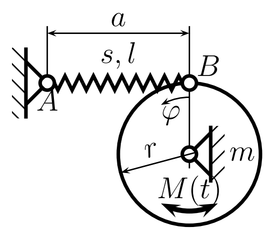
Faculty of Mechanical Engineering, BUTE	NONLINEAR VIBRATIONS	Name: Kelle Gergő
Dept. of Applied Mechanics	HOMEWORK 2	Neptun code: GBBNUL
2023/24/2 (Spring)	Deadline: 2024 May 14	Signature:

Analysis of a nonlinear mechanical system

The sketched 1 D.o.F. mechanical system can move in the horizontal plane. The tensionless length of the spring of stiffness s is l.



Data

l [cm]	a [cm]	r [cm]	l_{min} [cm]	l_{max} [cm]	m [kg]	s [N/m]
12	12	5	5	20	3	600

Tasks:

1. Choosing the coordinate can be seen in the figure, determine the potential function $\mathcal{U}$ of the *unexcited* system and the possible equilibrium points.
2. Draw the potential function and sketch some typical trajectories of the phase space in accordance with the drawn function. Generate the phase portrait with an appropriate PC software, too.
3. Draw the bifurcation diagram of the equilibrium points choosing l as bifurcation parameter in the range $l \in [l_{min}, l_{max}]$. Mark the stable and unstable branches in the diagram, too.
4. Determine *approximately* the dependence of the time-period on the amplitude of *free* oscillation around the stable equilibrium occuring at the parameter value $a = 12$ cm with the help of Poincare's small parameter method. Check the analytical results with numerical calculations (simulations).
5. Determine the amplification diagram of the stationary oscillations around the previous equilibrium point for the case of *harmonic excitation*. Choose some typical excitation amplitudes for the diagram.

Budapest University of Technology and Economics
Faculty of Mechanical Engineering

Nonlinear Vibrations
II. Homework

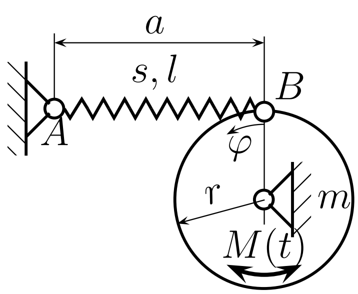
Gergő Kelle

May 13, 2024

Contents

Initial Data

The initial data corresponding to my NEPTUN code is present on Figure 1, along the scaled representation of the problem.



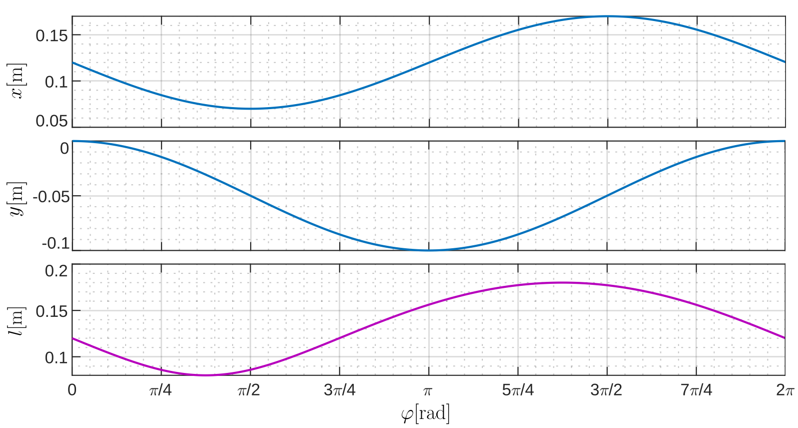
Data

l [m]	a [m]	r [m]	l_{min} [m]	l_{max} [m]	m [kg]	s [N/m]
0.12	0.12	0.05	0.05	0.2	3	600

