamp.

**Introduction**

There are two kinds of computers. A digital computer is most likely what comes to mind when you hear the word "computer." Numbers are represented by sets of 1's and 0's, where 1 and 0 are two separate voltages (usually +5V and 0V). Operations are either simple logical operations (such as AND, OR, and so on) or mathematical operations (such as addition or subtraction). Calculus-type operations are extremely difficult to perform. The analogue computer is the opposite form, in which numbers are represented by continually altering values. Voltage is the most commonly used quantity. The OPAMP circuits can be used for performing various mathematical operations with varying voltages. These circuits can be used in conjunction to solve a variety of differential equations.

**Procedure**

Step 1: Derive differential equation for the given system in figure 1

Step 2: Identify Suitable Op-Amp Circuit Configuration

Step 3: Implement the Circuit in MATLAB simscape

Step 4: Apply the Input and analyse the Output

Step 5: Consider Initial Conditions (Optional)

**Mechanical Translational System:**

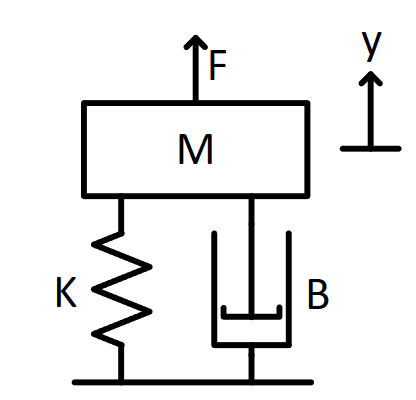
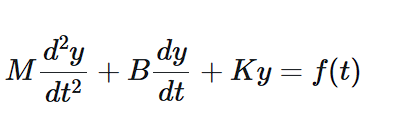
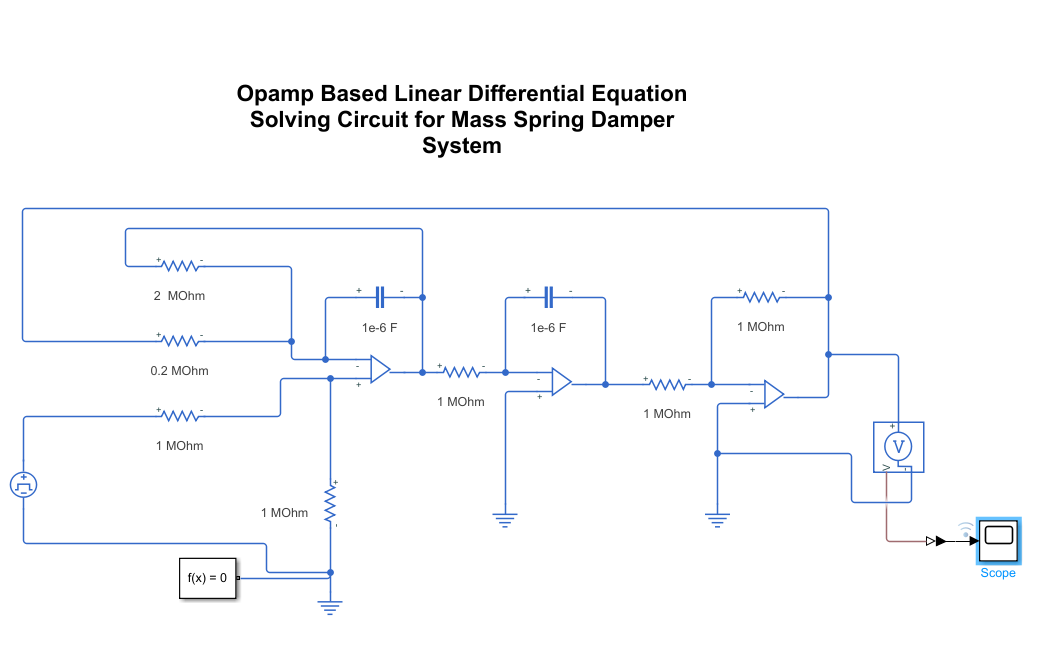


Figure Mass Spring Damper System

**Differential Equation for Mass Spring Damper System:**



**MATLAB Circuit**

****

**Output Waveform**

****

**Inference**

**Result**

The linear differential equation of a mass spring damper system is simulated using opamp based circuits using MATLAB simscape, and the system is tested for different inputs.

|  |  |
| --- | --- |
| Ex. No.: 2 | **Numerical Simulation of given Nonlinear Differential Equations** |
| Date: |

**Aim**

The objective of this lab experiment is to understand the process of numerically simulating nonlinear differential equations

