TÉCNICO LISBOA

UNIVERSITY OF LISBON

INSTITUTO SUPERIOR TECNICO

Project work 1

Ship Vibration

Numerical solutions for the dynamic responses of simple system

Davide Melozzi

Ist1102230

MENO 2021/2022

INTRODUCTION

The objective of this project work is to calculate the dynamic responses of simple systems subjected to nonperiodic excitations by numerical solutions. A one-degree-of-freedom system with a viscous damping is considered. A code of the numerical methods is developed in MATLAB environment.

The mass analized is connected to two dampers and three springs. It is observed that the amplitude of vibration of the damped system decreased to 25% of its initial value after 5 consecutive cycles. Determine the response of the system under a force F(t), starting from rest.

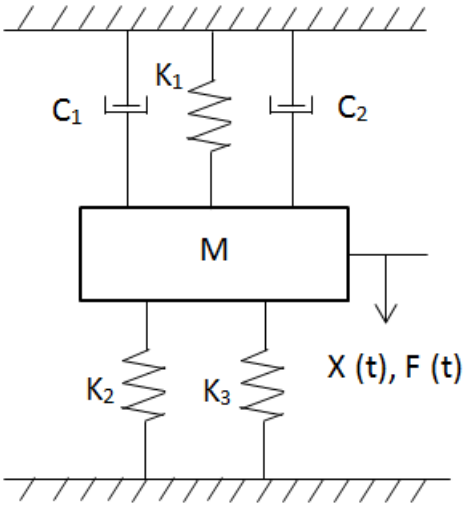


Fig.1 Representation of one degree system under viscous damping

Description of the calculations

1) Determine the differential equation of the motion of this system

- Calculate the equivalent mass, spring constant and damping constant of the system.

Initial data

K1=600 N/m
K2=200 N/m
K3=100 N/m
m=2.0 kg

The initial objective is to calculate the main quantities of the system, such as the equivalent mass, the spring constant and the damping constant. To do this, we start by solving the respective elements, obtaining unique equivalent quantities which are given by the arrangement of the components. C1 and C2 will be replaced by Ceq, which will be calculated later following the steps of a damped system in one degree of freedom, while the quantities K1,K2,K3 will be summed up to form Keq.

$$Keq = K1 + K2 + K3$$

Then, taking into account that in our system the amplitude response decreases by 25% after 5 consecutive cycles, in order to continue the calculation procedure of our variables we have to consider the logarithmic decreasing function, which considers the logarithmic decreasing trend according to the initial and final position of the amplitude and then allows us to calculate the damping factor.

Logarithmic decreasing:

$$\delta = \frac{1}{n} \ln \frac{xi}{xi + n} = 0{,}277$$

Where n is equal to 5 cycles and xi/xi+n are related by the 25% of decreasing amplitude, so $x_1=4x_2$

Next, we will calculate the damping factor:

$$\xi = \frac{\delta}{\sqrt{(2\pi)^2 + \delta^2}} = 0{,}044 \quad with\ 0 < \xi < 1$$

Having the dumping factor it is now possible to calculate the natural frequency and then the frequency related to the effect of damping.

$$\omega n = \sqrt{\frac{Keq}{Meq}} = 21{,}2\,\frac{rad}{s} \qquad having\ Meq = M = initial\ value$$

$$\omega d = \sqrt{1\omega - \xi^2} * \omega n = 21{,}18\,rad/s$$

As a last step, we obtain by means of the damping coefficient formula, the value of Ceq, which is mainly given by the type of system and subject forces we have taken into consideration.

$$Ceq = 2 * \xi * Meq * \omega n = 3{,}73\,\frac{N}{m} * s$$

MATLAB CODE DAMPING COEFICIENT:

```
function C = get_damping_coefficient(A_ratio, m, k, n)
    wn = sqrt(k / m); % Natural frequency
    sigma = log(A_ratio) / n;
    zeta_ = sigma / sqrt((2 * pi)^2 + (sigma)^2);
    C = 2 * m * wn * zeta_; % Damping Constant | Kilograms per Second
end
```

- Free body diagram

it is possible to see the differences in the distribution of compression/tension forces, the weight force of the mass, the responses of the dampers and springs and above all to understand the direction of the displacement inherent in the system in dynamic equilibrium.

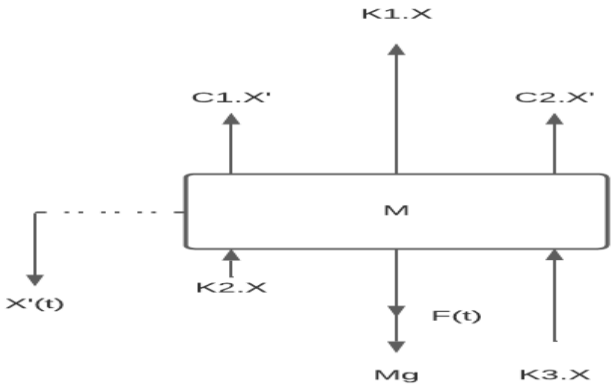


Fig.2 Free body diagram

- Using the second Newton's law, derive the differential equation of the motion

Let us consider the equation of motion in which all the characteristic variables of the system, inherent to acceleration, velocity and displacement, have been calculated.

To solve the equation of motion numerically, a time step (dt) of 0.005 seconds was chosen (in order to generate the highest possible precision without incurring floating point arithmetic errors), and it was decided to analyze the time interval of time from 0 to 2 seconds in order to clearly visualize the motion without taking an excessively long time to execute the programs.

By placing the equation equal to the represented total force of the system in response to time F(t), it is possible to derive the differential equation of motion in its forms, considering that since the function of the system F(t) is a time-dependent function, the representation can only occur for certain time intervals, in which the force function approximates to Heaviside functions H(t).

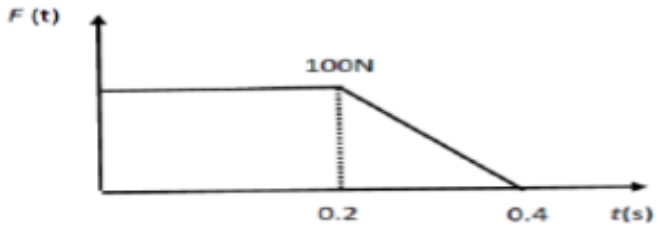


Fig.3 Applied force in relation of time of the system

The differential equation of mass motion related to our system is:

$$MX''(t) + (C1 + C2)X'(t) + (K1 + K2 + K3)X(t) = F(t)$$

Where time-dependent force F(t) is defined as:

$$F(t) = \begin{cases} 200 & \text{if } t < 0.2s \\ -500t + 300 & \text{if } 0.2s \leq t \leq 0.4s \\ 0 & t > 0.4s \end{cases}$$

$$F(t) = \begin{cases} 1000.t & \text{for } 0 \leq t \leq 0.2 \\ 0 & \text{for } t > 0.2 \end{cases}$$

Where we have to consider two corelated function that permit us to derive the differential equation of the motion considering the function F(t):

Laplace transform is a technique/tool for solving complex differential equations and it is the integral transform of the given derivative function with real variable t to convert into a complex function with variable s. This tool is to be used to perform the analytical solution along with the concept of the Heaviside step function for formulating the time depending exciting force.

The Heaviside step function defined as:

$$H(t) = \begin{cases} 0 & \text{for } t < 0.2 \\ 1 & \text{for } t > 0.2 \end{cases}$$

So the function is written like:

$$F(t) = F_1(t).H(0.2 - t)$$

And optimizated for:

$$F_1(t) = 1000.t$$

Obtained from the graph of the time function and the implementation of the Heaviside function, our derived equation of motion follows the time-dependent behaviours and then to the variation of time corresponds a variation of F(t) identified by H(t), thanks to this it is possible to proceed with the derivation of two functions having different F(t).

$$\mathrm{L}\{2x''(t) + 3.73x'(t) + 900x(t)\} = \mathrm{L}\{F(t)\}$$

Where for Heaviside:

$$F(t) = 100 + (t - 0.2)(0 - 500t) + (t - 0.4)(500t - 300)$$

MATLAB CODE FORCE:

```
function F = get_force(t)
    % Force from time vector
    F = zeros(length(t), 1);

    for i = 1:length(t)

        if t(i) <= 0.2
            F(i) = 100;
        elseif t(i) > 0.4
            F(i) = 0;
        else
            F(i) = -500 * t(i) + 300;
        end

    end

end
```

2) Calculate the dynamic responses of the system by using direct numerical integration methods. Three methods are considered.

- Method of Wilson

The numerical integration methods mentioned include discretizing a time period and establishing assumptions about the kinematic (position, velocity, and acceleration) and dynamic (imprinted forces) behavior between the discretized points. We are considering a differential equation of the form: mx"(t) + cx'(t) + kx(t) =F for all subsequent studies (t)

Where m denotes the equivalent mass, x"(t) the acceleration, c the damping constant, x'(t) the velocity, k the stiffness constant, x(t) the displacement, and F(t) an externally applied force.

In case of Wilson Method is an extension of the linear acceleration approach that takes into account the linear variation not only between inserting the factor and adjusting the step size to dt' = θdt. Assume that the acceleration changes linearly from time t to time t +θdt, also this method is unconditionally stable since θ is greater than 1.37, and it is common to adopt the value θ=1.4.

It is feasible to deduce the value of displacement, velocity, and acceleration in the following stages by using the above formula and iterating it in MATLAB.