Figure 1: Scaled representation of the problem

1 Task - Potential Function and The Equation of Motion

For the potential energy function we have to determine the function of the spring length as φ varies. The distance from point A to point B is:



$$x: \quad a - r\sin(\varphi) \tag{1}$$

$$y: \quad r\left(\cos(\varphi) - 1\right) \tag{2}$$

$$l: \quad \sqrt{x^2 + y^2} \tag{3}$$

Figure 2: Spring length

Since we determined the spring's length change as φ varies potential energy function can be easily written as:

$$U = \frac{1}{2}s\left(\sqrt{(a - r\sin(\varphi))^2 + (r(\cos(\varphi) - 1))^2} - l\right)^2 \tag{4}$$

While the kinetic energy of the nonlinear system is:

$$T = \frac{1}{2}\,\Theta\,\dot{\varphi}^2 \tag{5}$$

$$\Theta = \frac{1}{2}mr^2 \tag{6}$$

$$T = \frac{1}{2}\left(\frac{1}{2}mr^2\right)\dot{\varphi}^2 \tag{7}$$

The equation of motion can be constructed with the help of Lagrange equation of the second kind

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial T}{\partial \dot{\varphi}} - \underbrace{\frac{\partial T}{\partial \varphi}}_{=0} + \underbrace{\frac{\partial D}{\partial \dot{\varphi}}}_{=0} + \frac{\partial U}{\partial \varphi} = 0. \tag{8}$$

After the ellimination of zero parts the partial derivative of potential energy and the kinetic energy is the following.

$$\frac{\partial U}{\partial \varphi} = \frac{s\left(l - \sqrt{r^2\left(\cos(\varphi)-1\right)^2 + \left(a - r\sin(\varphi)\right)^2}\right) \cdot r\left(a\cos(\varphi) - r\sin(\varphi)\right)}{\sqrt{r^2\left(\cos(\varphi)-1\right)^2 + \left(a - r\sin(\varphi)\right)^2}} \tag{9}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial T}{\partial \dot{\varphi}} = \frac{1}{2}mr^2\ddot{\varphi} \tag{10}$$

Therefore the equation of motion is:

$$\frac{1}{2}mr^2\ddot{\varphi} + \frac{\partial U}{\partial \varphi} = 0 \tag{11}$$

The equlibrium points can be calculated as

$$\frac{\partial U}{\partial \varphi} = 0. \tag{12}$$

With the help of the potential function plot it is easy to point out the equlibrium points.

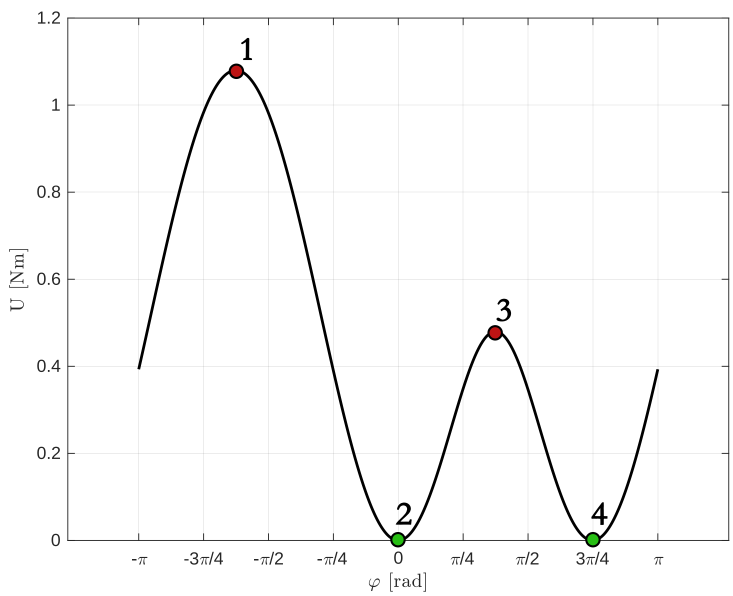


Figure 3: Potential Function and Equlibrium Points

Equilibrium points:

$$\varphi_1 = -1.9706 \tag{13}$$

$$\varphi_2 = 0 \tag{14}$$

$$\varphi_3 = \frac{3}{8}\pi \tag{15}$$

$$\varphi_4 = \frac{3}{4}\pi \tag{16}$$

2 Task - Potential Function and Phase Portrait Plot

The potential function plot has already showed at Figure 3. After reordering the Equation (11) we get the following from of equation of motion:

$$\ddot{\varphi} + \underbrace{\frac{2}{mr^2}\frac{\partial U}{\partial \varphi}}_{f(\varphi):=} = 0. \tag{17}$$