**Introduction**

Differential equations play a crucial role in modeling real-world phenomena in physics, engineering, biology, and many other disciplines. Nonlinear differential equations are particularly challenging to solve analytically, so numerical methods like ode45 become invaluable in simulating their behavior.

ode45 is a powerful function in MATLAB used for solving ordinary differential equations (ODEs) numerically. It employs a Runge-Kutta method (specifically, the Dormand-Prince method) to approximate the solution of the ODE. The general syntax of ode45 is as follows:

[t, y] = ode45(@odefun, tspan, y0, options)

Where:

t is the vector of time points at which the solution is computed.

y is the matrix of solutions, where each row corresponds to the state of the system at the corresponding time in t.

odefun is a function handle that defines the ODEs you want to solve.

tspan is the time interval over which you want to solve the ODE. It is specified as [t0, tf], where t0 is the initial time and tf is the final time.

y0 is the initial condition of the system at time t0.

options (optional) is a structure that allows you to specify additional settings for ode45

Mass spring damper system with non linear stiffness

-----------------------------------(1)

**Procedure**

Step 1: Choose the non-linear differential equation representing the real-world physical system.

Step 2:

Open MATLAB and create a script file

Step3: Define the ODE function which represent the non-linear system

Step4: Set initial conditions and parameters

Step5: Solve the ODE using ode45 function

Step6: Plot the result displacement vs time

Step7: Experiment with different data and write the inference

**Matlab Code**

% Constants

m = 1; % Mass (kg)

c = 0.1; % Damping coefficient (Ns/m)

k = 1; % Linear stiffness coefficient (N/m)

alpha = 0.5; % Nonlinear stiffness coefficient (N/m^3)

% Initial conditions

x0 = 0.2; % initial displacement

v0 = 0.0; % initial velocity

% Time points for integration

t\_start = 0;

t\_end = 30;

dt = 0.01;

t\_points = t\_start:dt:t\_end;

% Numerical integration using MATLAB's ode45 solver

[t, X] = ode45(@(t, x) duffing\_oscillator(t, x, m, c, k, alpha), t\_points, [x0; v0]);

% Extract displacement values

x\_values = X(:, 1);

% Plotting the displacement waveform

figure;

plot(t, x\_values);

xlabel('Time (s)');

ylabel('Displacement (m)');

title('Duffing Oscillator - Sample Displacement Waveform');

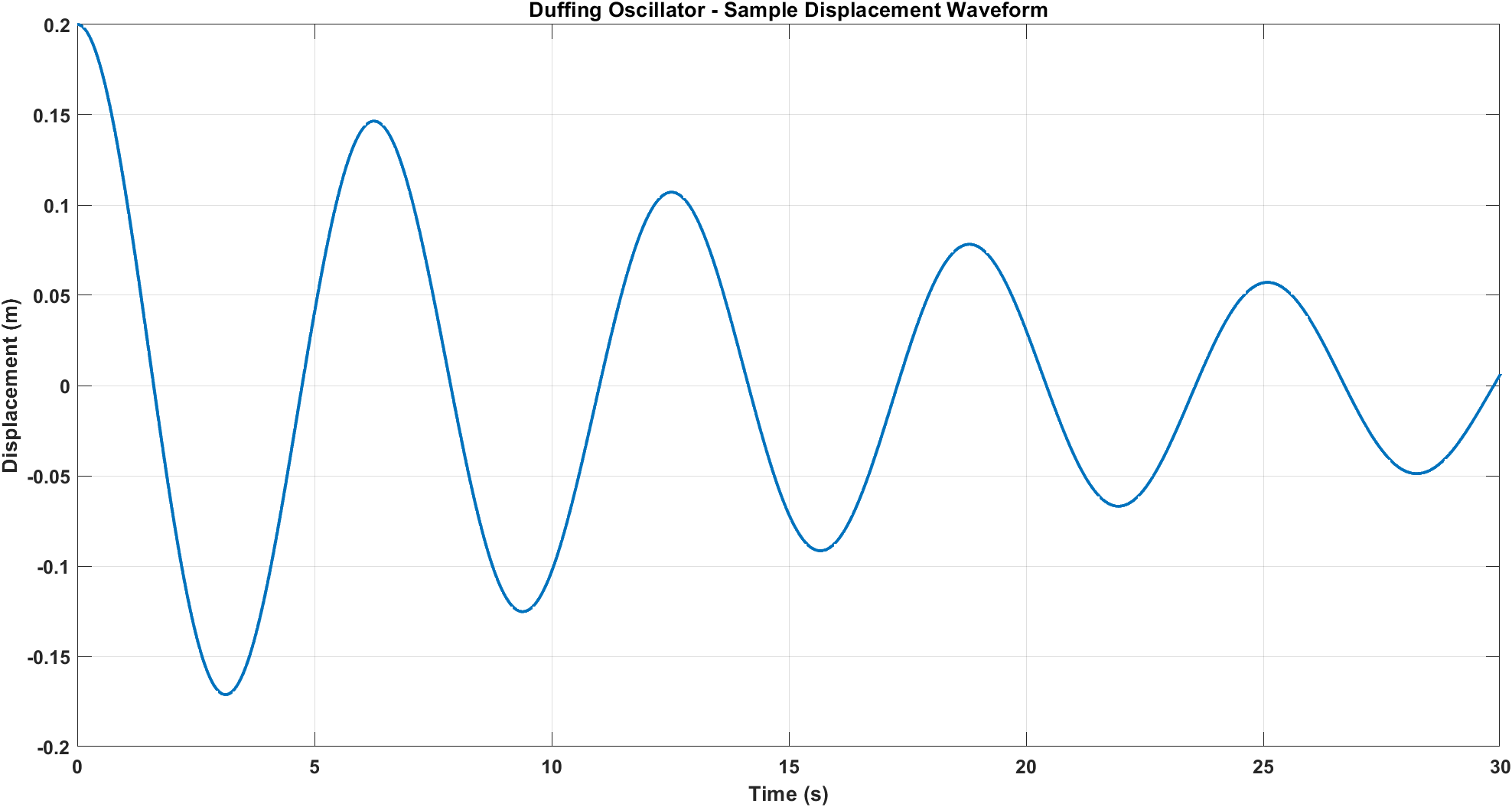
grid on;