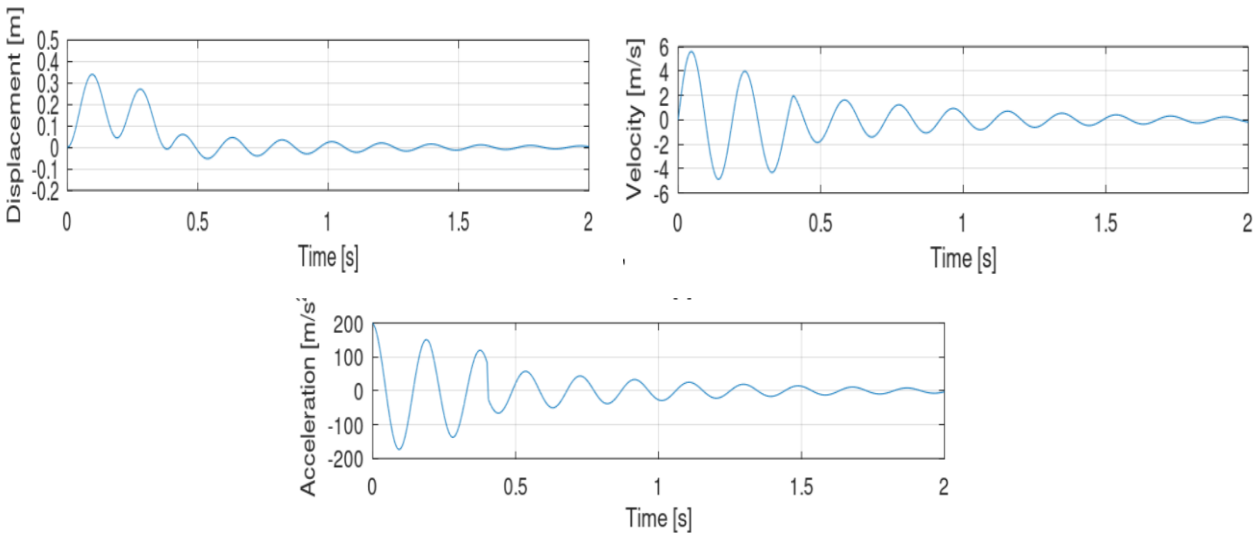
$$\ddot{x}_{i+1} = \frac{6}{\theta^3 \Delta t^2}(x_{i+\theta} - x_i) - \frac{6}{\theta^2 \Delta t}\dot{x}_i + \left(1 - \frac{3}{\theta}\right)\ddot{x}_i$$

$$\dot{x}_{i+1} = \dot{x}_i + \frac{\Delta t}{2}(\ddot{x}_{i+1} + \ddot{x}_i)$$

$$x_{i+1} = x_i + \Delta t \dot{x}_i + \frac{\Delta t^2}{6}(\ddot{x}_{i+1} + 2\ddot{x}_i)$$

The starting location x0 and velocity x'0 values must be supplied before the initial acceleration x"0 can be calculated.

We can then plot the results obtained using this method as a function of three main quantities: displacement, speed and acceleration.



MATLAB CODE WILSON

```
function [x, xdot, x2dot] = wilson_integration(t, F, x0, xdot0, m, k, c, teta)

    % Initializing Variables
    dt = t(2) - t(1);
    x = zeros(length(t), 1); xdot = x; x2dot = x;

    % Initial Conditions
    x(1) = x0; xdot(1) = xdot0;
    x2dot(1) = (F(1) - k * x0 - c * xdot0) / m;

    % Integration Constants
    a0 = 6 / (teta * dt)^2;
    a1 = 3 / (teta * dt);
    a2 = 2 * a1;
    a3 = teta * dt / 2;
    a4 = a0 / teta;
    a5 = -a2 / teta;
    a6 = 1 - (3 / teta);
    a7 = dt / 2;
    a8 = dt^2/6;

    % Form Effective Stiffness
    Keff = k + a0 * m + a1 * c;

    for i = 1:(length(t) - 1)
        Feff = F(i) + teta * (F(i + 1) - F(i)) + ...
            m * (a0 * x(i) + a2 * xdot(i) + 2 * x2dot(i) + ...
            c * (a1 * x(i) + 2 * xdot(i) + a3 * x2dot(i)));
        xi_plus_teta = Feff / Keff;


        % xdot(i + 1), accesses the variable x2dot(i + 1)
        x2dot(i + 1) = a4 * (xi_plus_teta - x(i)) + a5 * xdot(i) + a6 * x2dot(i);
        xdot(i + 1) = xdot(i) + a7 * (x2dot(i + 1) + x2dot(i));
        x(i + 1) = x(i) + dt * xdot(i) + a8 * (x2dot(i + 1) + 2 * x2dot(i));
    end

end
```

- Method of Newmark

Newmark's method, on the other hand, delivers a changing weight in the acceleration extremes of intervals in terms of velocity and acceleration.

The equations representing aceleration, speed and displacement are the following:

$$x_{i+1} = \left(\frac{m}{\alpha\Delta t^2} + \frac{\beta c}{\alpha\Delta t} + k\right)^{-1}\left\{F_{i+1} + \frac{mx_i}{\alpha\Delta t^2} + \frac{m\dot{x}_i}{\alpha\Delta t} + \left(\frac{1}{2\alpha} - 1\right)m\ddot{x}_i + \frac{c\beta x_i}{\alpha\Delta t} + \left(\frac{\beta}{\alpha} - 1\right)c\dot{x}_i + \left(\frac{\beta}{\alpha} - 2\right)\frac{\alpha\Delta t\ddot{x}_i c}{2}\right\}$$

$$\dot{x}_{i+1} = \dot{x}_i + (1-\beta)\,\Delta t\,\ddot{x}_i + \beta\Delta t\,\ddot{x}_{i+1}$$

$$\ddot{x}_{i+1} = \frac{x_{i+1} - x_i}{\alpha\Delta t^2} - \frac{\dot{x}_i}{\alpha\Delta t} - \left(\frac{1}{2\alpha} - 1\right)\ddot{x}_i$$

$\beta = 1/2, \alpha = 1/6$ linear acceleration method

$\beta = 1/2, \alpha = 1/4$ constant acceleration method

And if

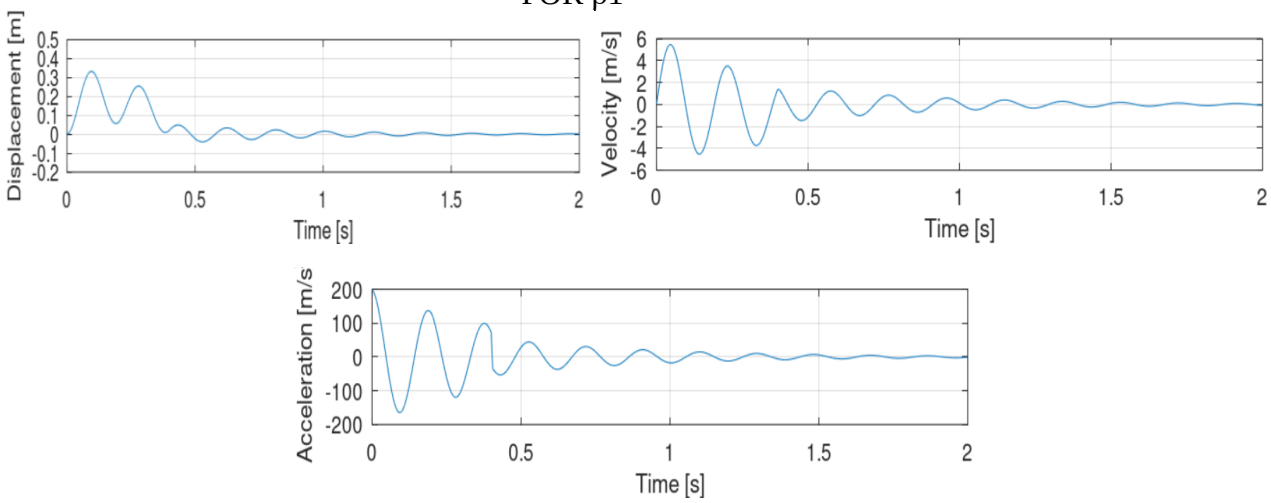
β <1/2 negative damping
β >1/2 positive damping that reduces the response amplitude
β =1/2 no damping

Where α and β are parameters that function as weights in computing the estimate of the acceleration and can be modified to ensure accuracy and stability. To begin the iteration, the initial location x0 and velocity x'0 must be supplied, and these values are then utilized to determine the initial acceleration x"0.

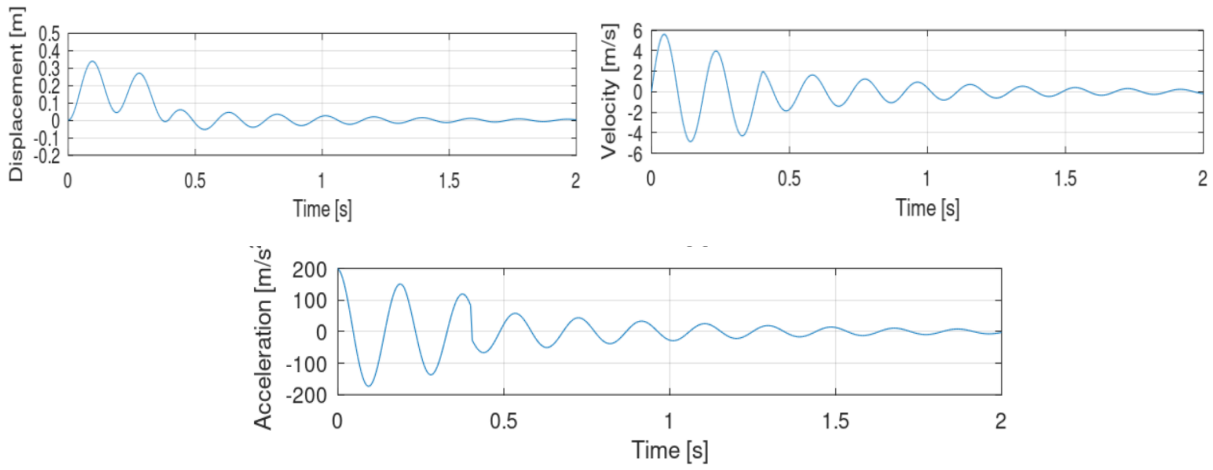
We can then plot the results obtained using this method as a function of three main quantities: displacement, speed and acceleration for:

(β=1/1.5 and α =1/6), (β=1/2 and α =1/6), (β=1/3 and α =1/6), .