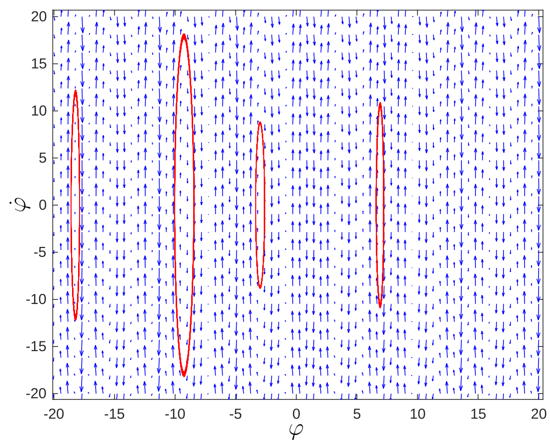


Figure 4: Phase portrait and trajectories

Using the Cauchy transformation we get:

$$x = \varphi \tag{18}$$

$$y = \dot{\varphi} \tag{19}$$

$$\dot{x} = y \tag{20}$$

$$\dot{y} = -f(x) \tag{21}$$

3 Task - Bifurcation Diagram

The bifurcation diagram were created with the bifurcation parameter of l that varies from l_{min} to l_{max}. On the plot the stable points are green while the unstables are indicated with red color. The stable part represents the local minimum places of the potential function while the unstable parts are stands for local maximas.

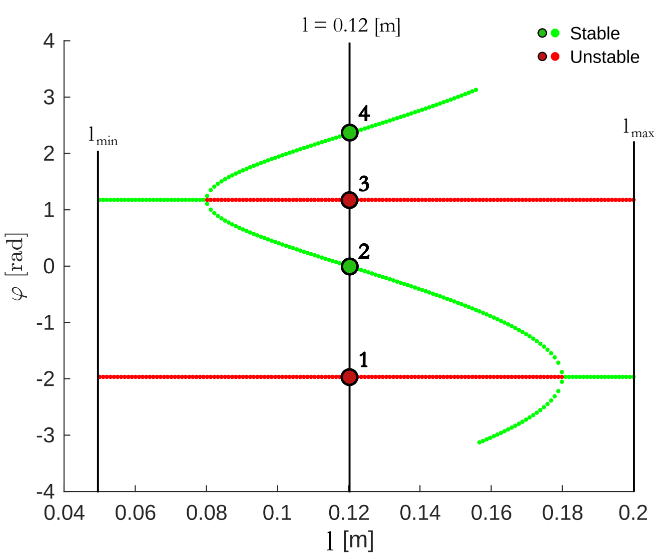


Figure 5: Bifurcation Diagram

The plot show similar results for the equlibrium points as determined before, both for its location and for its type.

4 Task - Poincare's Small Parameter Method

The task asks to choose a to be 12 cm, but this is the same value as my initial data for this variable. Consequently, the stable equilibrium occurs at $\varphi = 0$, given that the spring is under tension for all other φ values. I will use Poincare's small parameter method to approximate time period. The calculation involves determining the 3rd order Taylor polynomial around the equilibrium point, where higher-order terms can be neglected.

$$f(\varphi) = f(0) + \frac{1}{1!}f'(0)\varphi + \frac{1}{2!}f''(0)\varphi^2 + \frac{1}{3!}f'''(0)\varphi^3 \dots \tag{22}$$

Since $f(0) = 0$ and $f''(0) = 0$ the equation can be simplified to

$$f(\varphi) = \omega_{n,lin}^2\varphi + \mu\varphi^3 \tag{23}$$

where $\omega_{n,lin}$ is the natural angular frequency in the linear case and mu is the small parameter. These parameters can be deduced from the derivative of the nonlinearity function

$$\omega_{n,lin} = \sqrt{f'(0)} = 20 \left[\frac{\text{rad}}{\text{s}}\right] \tag{24}$$

$$\mu = \frac{1}{3!}f'''(0) = -266.6667 \quad \left[\frac{\text{rad}}{\text{s}^2}\right] \tag{25}$$