function dxdt = duffing\_oscillator(~, x, m, c, k, alpha)

dxdt = [x(2); -(c\*x(2) + k\*x(1) + alpha\*x(1)^3) / m];

end

**Output Waveform:**



**Inference:**

**Result:**

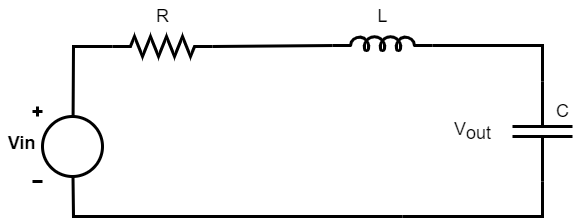
Thus, a non-linear differential equation is simulated numerically using MATLAB ‘ode45’ solver and the behaviour of oscillator is studied under different conditions.

|  |  |
| --- | --- |
| Ex. No.: 3 | **Realtime Simulation of Differential Equation** |
| Date: |

**Aim**

To simulate the differential equation of series RLC circuit using Matlab

**Introduction**

The series RLC circuit above has a single loop with the instantaneous current flowing through the loop being the same for each circuit element. Since the inductive and capacitive reactance’s XL and XC are a function of the supply frequency, the sinusoidal response of a series RLC circuit will therefore vary with frequency, ƒ. Then the individual voltage drops across each circuit element of R, L and C element will be “out-of-phase”

The differential equation for the above system for input and output voltage is given by equations 1 and 2

----------------------------------(1)

-----------------------------------------------(2)

The above equation can be represented in S domain using the equation 3.

--------------------------------(3)

**Procedure**

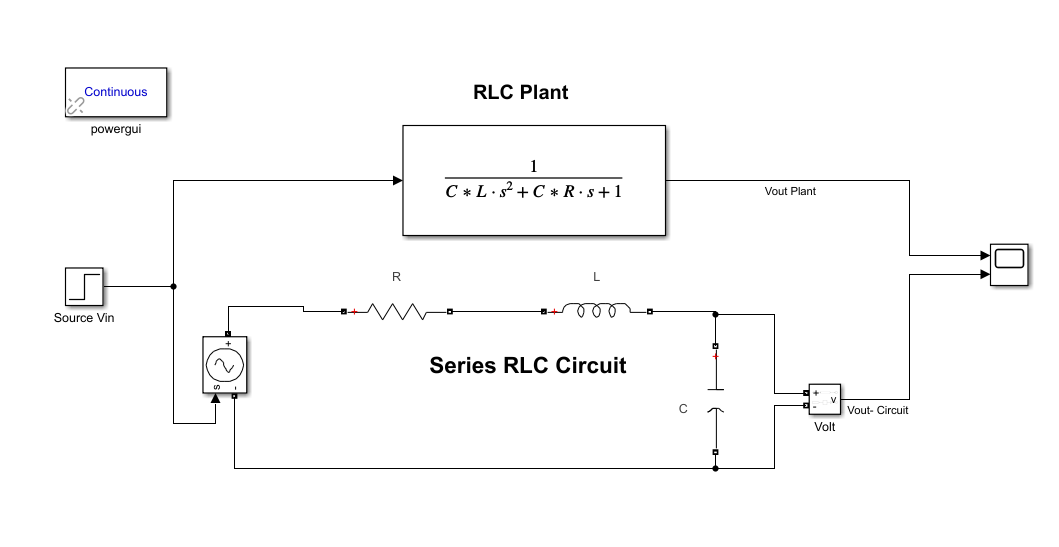
Step1: Input the transfer function obtained from differential equations in transfer function block