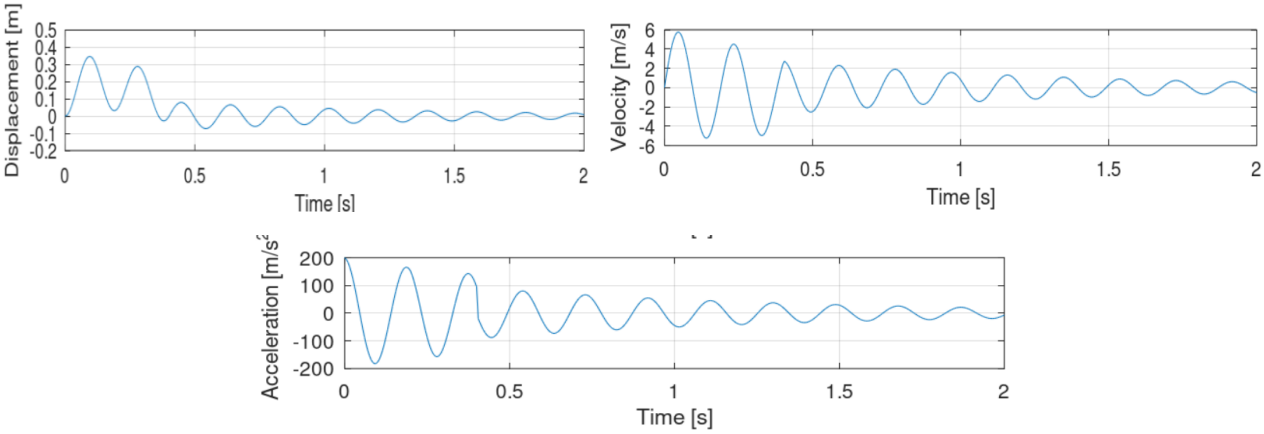
FOR β1



FOR β2



FOR β3



MATLAB CODE NEWMARK:

```
function [x, xdot, x2dot] = newmark_integration(t, F, x0, xdot0, m, k, c, alfa, beta_)
    % Newmark Integration Method

    % Variables
    dt = t(2) - t(1);
    x = zeros(length(t), 1); xdot = x; x2dot = x;

    % Initial Conditions
    x(1) = x0; xdot(1) = xdot0;
    x2dot(1) = (F(1) - k * x0 - c * xdot0) / m;

    % Integration Constants
    a0 = 1 / (alfa * dt^2);
    a1 = beta_ / (alfa * dt);
    a2 = 1 / (alfa * dt);
    a3 = ((1 / (2 * alfa)) - 1);
    a4 = (beta_ / alfa) - 1;
    a5 = ((alfa * dt) / 2) * ((beta_ / alfa) - 2);
    a6 = beta_ * dt;
    a7 = (1 - beta_) * dt;

    % Form Effective Stiffness
    Keff = k + a0 * m + a1 * c;

    for i = 1:(length(t) - 1)
        Feff = F(i + 1) + ...
            m * (a0 * x(i) + a2 * xdot(i) + a3 * x2dot(i)) + ...
            c * (a1 * x(i) + a4 * xdot(i) + a5 * x2dot(i));
        % xdot(i + 1), for example, accesses the variable x2dot(i+1),
        x(i + 1) = Feff / Keff;
        x2dot(i + 1) = a0 * (x(i + 1) - x(i)) - a2 * xdot(i) - a3 * x2dot(i);
        xdot(i + 1) = xdot(i) + a7 * x2dot(i) + a6 * x2dot(i + 1);
    end

end
```

- Method of Central difference

The central differences approach is used to characterize the velocity and acceleration at the positions (i-1) and (i+1).

Where the value i indicates that we are examining the discretized point i at time t, while the subscripts i+1 and i-1 indicate that we are considering the discretized points t + dt and t-dt, respectively.

The equations used for iteration in the speed, acceleration and displacement values are as follows

$$\ddot{x}_0 = \frac{1}{m}(F_0 - c\,\dot{x}_0 - k\,x_0)$$

$$\dot{x}_i = \frac{1}{2\Delta t}(x_{i+1} - x_{i-1})$$

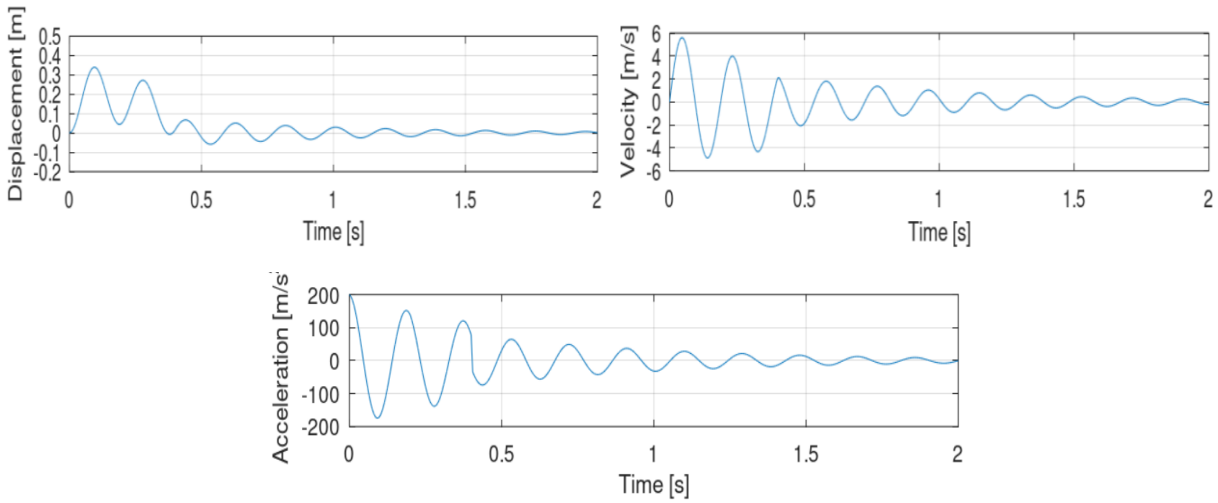
$$x_1 = x_0 - \Delta t\dot{x}_0 + \frac{\Delta t^2}{2}\ddot{x}_0$$

To begin the iteration of the method, supply the initial location x0 and velocity x'0, which are utilized to determine the value of x0" from the differential equation.

This method is conditionally stable, and it is proved that it is stable only:

$$\Delta t \leq T_n / \pi$$

We can then plot the results obtained using this method as a function of three main quantities: displacement, speed and acceleration:



MATLAB CODE CENTRAL DIFFERENCE:

```
function [x, xdot, x2dot] = central_difference_integration(t, F, x0, xdot0, m, k, c)
    % Central Difference Integration Method
    % Initializing Variables
    dt = t(2) - t(1);
    x = zeros(length(t) + 1, 1);
    xdot = zeros(length(t), 1); x2dot = xdot;
```

```
% Initial Conditions
x2dot0 = (F(1) - c * xdot0 - k * x0) / m;
x(1) = x0; xdot(1) = xdot0; x2dot(1) = x2dot0;

% Integration Constants
a0 = ((2 * m) / (dt^2)) - k;
a1 = (c / (2 * dt)) - (m / (dt^2));
K_circunflex = (m / (dt^2)) + (c / (2 * dt));

for i = 1:(length(t))

    % We define the variable x_iminus1 because, in the first iteration
    % (when i=1), x(i-1) is x(0), which does not exist
    if i == 1
        x_iminus1 = x0 - dt * xdot0 + (((dt^2) * x2dot0) / 2);
    else
        x_iminus1 = x(i - 1);
    end
  F_circunflex = a0 * x(i) + a1 * x_iminus1 + F(i);

    % xdot(i + 1), for example, accesses the variable x(i + 1),
    x(i + 1) = F_circunflex / K_circunflex;
    xdot(i) = (x(i + 1) - x_iminus1) / (2 * dt);
    x2dot(i) = (x(i + 1) - 2 * x(i) + x_iminus1) / (dt^2);
end
    x = x(1:end - 1);
  end
```

3) Calculate the dynamic responses of the system by using the methods of response integration

- Method of Integration by Constant Approximation

The equations of equilibrium serve as the foundation for direct integration methods. These strategies are beneficial for complicated or nonlinear systems.

Simpler systems may be solved by an analytical solution of the equations, although this will only apply to harmonic excitations.

For the scenario under consideration, in which we study a system that is initially resting in its equilibrium state, the excitation force may be thought of as a collection of impulses with extremely short duration dt and an amplitude F(T). This results in Duhamel's integral convolution.

$$x(t) = \frac{1}{m\omega_d}\int_0^t F(\tau).\exp\{-\xi\omega_n(t-\tau)\}.\sin\omega_d(t-\tau).d\tau$$

It should be noted that the computation of the system reaction in this situation is fundamentally different from the previous cases where the equation of motion had to be solved at each time instant. In this scenario, it is important to compute anintegration numerically.

Where F(t) is a sequence of impulse superposition. This series of pulses is represented "with an amplitude equal to the average value of the function in time interval d" in the method of

response integration by constant approximation. The system's reaction in the intervals ti and ti-1 may then be calculated.