Subsequently, the time period can be determined as:

$$T = \frac{2\pi}{\omega_n} \approx \frac{2\pi}{\sqrt{\omega_{n,lin}^2 + \frac{3}{4}\mu A^2}} \tag{26}$$

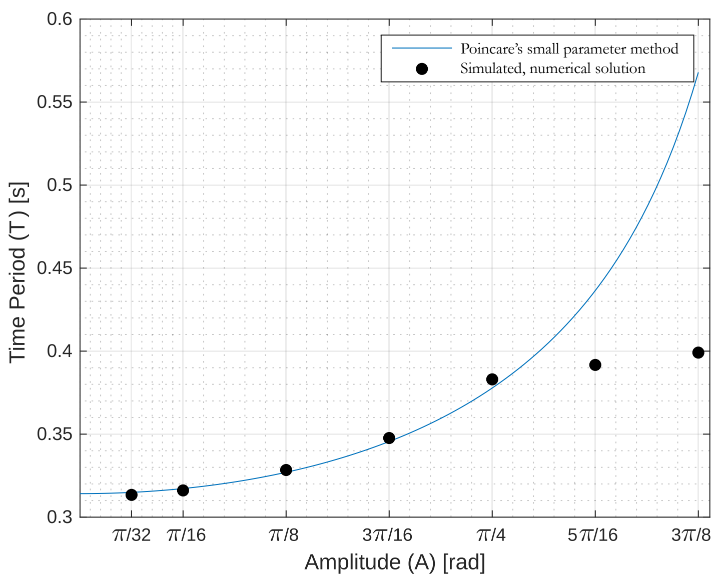


Figure 6: Time-period - Amplitude Dependence Diagram

The simulated results for the time-period were obtained by numerically solving the equation of motion using the *ode45* solver in *MATLAB*. The initial conditions of $\varphi(t=0) = X$ and $\dot{\varphi}(t=0) = 0$ were used to investigate the behavior of the system under different starting conditions (since X vary from $\pi/32$ to $3\pi/8$). The time-period was then calculated from the resulting motion trajectory, allowing for comparison with the theoretical predictions obtained from Poincare's small parameter method.

5 Task - Amplification Diagram of The Stationary Oscillations

Utilizing Poincare's small parameter method, we derive the equation:

$$\omega_n^2 A + \frac{3}{4}\mu A^3 = f_0\omega_n^2 + \omega_{n,lin}^2 A. \tag{27}$$

Here, f_0 represents harmonic excitation and is expressed as:

$$f_0 = \frac{F_0}{m\omega_n^2} \tag{28}$$

The middle curve, or the so called "backbone-curve" is obtained for $f_0 = 0$, which yields:

$$\frac{\omega_{n,lin}^2}{\omega_n^2} - \frac{A^2}{\frac{4\omega_n^2}{3\mu}} = 1 \tag{29}$$

The amplification diagram for stationary oscillations around equilibrium, considering f_0 values of 0, 0.2, and 0.4, is depicted in Figure 7.