Step2: Apply a step input to the RLC plant

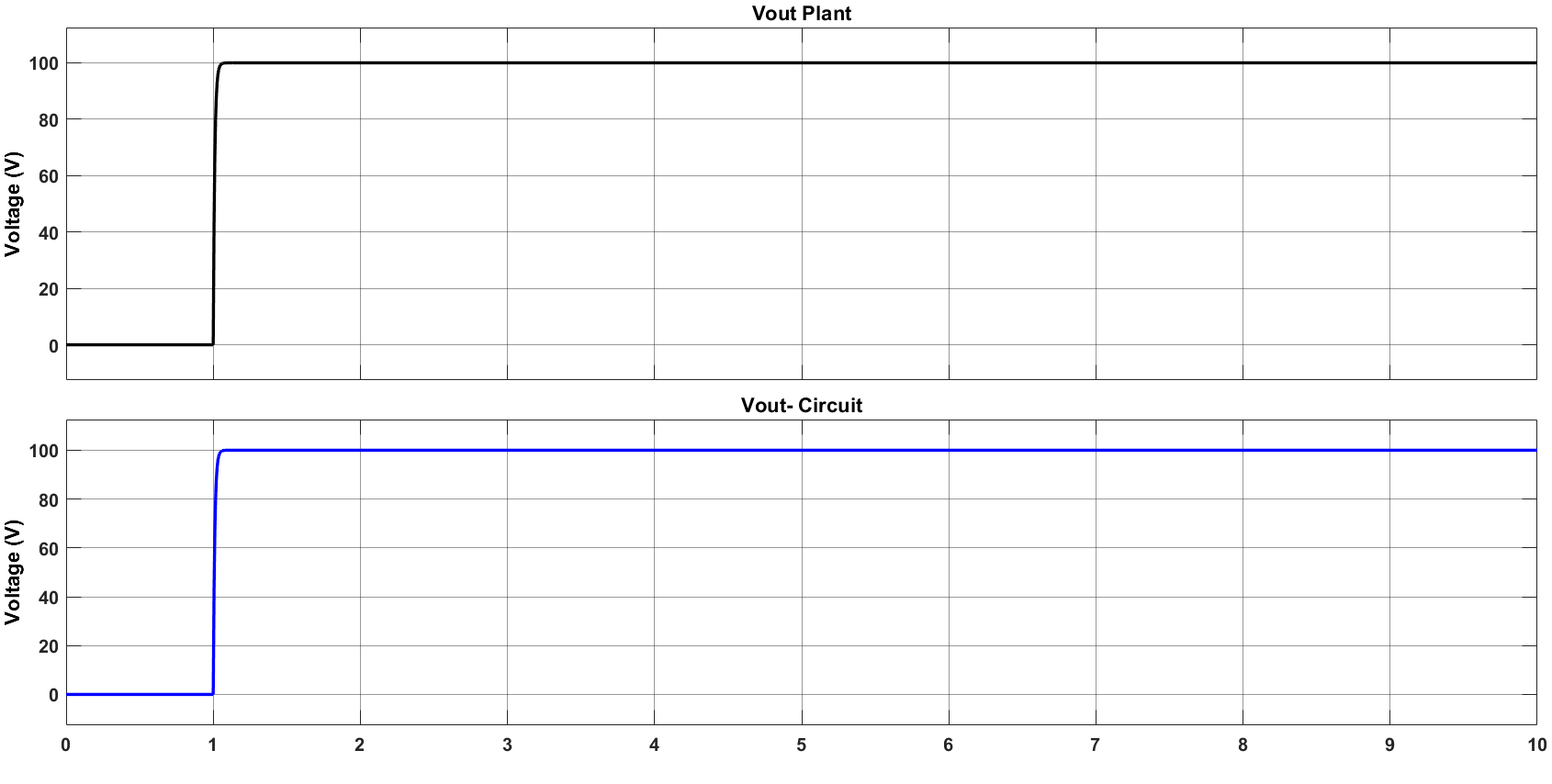
Step3: Plot the Vin and Vout waveform in scope.

Step4: Compare and verify the results simulated using RLC circuit in Simulink.

**Matlab Circuit**



**Output Waveform**

****

**Result**

Thus, the realtime differential equation circuit has been simulated using Simulink