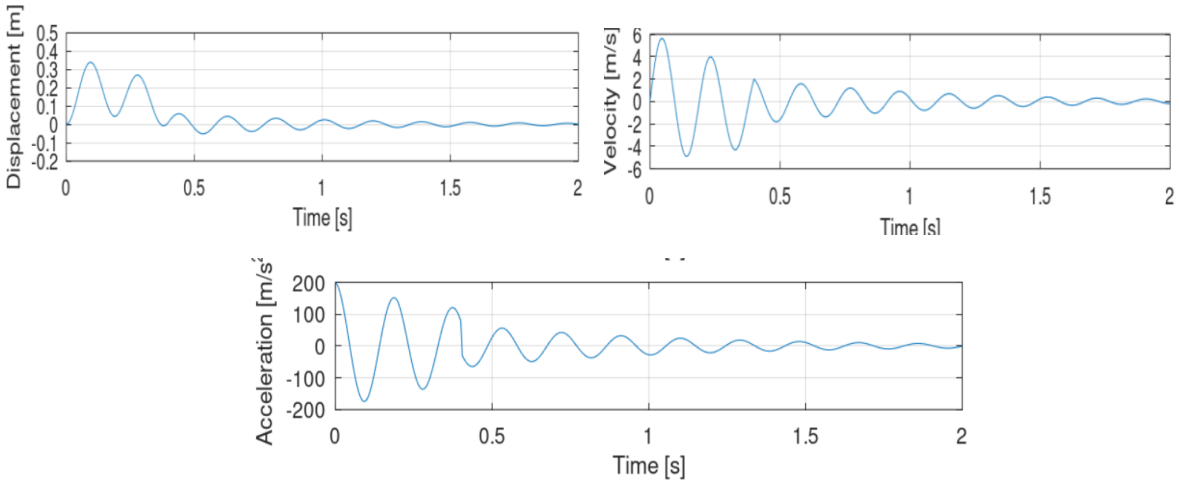
$$x_i = \frac{F_i}{k}\left\{1-\exp\left[-\xi\omega_n\left(t-t_{i-1}\right)\right]\left[\cos\omega_d\left(t-t_{i-1}\right)+\frac{\xi\omega_n}{\omega_d}\sin\omega_d\left(t-t_{i-1}\right)\right]\right\}+$$
$$+\exp\left[-\xi\omega_n\left(t-t_{i-1}\right)\right]\left[x_{i-1}\cos\omega_d\left(t-t_{i-1}\right)+\frac{\dot{x}_{i-1}+\xi\omega_n x_{i-1}}{\omega_d}\sin\omega_d\left(t-t_{i-1}\right)\right]$$

Deriving and substituing t with ti we obtain:

$$\dot{x}_i = \frac{F_i\omega_d}{k}\exp(-\xi\omega_n\Delta t_i)\left(1+\frac{\xi^2\omega_n^2}{\omega_d^2}\right)\sin\omega_d\cdot\Delta t_i+\omega_d\exp(-\xi\omega_n\Delta t_i)\times$$
$$\left[-x_{i-1}\sin\omega_d\cdot\Delta t_i+\frac{\dot{x}_{i-1}+\xi\omega_n x_{i-1}}{\omega_d}\cos\omega_d\cdot\Delta t_i-\frac{\xi\omega_n}{\omega_d}\left(x_{i-1}\cos\omega_d\cdot\Delta t_i+\frac{\dot{x}_{i-1}+\xi\omega_n x_{i-1}}{\omega_d}\sin\omega_d\cdot\Delta t_i\right)\right]$$

These two equations allow the calculation of displacement and velocity at time from the results at the time ti and ti-1.

We can then plot the results obtained using this method as a function of three main quantities: displacement, speed and acceleration:



MATLAB CODE CONSTANT APROXIMATION:

```
function [x, xdot, x2dot] = constant_approximation_int(t, F, x0, xdot0, m, k, c)
    % Constant Approximation Integration Method
    % Initializing Variables
    t = t(2) - t(1);
    x = zeros(length(t), 1); xdot = x; x2dot = x;

    % Initial Conditions
    x(1) = x0; xdot(1) = xdot0;
    x2dot(1) = (F(1) - k * x0 - c * xdot0) / m;

    % Integration Constants
```

```
    wn = sqrt(k / m);
    zeta_ = c / (2 * m * wn);
    wd = wn * sqrt(1 - zeta_^2);

    for i = 2:(length(t))
        dti = t(i) - t(i - 1);

        % Calculating x(i)
        first_factor = (F(i) / k) * (1 - ...
            exp(-zeta_ * wn * dti) * ...
            (cos(wd * dti) + (zeta_ * wn / wd) * sin(wd * dti)));

        second_factor = (exp(-zeta_ * wn * dti)) * ...
            (x(i - 1) * cos(wd * dti) + ...
            ((xdot(i - 1) + zeta_ * wn * x(i - 1)) / wd) * sin(wd * dti));

        x(i) = first_factor + second_factor;

        % Calculating xdot(i)
        first_factor = (F(i) * wd / k) * exp(-zeta_ * wn * dti)* ...
            (1 + (zeta_ * wn / wd)^2) * sin(wd * dti);

        second_factor = wd * exp(-zeta_ * wn * dti);

        third_factor = (-x(i - 1) * sin(wd * dti));

        fourth_factor = ((xdot(i - 1) + zeta_ * wn * x(i - 1)) / wd) * ...
            cos(wd * dti);

        fifth_factor = (-zeta_ * wn / wd) * ...
            (x(i - 1) * cos(wd * dti) + ...
            ((xdot(i - 1) + zeta_ * wn * x(i - 1)) / wd) * sin(wd * dti));

        xdot(i) = first_factor + second_factor * ...
            (third_factor + fourth_factor + fifth_factor);

        % Calculating x2dot(i)
        x2dot(i) = (F(i) - c * xdot(i) - k * x(i)) / m;
    end

end
```

- Method of integration by Linear Approximation

We achieve a better representation of the force's history by describing the fluctuation of the excitation function by a linear approximation between two places. The recurrence formula is produced in the same way as the previous method, with the linear technique that differs in that it considers a linear approximation between two points ti and ti-1.

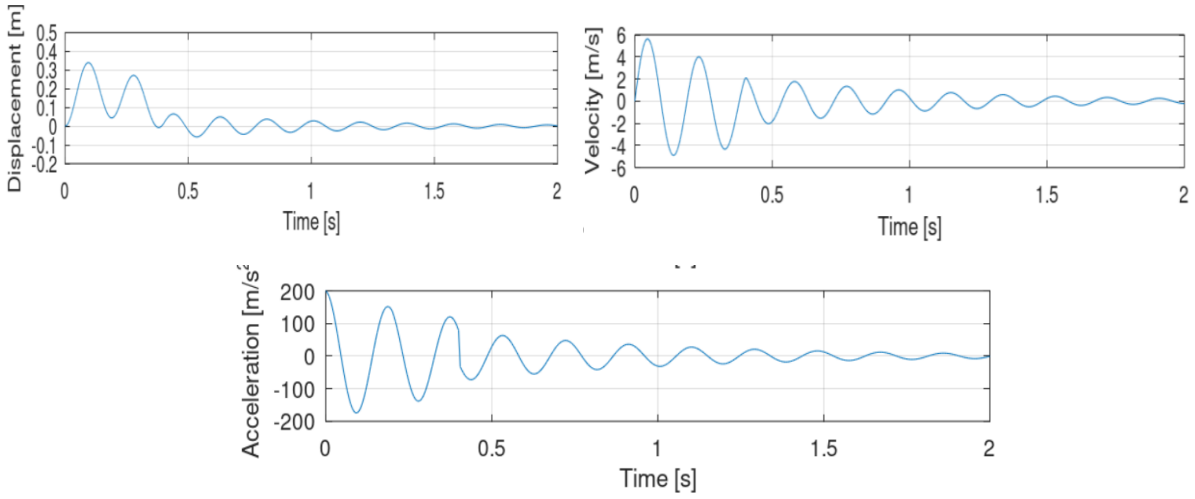
Is possible to obtain displacement and velocity. Afer that the acceleration is calculated from the equation of motion:

$$x_i = \frac{(F_i - F_{i-1})}{k\Delta t_i}\left\{\Delta t_i - \frac{2\xi}{\omega_n} + \exp(-\xi\omega_n \Delta t_i)\left(\frac{2\xi}{\omega_n}\cos\omega_d \Delta t_i - \frac{\omega_d^2 - \xi^2\omega_n^2}{\omega_n{}^2\omega_d}\sin\omega_d \Delta t_i\right)\right\} +$$
$$+ \frac{F_{i-1}}{k}\left[1 - \exp(-\xi\omega_n \Delta t_i)\left(\cos\omega_d \Delta t_i + \frac{\xi\omega_n}{\omega_d}\sin\omega_d \Delta t_i\right)\right] +$$
$$+ \exp(-\xi\omega_n \Delta t_i)\left(x_{i-1}\cos\omega_d \Delta t_i + \frac{\dot{x}_{i-1} + \xi\omega_n x_{i-1}}{\omega_d}\sin\omega_d \Delta t_i\right)$$

And

$$\dot{x}_i = \frac{(F_i - F_{i-1})}{k\Delta t_i}\left\{1 - \exp(-\xi\omega_n \Delta t_i)\left(\cos\omega_d \Delta t_i + \frac{\xi\omega_n}{\omega_d}\sin\omega_d \Delta t_i\right)\right\} +$$
$$+ \frac{F_{i-1}}{k}\exp(-\xi\omega_n \Delta t_i)\frac{\omega_n^2}{\omega_d}\sin\omega_d \Delta t_i + \exp(-\xi\omega_n \Delta t_i).$$
$$\left\{\dot{x}_{i-1}\cos\omega_d \Delta t_i - \frac{\xi\omega_n}{\omega_d}\left(\dot{x}_{i-1} + \frac{\omega_n}{\xi}x_{i-1}\right)\sin\omega_d \Delta t_i\right\}$$

We can then plot the results obtained using this method as a function of three main quantities: displacement, speed and acceleration:



MATLAB CODE LINEAR APROXIMATION:

```
function [x, xdot, x2dot] = linear_approximation_int(t, F, x0, xdot0, m, k, c)
    % Linear Approximation Integration Method
    % Initializing Variables
    t = t(2) - t(1);
    x = zeros(length(t), 1); xdot = x; x2dot = x;

    % Initial Conditions
    x(1) = x0; xdot(1) = xdot0;
    x2dot(1) = (F(1) - k * x0 - c * xdot0) / m;

    % Integration Constants
    wn = sqrt(k / m);
    zeta_ = c / (2 * m * wn);
    wd = wn * sqrt(1 - zeta_^2);
```

```
    for i = 2:(length(t))
        dti = t(i) - t(i - 1);

        % Calculating x(i)
        first_multiplier = ((F(i) - F(i - 1)) / (k * dti));
        f1 = dti; f2 = (2 * zeta_ / wn);
        f3 = exp(-zeta_ * wn * dti);
        f31 = ((2 * zeta_) / wn) * cos(wd * dti);
        f32 = ((wd^2 - zeta_^2 * wn^2) / (wn^2 * wd)) * sin(wd * dti);
        first_factor = first_multiplier * (f1 - f2 + f3 * (f31 - f32));

        second_multiplier = (F(i - 1) / k);
        s1 = 1; s2 = exp(-zeta_ * wn * dti);
        s21 = cos(wd * dti);
        s22 = (zeta_ * wn / wd) * sin(wd * dti);
        second_factor = second_multiplier * (s1 - s2 * (s21 + s22));

        third_multiplier = exp(-zeta_ * wn * dti);
        t1 = x(i - 1) * cos(wd * dti);
        t2 = ((xdot(i - 1) + zeta_ * wn * x(i - 1)) / wd) * sin(wd * dti);
        third_factor = third_multiplier * (t1 + t2);

        x(i) = first_factor + second_factor + third_factor;

        % Calculating xdot(i)
        first_multiplier = ((F(i) - F(i - 1)) / (k * dti));
        f1 = 1; f2 = exp(-zeta_ * wn * dti);
        f21 = cos(wd * dti);
        f22 = (zeta_ * wn / wd) * sin(wd * dti);
        first_factor = first_multiplier * (f1 - f2 * (f21 + f22));

        s1 = (F(i - 1) / k) * exp(-zeta_ * wn * dti) * ((wn^2) / wd) * sin(wd * dti);
        s2 = exp(-zeta_ * wn * dti);
        s21 = xdot(i - 1) * cos(wd * dti);
        s22 = (zeta_ * wn / wd) * (xdot(i - 1) + wn * x(i - 1) / zeta_) * sin(wd *
dti);
        second_factor = s1 + s2 * (s21 - s22);

        xdot(i) = first_factor + second_factor;

        % Calculating x2dot(i)
        x2dot(i) = (F(i) - c * xdot(i) - k * x(i)) / m;
    end

end
```

4) Compare the results obtained above with the analytical calculation by using the method of Laplace trasform method

- Laplace trasform method

The Laplace transform may be used to convert an ordinary differential equation into an algebraic equation, which is often easier to work with. Another feature of the Laplace transform is that it can cope with external forces that are either impulsive (the force lasts a very short time period, such as when a bat strikes a baseball) or that go on and off for a regular or irregular amount of time.

$$\mathcal{L}(f) = \widehat{f}(s) = \int_0^{\infty} e^{-st} f(t)\, dt$$

The properties of applying the Laplace transform to equations are derived naturally from the principles of integration. Among the most important significant qualities in the matter under consideration

At first glance, the formula appears strange; yet, the transform has excellent features that make it attractive in engineering mathematics. For example, it converts the derivative operation to algebraic multiplication: if f 0(t) is the derivative of a function f(t), then L(f 0) = s L(f) f(0): The aforementioned characteristic is useful in solving ordinary differential equations.

The representing function of trasformation is given by the Heaviside function, which identify the behaviour of the trasformation given by range values.

The HEAVISIDE or unit step function $u(t)$ is defined by the following two valued function

$$u(t) = \begin{cases} 0 & t < 0 \\ 1 & t > 0 \end{cases}. \qquad (5.2)$$

If we declare that L(u(t)) = 1 s for s > 0, it indicates that functions u(t) in the t-domain have the representation 1 s in the s-domain through the Laplace transform (see figure).

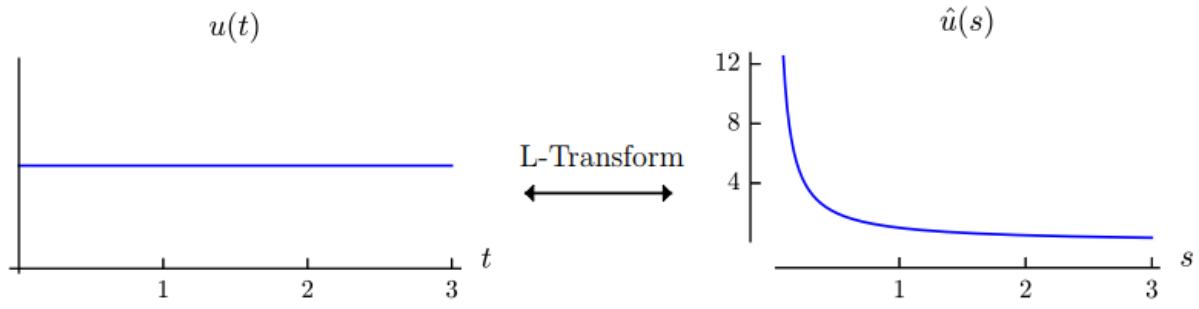
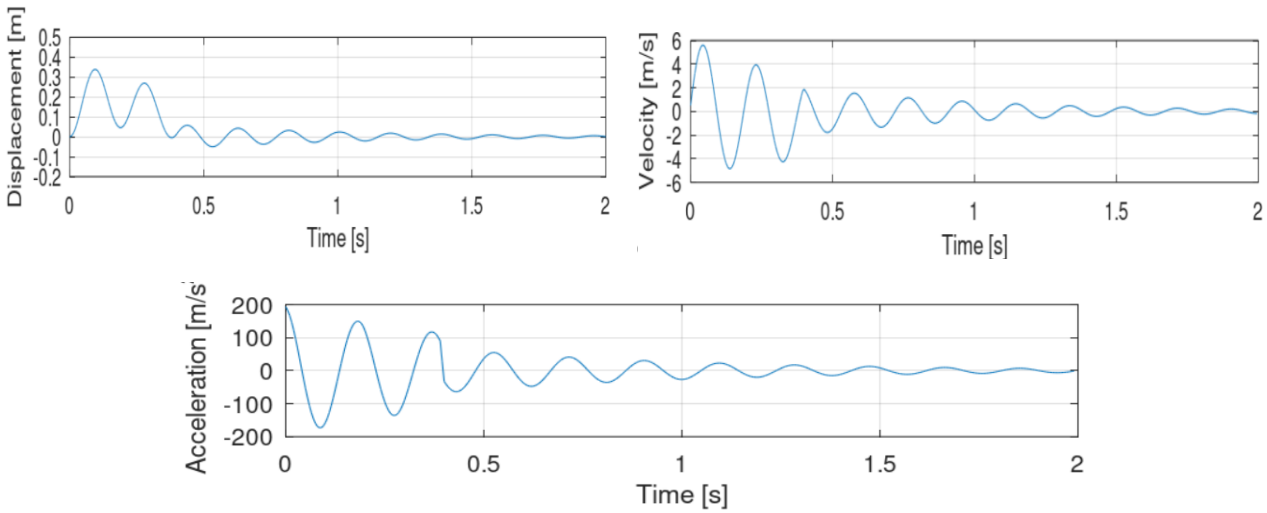


Fig. The representation of inverse trasformation of Laplace

The following relation can also be used to create an inverse transformation from s-domain to t-domain.

$$\mathsf{L}(f(t)) = \hat{f}(s) \Leftrightarrow f(t) = \mathsf{L}^{-1}(\hat{f}(s))$$

We can then plot the results obtained using this method as a function of three main quantities: displacement, speed and acceleration:



MATLAB CODE LAPLACE TRASFORM METHOD:

```
function teta = own_heaviside(t)
    % This is a polyfill for the MATLAB heaviside function,
    % as it doesn't work well on Octave
    if t <= 0
        teta = 0;
   else
        teta = 1;
    end

end
function [x, xdot, x2dot] = solve_by_laplace(times, m, c, k)
    % Analytical Method using Laplace Transform
    syms x t s;

    F = 100 + (0 - 500 * t) * heaviside(t - 0.2) + (500 * t - 300) * heaviside(t -
0.4);
    F_laplace = laplace(F, t, s);
    X_laplace = F_laplace / (m * s^2 + c * s + k);

    x_of_t = ilaplace(X_laplace);
    xdot_of_t = diff(x_of_t, t);
    x2dot_of_t = diff(xdot_of_t, t);

    get_x = @(t) 100 * heaviside(t - 2/5) * ((exp(1521/2500 - (1521 * t) / 1000) *
(cos((1097686559^(1/2) * (t - 2/5)) / 1000) + (1521 * 1097686559^(1/2) *
sin((1097686559^(1/2) * (t - 2/5)) / 1000)) / 1097686559)) / 1100 - 1/1100) - 500 *
heaviside(t - 1/5) * (t / 1100 + (1521 * exp(1521/5000 - (1521 * t) / 1000) *
(cos((1097686559^(1/2) * (t - 1/5)) / 1000) - (547686559 * 1097686559^(1/2) *
sin((1097686559^(1/2) * (t - 1/5)) / 1000)) / 1669581256239)) / 605000000 -
111521/605000000) + 500 * heaviside(t - 2/5) * (t / 1100 + (1521 * exp(1521/2500 -
(1521 * t) / 1000) * (cos((1097686559^(1/2) * (t - 2/5)) / 1000) - (547686559 *
1097686559^(1/2) * sin((1097686559^(1/2) * (t - 2/5)) / 1000)) / 1669581256239)) /
```

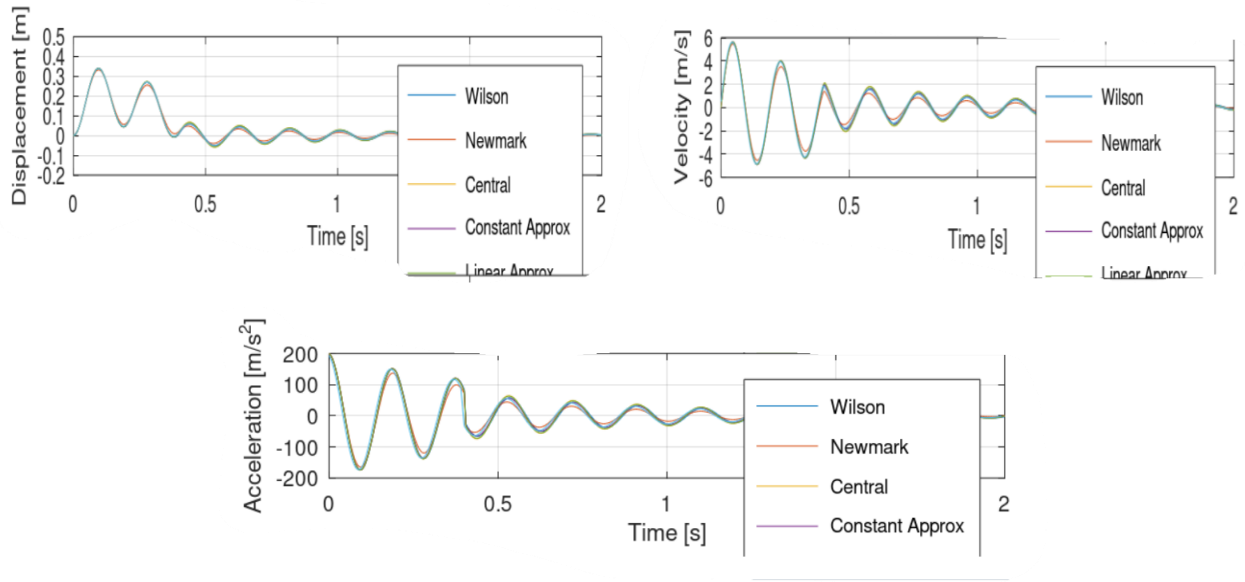
```
605000000 - 221521/605000000) - (2 * exp(-(1521 * t) / 1000) * (cos((1097686559^(1/2) *
t) / 1000) + (1521 * 1097686559^(1/2) * sin((1097686559^(1/2) * t) / 1000)) /
1097686559)) / 11 + 2/11;
    get_xdot = @(t) 500 * heaviside(t - 1/5) * ((1521 * exp(1521/5000 - (1521 * t) /
1000) * ((547686559 * cos((1097686559^(1/2) * (t - 1/5)) / 1000)) / 1521000 +
(1097686559^(1/2) * sin((1097686559^(1/2) * (t - 1/5)) / 1000)) / 1000)) / 605000000 +
(2313441 * exp(1521/5000 - (1521 * t) / 1000) * (cos((1097686559^(1/2) * (t - 1/5)) /
1000) - (547686559 * 1097686559^(1/2) * sin((1097686559^(1/2) * (t - 1/5)) / 1000)) /
1669581256239)) / 605000000000 - 1/1100) - 500 * heaviside(t - 2/5) * ((1521 *
exp(1521/2500 - (1521 * t) / 1000) * ((547686559 * cos((1097686559^(1/2) * (t - 2/5)) /
1000)) / 1521000 + (1097686559^(1/2) * sin((1097686559^(1/2) * (t - 2/5)) / 1000)) /
1000)) / 605000000 + (2313441 * exp(1521/2500 - (1521 * t) / 1000) *
(cos((1097686559^(1/2) * (t - 2/5)) / 1000) - (547686559 * 1097686559^(1/2) *
sin((1097686559^(1/2) * (t - 2/5)) / 1000)) / 1669581256239)) / 605000000000 - 1/1100)
+ 100 * dirac(t - 2/5) * ((exp(1521/2500 - (1521 * t) / 1000) * (cos((1097686559^(1/2)
* (t - 2/5)) / 1000) + (1521 * 1097686559^(1/2) * sin((1097686559^(1/2) * (t - 2/5)) /
1000)) / 1097686559)) / 1100 - 1/1100) - (2 * exp(-(1521 * t) / 1000) * ((1521 *
cos((1097686559^(1/2) * t) / 1000)) / 1000 - (1097686559^(1/2) * sin((1097686559^(1/2)
* t) / 1000)) / 1000)) / 11 - 100 * heaviside(t - 2/5) * ((1521 * exp(1521/2500 - (1521
* t) / 1000) * (cos((1097686559^(1/2) * (t - 2/5)) / 1000) + (1521 * 1097686559^(1/2) *
sin((1097686559^(1/2) * (t - 2/5)) / 1000)) / 1097686559)) / 1100000 - (exp(1521/2500 -
(1521 * t) / 1000) * ((1521 * cos((1097686559^(1/2) * (t - 2/5)) / 1000)) / 1000 -
(1097686559^(1/2) * sin((1097686559^(1/2) * (t - 2/5)) / 1000)) / 1000)) / 1100) - 500
* dirac(t - 1/5) * (t / 1100 + (1521 * exp(1521/5000 - (1521 * t) / 1000) *
(cos((1097686559^(1/2) * (t - 1/5)) / 1000) - (547686559 * 1097686559^(1/2) *
sin((1097686559^(1/2) * (t - 1/5)) / 1000)) / 1669581256239)) / 605000000 -
111521/605000000) + 500 * dirac(t - 2/5) * (t / 1100 + (1521 * exp(1521/2500 - (1521 *
t) / 1000) * (cos((1097686559^(1/2) * (t - 2/5)) / 1000) - (547686559 *
1097686559^(1/2) * sin((1097686559^(1/2) * (t - 2/5)) / 1000)) / 1669581256239)) /
605000000 - 221521/605000000) + (1521 * exp(-(1521 * t) / 1000) *
(cos((1097686559^(1/2) * t) / 1000) + (1521 * 1097686559^(1/2) * sin((1097686559^(1/2)
* t) / 1000)) / 1097686559)) / 5500;
    get_x2dot = @(t) 100 * dirac(1, t - 2/5) * ((exp(1521/2500 - (1521 * t) / 1000) *
(cos((1097686559^(1/2) * (t - 2/5)) / 1000) + (1521 * 1097686559^(1/2) *
sin((1097686559^(1/2) * (t - 2/5)) / 1000)) / 1097686559)) / 1100 - 1/1100) + (1521 *
exp(-(1521 * t) / 1000) * ((1521 * cos((1097686559^(1/2) * t) / 1000)) / 1000 -
(1097686559^(1/2) * sin((1097686559^(1/2) * t) / 1000)) / 1000)) / 2750 - 200 * dirac(t
- 2/5) * ((1521 * exp(1521/2500 - (1521 * t) / 1000) * (cos((1097686559^(1/2) * (t -
2/5)) / 1000) + (1521 * 1097686559^(1/2) * sin((1097686559^(1/2) * (t - 2/5)) / 1000))
/ 1097686559)) / 1100000 - (exp(1521/2500 - (1521 * t) / 1000) * ((1521 *
cos((1097686559^(1/2) * (t - 2/5)) / 1000)) / 1000 - (1097686559^(1/2) *
sin((1097686559^(1/2) * (t - 2/5)) / 1000)) / 1000)) / 1100) + (2 * exp(-(1521 * t) /
1000) * ((1097686559 * cos((1097686559^(1/2) * t) / 1000)) / 1000000 + (1521 *
1097686559^(1/2) * sin((1097686559^(1/2) * t) / 1000)) / 1000000)) / 11 - 500 *
dirac(1, t - 1/5) * (t / 1100 + (1521 * exp(1521/5000 - (1521 * t) / 1000) *
(cos((1097686559^(1/2) * (t - 1/5)) / 1000) - (547686559 * 1097686559^(1/2) *
sin((1097686559^(1/2) * (t - 1/5)) / 1000)) / 1669581256239)) / 605000000 -
111521/605000000) + 500 * dirac(1, t - 2/5) * (t / 1100 + (1521 * exp(1521/2500 - (1521
* t) / 1000) * (cos((1097686559^(1/2) * (t - 2/5)) / 1000) - (547686559 *
1097686559^(1/2) * sin((1097686559^(1/2) * (t - 2/5)) / 1000)) / 1669581256239)) /
605000000 - 221521/605000000) + 500 * heaviside(t - 2/5) * ((2313441 * exp(1521/2500 -
(1521 * t) / 1000) * ((547686559 * cos((1097686559^(1/2) * (t - 2/5)) / 1000)) /
1521000 + (1097686559^(1/2) * sin((1097686559^(1/2) * (t - 2/5)) / 1000)) / 1000)) /
302500000000 + (3518743761 * exp(1521/2500 - (1521 * t) / 1000) *
(cos((1097686559^(1/2) * (t - 2/5)) / 1000) - (547686559 * 1097686559^(1/2) *
sin((1097686559^(1/2) * (t - 2/5)) / 1000)) / 1669581256239)) / 605000000000000 - (1521
* exp(1521/2500 - (1521 * t) / 1000) * ((1097686559 * cos((1097686559^(1/2) * (t -
2/5)) / 1000)) / 1000000 - (547686559 * 1097686559^(1/2) * sin((1097686559^(1/2) * (t -
2/5)) / 1000)) / 1521000000)) / 605000000) - 500 * heaviside(t - 1/5) * ((2313441 *
exp(1521/5000 - (1521 * t) / 1000) * ((547686559 * cos((1097686559^(1/2) * (t - 1/5)) /
1000)) / 1521000 + (1097686559^(1/2) * sin((1097686559^(1/2) * (t - 1/5)) / 1000)) /
```

```
1000)) / 302500000000 + (3518743761 * exp(1521/5000 - (1521 * t) / 1000) *
(cos((1097686559^(1/2) * (t - 1/5)) / 1000) - (547686559 * 1097686559^(1/2) *
sin((1097686559^(1/2) * (t - 1/5)) / 1000)) / 1669581256239)) / 605000000000000 - (1521
* exp(1521/5000 - (1521 * t) / 1000) * ((1097686559 * cos((1097686559^(1/2) * (t -
1/5)) / 1000)) / 1000000 - (547686559 * 1097686559^(1/2) * sin((1097686559^(1/2) * (t -
1/5)) / 1000)) / 1521000000)) / 605000000) - (2313441 * exp(-(1521 * t) / 1000) *
(cos((1097686559^(1/2) * t) / 1000) + (1521 * 1097686559^(1/2) * sin((1097686559^(1/2)
* t) / 1000)) / 1097686559)) / 5500000 - 1000 * dirac(t - 2/5) * ((1521 * exp(1521/2500
- (1521 * t) / 1000) * ((547686559 * cos((1097686559^(1/2) * (t - 2/5)) / 1000)) /
1521000 + (1097686559^(1/2) * sin((1097686559^(1/2) * (t - 2/5)) / 1000)) / 1000)) /
605000000 + (2313441 * exp(1521/2500 - (1521 * t) / 1000) * (cos((1097686559^(1/2) * (t
- 2/5)) / 1000) - (547686559 * 1097686559^(1/2) * sin((1097686559^(1/2) * (t - 2/5)) /
1000)) / 1669581256239)) / 605000000000 - 1/1100) + 1000 * dirac(t - 1/5) * ((1521 *
exp(1521/5000 - (1521 * t) / 1000) * ((547686559 * cos((1097686559^(1/2) * (t - 1/5)) /
1000)) / 1521000 + (1097686559^(1/2) * sin((1097686559^(1/2) * (t - 1/5)) / 1000)) /
1000)) / 605000000 + (2313441 * exp(1521/5000 - (1521 * t) / 1000) *
(cos((1097686559^(1/2) * (t - 1/5)) / 1000) - (547686559 * 1097686559^(1/2) *
sin((1097686559^(1/2) * (t - 1/5)) / 1000)) / 1669581256239)) / 605000000000 - 1/1100)
- 100 * heaviside(t - 2/5) * ((1521 * exp(1521/2500 - (1521 * t) / 1000) * ((1521 *
cos((1097686559^(1/2) * (t - 2/5)) / 1000)) / 1000 - (1097686559^(1/2) *
sin((1097686559^(1/2) * (t - 2/5)) / 1000)) / 1000)) / 550000 - (2313441 *
exp(1521/2500 - (1521 * t) / 1000) * (cos((1097686559^(1/2) * (t - 2/5)) / 1000) +
(1521 * 1097686559^(1/2) * sin((1097686559^(1/2) * (t - 2/5)) / 1000)) / 1097686559)) /
1100000000 + (exp(1521/2500 - (1521 * t) / 1000) * ((1097686559 * cos((1097686559^(1/2)
* (t - 2/5)) / 1000)) / 1000000 + (1521 * 1097686559^(1/2) * sin((1097686559^(1/2) * (t
- 2/5)) / 1000)) / 1000000)) / 1100);

    x = zeros(length(times), 1); xdot = x; x2dot = x;

    for i = 1:length(times)
        x(i) = get_x(times(i));
        xdot(i) = get_xdot(times(i));
        x2dot(i) = get_x2dot(times(i));
    end

end
function f = solve_hardcoded(t)
    % Inverse Laplace Transform of function F(s) (see below)
    f = zeros(length(t), 1);

    for i = 1:length(t)
        f(i) = -0.047027 * exp(0 - 1.5271 * t(i)) * own_heaviside(t(i) - 0.4) * sin(0 -
33.1311 * t(i)) - ...
            0.454545 * t(i) * own_heaviside(t(i) - 0.4) * sin(0 - 33.1311 * t(i)) *
sin(33.1311 * t(i) + 0) + ...
            0.273989 * own_heaviside(t(i) - 0.4) * sin(0 - 33.1311 * t(i)) *
sin(33.1311 * t(i) + 0) - ...
            0.0712084 * exp(0 - 1.5271 * t(i)) * own_heaviside(t(i) - 0.4) * sin(0 -
33.1311 * t(i)) * sin(66.2621 * t(i) + 0) - ...
            0.00844246 * exp(0 - 1.5271 * t(i)) * own_heaviside(t(i) - 0.2) * sin(0 -
33.1311 * t(i)) + ...
            0.454545 * t(i) * own_heaviside(t(i) - 0.2) * sin(0 - 33.1311 * t(i)) *
sin(33.1311 * t(i) + 0) - ...
            0.0921712 * own_heaviside(t(i) - 0.2) * sin(0 - 33.1311 * t(i)) *
sin(33.1311 * t(i) + 0) + ...
            0.00392463 * exp(0 - 1.5271 * t(i)) * own_heaviside(t(i) - 0.2) * sin(0 -
33.1311 * t(i)) * sin(66.2621 * t(i) + 0) + ...
            0.0712084 * exp(0 - 1.5271 * t(i)) * own_heaviside(t(i) - 0.4) * cos(0 -
33.1311 * t(i)) + ...
            0.454545 * t(i) * own_heaviside(t(i) - 0.4) * cos(33.1311 * t(i) + 0) *
cos(0 - 33.1311 * t(i)) - ...
```

```
            0.273989 * own_heaviside(t(i) - 0.4) * cos(33.1311 * t(i) + 0) * cos(0 -
33.1311 * t(i)) + ...
            0.0712084 * exp(0 - 1.5271 * t(i)) * own_heaviside(t(i) - 0.4) *
cos(66.2621 * t(i) + 0) * cos(0 - 33.1311 * t(i)) - ...
            0.00392463 * exp(0 - 1.5271 * t(i)) * own_heaviside(t(i) - 0.2) * cos(0 -
33.1311 * t(i)) - ...
            0.454545 * t(i) * own_heaviside(t(i) - 0.2) * cos(33.1311 * t(i) + 0) *
cos(0 - 33.1311 * t(i)) + ...
            0.0921712 * own_heaviside(t(i) - 0.2) * cos(33.1311 * t(i) + 0) * cos(0 -
33.1311 * t(i)) - ...
            0.00392463 * exp(0 - 1.5271 * t(i)) * own_heaviside(t(i) - 0.2) *
cos(66.2621 * t(i) + 0) * cos(0 - 33.1311 * t(i)) + ...
            2.77556 * 10^-17 * t(i) * own_heaviside(t(i) - 0.4) * sin(33.1311 * t(i) +
0) * cos(0 - 33.1311 * t(i)) - ...
            1.67304 * 10^-17 * own_heaviside(t(i) - 0.4) * sin(33.1311 * t(i) + 0) *
cos(0 - 33.1311 * t(i)) + ...
            0.047027 * exp(0 - 1.5271 * t(i)) * own_heaviside(t(i) - 0.4) * sin(66.2621
* t(i) + 0) * cos(0 - 33.1311 * t(i)) - ...
            2.77556 * 10^-17 * t(i) * own_heaviside(t(i) - 0.2) * sin(33.1311 * t(i) +
0) * cos(0 - 33.1311 * t(i)) + ...
            5.45863 * 10^-20 * own_heaviside(t(i) - 0.2) * sin(33.1311 * t(i) + 0) *
cos(0 - 33.1311 * t(i)) + ...
            0.00844246 * exp(0 - 1.5271 * t(i)) * own_heaviside(t(i) - 0.2) *
sin(66.2621 * t(i) + 0) * cos(0 - 33.1311 * t(i)) + ...
            2.77556 * 10^-17 * t(i) * own_heaviside(t(i) - 0.4) * sin(0 - 33.1311 *
t(i)) * cos(33.1311 * t(i) + 0) - ...
            1.67304 * 10^-17 * own_heaviside(t(i) - 0.4) * sin(0 - 33.1311 * t(i)) *
cos(33.1311 * t(i) + 0) + ...
            0.047027 * exp(0 - 1.5271 * t(i)) * own_heaviside(t(i) - 0.4) * sin(0 -
33.1311 * t(i)) * cos(66.2621 * t(i) + 0) - ...
            2.77556 * 10^-17 * t(i) * own_heaviside(t(i) - 0.2) * sin(0 - 33.1311 *
t(i)) * cos(33.1311 * t(i) + 0) + ...
            5.45863 * 10^-20 * own_heaviside(t(i) - 0.2) * sin(0 - 33.1311 * t(i)) *
cos(33.1311 * t(i) + 0) + ...
            0.00844246 * exp(0 - 1.5271 * t(i)) * own_heaviside(t(i) - 0.2) * sin(0 -
33.1311 * t(i)) * cos(66.2621 * t(i) + 0) + ...
            0.00419024 * exp(0 - 1.5271 * t(i)) * sin(0 - 33.1311 * t(i)) - 0.181818 *
sin(0 - 33.1311 * t(i)) * sin(33.1311 * t(i) + 0) + ...
            0.0909091 * exp(0 - 1.5271 * t(i)) * sin(0 - 33.1311 * t(i)) * sin(66.2621
* t(i) + 0) - 0.0909091 * exp(0 - 1.5271 * t(i)) * cos(0 - 33.1311 * t(i)) + ...
            0.181818 * cos(33.1311 * t(i) + 0) * cos(0 - 33.1311 * t(i)) - 0.0909091 *
exp(0 - 1.5271 * t(i)) * cos(66.2621 * t(i) + 0) * cos(0 - 33.1311 * t(i)) - ...
            4.49566 * 10^-20 * sin(33.1311 * t(i) + 0) * cos(0 - 33.1311 * t(i)) -
0.00419024 * exp(0 - 1.5271 * t(i)) * sin(66.2621 * t(i) + 0) * cos(0 - 33.1311 * t(i))
- ...
            4.49566 * 10^-20 * sin(0 - 33.1311 * t(i)) * cos(33.1311 * t(i) + 0) -
0.00419024 * exp(0 - 1.5271 * t(i)) * sin(0 - 33.1311 * t(i)) * cos(66.2621 * t(i) +
0);
    end

end
```

- Comparison

Finally, we can note that through comparison between the different numerical methods and methods of integration, the response of the system persists consistent and does not deviate much from the trend, also in reference to the use of the method of inverse Laplace transformation we always get a trend quite synchronized with the others, this means that the calculation procedure has been done correctly and the behavior in response to a viscous dumping through the analysis of different methods is efficient and it is possible, albeit slightly, to identify the method with the lowest peak in reference to the three quantities of the equation of motion: displacement, velocity and acceleration, thus predicting which method best approximates the response and avoids high accelerations in the reference system under consideration, having one degree of freedom.



MATLAB CODE COMPARISON:

```
function _ = plot_numerical_vs_analytical(t, x_wilson, xdot_wilson, x2dot_wilson,
x_newmark, xdot_newmark, x2dot_newmark, x_central, xdot_central, x2dot_central,
x_const, xdot_const, x2dot_const, x_linear, xdot_linear, x2dot_linear, x_analytical,
xdot_analytical, x2dot_analytical)

    xlabel_ = "Time [s]";
    title_ = "Comparison of Wilson, Newmark, Central Difference, Constant
Approximation, Linear Approximation and Analytical methods";
    legend_ = ["Wilson", "Newmark", "Central", "Constant Approx", "Linear Approx",
"Analytical"];

    figure_ = figure("name", title_);
    title(title_);

    subplot(3, 1, 1);
    plot(t, x_wilson,t; x_newmark,t; x_central,t; x_const,t; x_linear,t; x_analytical);
xlabel(xlabel_);
    ylabel("Displacement [m]");
    legend(legend_);
    grid on;

    subplot(3, 1, 2);
```

```
    plot(t, xdot_wilson,
    t; xdot_newmark,
    t; xdot_central,
    t; xdot_const,
    t; xdot_linear,
    t(:, 1:length(xdot_analytical)), xdot_analytical);
    xlabel(xlabel_);
    ylabel("Velocity [m/s]");
    legend(legend_);
    grid on;

    subplot(3, 1, 3);
    plot(t, x2dot_wilson,
    t; x2dot_newmark,
    t; x2dot_central,
    t; x2dot_const,
    t; x2dot_linear,
    t(:, 1:length(x2dot_analytical)), x2dot_analytical);
    xlabel(xlabel_);
    ylabel("Acceleration [m/s^2]");
    legend(legend_);
    grid on;
    saveas(figure_, strcat("./autogenerated_figures/", title_, ".png"));
end
```

5) Conclusions

A one-degree-of-freedom system with viscous damping was investigated in this research, with an arrangement of springs and dampers and the application of a vibration response, and defined in terms of the Heaviside function. The equivalent mass, spring constant, and damping constant of the system were derived using provided data on the system's behavior over time, the building of the free body diagram, and implications of the solution of the associated differential equation. Following the construction of MATLAB code, the system response was studied using direct numerical integration methods (central difference, Newmark, Wilson) and response integration by constant and linear approximation. The analytical solution was then achieved through the use of the Laplace transform in both MATLAB and the manual technique.

According to the visual and numerical comparison of the methods' results with the analytical solution, the response integration methods appear to be more accurate; the Newmark and Wilson methods were more accurate than the central difference method; and, for the type of system analyzed, a choice of $\beta = 1/2$ is more indicated for the application of the Newmark method. A multivariate analysis considering an adjustment of the integration variables assuming a modification of the sampling interval is proposed for further research. A further examination of the Wilson approach is also advised, in order to explore its merits and shortcomings in comparison to the Newmark method by varying the variables α, β, θ and in order to explore its merits and shortcomings.

References:

- Vibration of Ships and Platforms Slides , https://fenix.tecnico.ulisboa.pt/disciplinas/VN/2021-2022/2-semestre/slides
- Wolfram Alpha Inverse Laplace Transform Calculator, https://www.wolframalpha.com/input?i=real+part+inverse+Laplace+transform+%28%28e%5E%28-0.4+s%29+%28500+-+100+s%29%29%2Fs%5E2+-+%28500+e%5E%28-0.2+s%29%29%2Fs%5E2+%2B+200%2Fs%29%2F%28s%5E2+%2B+3.0542+s+%2B+1100%29

- Project work/Ship vibration, https://fenix.tecnico.ulisboa.pt/disciplinas/VN/2021-2022/2-semestre/project1_2022
- STABILITY_AND_ACCURACY_ANALYSIS_OF_DIREC.pdf
- https://sites.ualberta.ca/~niksirat/ODE/chapter-5ode.pdf

MAIN MATLAB CODE:

```
close all; % Closes all figures
clear; % Clears variables from workspace
clc; % Clears all text from command window

% Importing functions from folders with given relative path
addpath(genpath("./formulas"));
addpath(genpath("./plotting"));
addpath(genpath("./response_integration"));
addpath(genpath("./direct_integration"));
addpath(genpath("./analytical"));

t = 0:dt:2; % Time vector (t(i), t(i + 1), t(i + 2), ...) | Seconds
F = get_force(t); % Force vector (F(t(i), F(t(i + 1)), ...) | Newtons
x0 = 0; % Initial position | Meters
xdot0 = 0; % Initial velocity | Meters per Second
M = 2; % Equivalent Mass | Kilograms
K = (600 + 200 + 100); % Spring Constant | Newtons per Meter

% Calculating Damping Constant C
N = 5; % Number of oscilations
A_ratio = 1/0.25; % Ratio between initial amplitude (A0) and amplitude in time NT
(A(NT))
C = get_damping_coefficient(A_ratio, M, K, N); % Damping Constant | Kilograms per
Second

% Direct Integration

teta = 1.4; alfa = 1/6; beta_a = 1/1.5; beta_b = 1/2; beta_c = 1/3;
[x_wilson, xdot_wilson, x2dot_wilson] = wilson_integration(t, F, x0, xdot0, M, K, C,
teta);
```

```
[xa_newmark, xadot_newmark, xa2dot_newmark] = newmark_integration(t, F, x0, xdot0, M,
K, C, alfa, beta_a);
[xb_newmark, xbdot_newmark, xb2dot_newmark] = newmark_integration(t, F, x0, xdot0, M,
K, C, alfa, beta_b);
[xc_newmark, xcdot_newmark, xc2dot_newmark] = newmark_integration(t, F, x0, xdot0, M,
K, C, alfa, beta_c);
[x_central, xdot_central, x2dot_central] = central_difference_integration(t, F, x0,
xdot0, M, K, C);
[x_const, xdot_const, x2dot_const] = constant_approximation_int(t, F, x0, xdot0, M, K,
C);
[x_linear, xdot_linear, x2dot_linear] = linear_approximation_int(t, F, x0, xdot0, M, K,
C);

% Analytical Solution using Laplace Transform
x_analytical = solve_hardcoded(t);
xdot_analytical = diff(x_analytical) / dt;
x2dot_analytical = diff(xdot_analytical) / dt;

% Plot all methods separately
plot_method(t, x_wilson, xdot_wilson, x2dot_wilson, "Integration with Method of
Wilson");
plot_method(t, xa_newmark, xadot_newmark, xa2dot_newmark, "Integration with Method of
Newmark 1");
plot_method(t, xb_newmark, xbdot_newmark, xb2dot_newmark, "Integration with Method of
Newmark 2");
plot_method(t, xc_newmark, xcdot_newmark, xc2dot_newmark, "Integration with Method of
Newmark 3");
plot_method(t, x_central, xdot_central, x2dot_central, "Integration with Method of
Central Difference");
plot_method(t, x_const, xdot_const, x2dot_const, "Response Integration by Constant
Approximation");
plot_method(t, x_linear, xdot_linear, x2dot_linear, "Response Integration by Linear
Approximation");
plot_method(t, x_analytical, xdot_analytical, x2dot_analytical, "Analytical solution
(using Laplace transform and Heaviside step function");

% Plot Newmark comparison for different betas
plot_newmark_comparison(t, alfa, xa_newmark, xadot_newmark, xa2dot_newmark, beta_a,
xb_newmark, xbdot_newmark, xb2dot_newmark, beta_b, xc_newmark, xcdot_newmark,
xc2dot_newmark, beta_c);

% Write text files containing data from displacement, velocity and acceleration
X = [t; x_wilson'; xa_newmark'; xb_newmark'; xc_newmark'; x_central'; x_const';
x_linear'; x_analytical'];
write_text_file(X, "Displacement");

XDOT = [t; xdot_wilson'; xadot_newmark'; xbdot_newmark'; xcdot_newmark'; xdot_central';
xdot_const'; xdot_linear'; [xdot_analytical; 0]'];
write_text_file(XDOT, "Velocity");

X2DOT = [t; x2dot_wilson'; xa2dot_newmark'; xb2dot_newmark'; xc2dot_newmark';
x2dot_central'; x2dot_const'; x2dot_linear'; [x2dot_analytical; 0; 0]'];
write_text_file(X2DOT, "Acceleration");
```