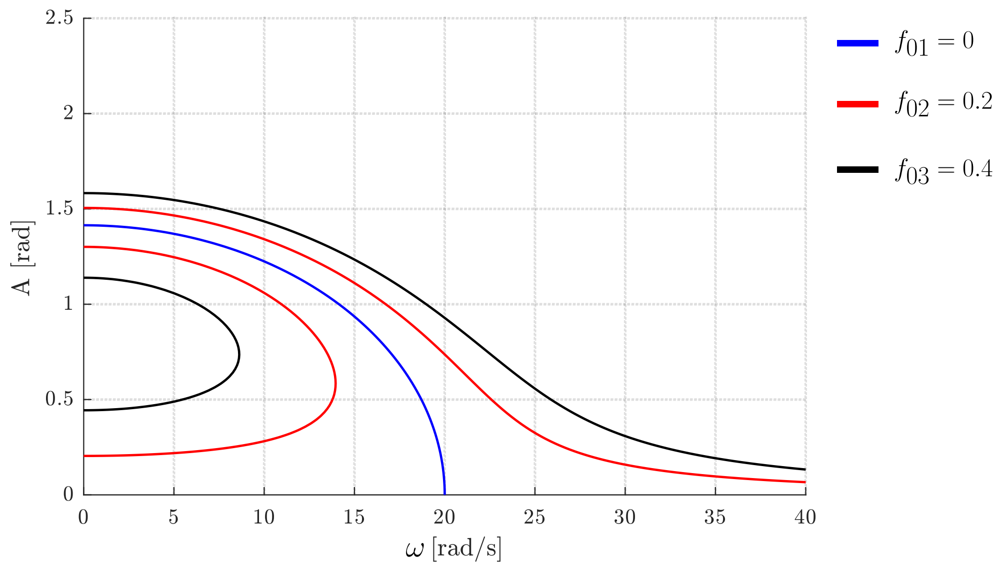


Figure 7: Amplification Diagram

Appendix: MATLAB Code

```
%%
clear all

%% Parameter Conversion and Symbol Definition
% Given parameters in centimeters
L_cm = 12;       % Length
a_cm = 12;       % Distance
r_cm = 5;        % Radius
L_min_cm = 5;    % Minimum Length
L_max_cm = 20;   % Maximum Length
m_num = 3;       % Mass [kg]
s_num = 600;     % Spring constant [N/m]

%%

% Convert parameters to meters
L_num = L_cm / 100;        % Length [m]
a_num = a_cm / 100;        % Amplitude [m]
r_num = r_cm / 100;        % Radius [m]
L_min_num = L_min_cm / 100;   % Minimum Length [m]
L_max_num = L_max_cm / 100;   % Maximum Length [m]

% Define symbols for parameters
syms L a r L_min L_max m s phi dphi ddphi

%% TASK 1
disp('-- Task 1 --')
%% Kinetic Energy and Potential function

% Define symbolic potential energy function
U = 1/2 * s * (((a - r * sin(phi))^2 + (r * (cos(phi)-1))^2)^(1/2)  - L)^2;
disp(U)
dU_dphi = diff(U, phi);
disp(dU_dphi)

dU_dphi_num = subs(dU_dphi, [L, a, r, s], [L_num, a_num, r_num, s_num]);

%% Brake 1.1
% Solve for phi
phi_solution = solve(dU_dphi_num == 0, phi , 'Real',true);
% Substitute parameter values
phi_solution_numeric = zeros(size(phi_solution)); % Initialize array to store
    numeric solutions

for i = 1:numel(phi_solution)
    phi_solution_numeric(i) = double(subs(phi_solution(i), [L, a, r, s], [L_num
    , a_num, r_num, s_num]));
end

%% Plotting the results

% Define range for phi
```

```
phi_range = linspace(-pi, pi, 500);

% Numerically substitute parameter values into U
U_numeric = subs(U, [L, a, r, s], [L_num, a_num, r_num, s_num]);

% Create array to store numeric values of U over phi_range
U_values = zeros(size(phi_range));

% Calculate U values over phi_range
for i = 1:length(phi_range)
    U_values(i) = double(subs(U_numeric, phi, phi_range(i)));
end
% Plot U as a function of phi
figure;
plot(phi_range, U_values, 'k', 'LineWidth', 1.5);  % Use black color and
    thicker line
xlabel('$\varphi$ [rad]', 'Interpreter', 'latex');
ylabel('U [Nm]', 'Interpreter', 'latex');
title('Potential Energy Function U($\varphi$)', 'Interpreter', 'latex');
%text(-pi/2, max(U_values), ['l = ', num2str(L_num)], 'HorizontalAlignment', '
    left', 'VerticalAlignment', 'top');

% Modify x-axis ticks and labels
xticks([-pi, -3*pi/4, -pi/2, -pi/4, 0, pi/4, pi/2, 3*pi/4, pi]);  % Set custom
    x-axis ticks
xticklabels({'-\pi', '-3\pi/4','-\pi/2', '-\pi/4', '0', '\pi/4', '\pi/2', '3\pi
    /4', '\pi'});  % Set custom x-axis tick labels

% Add grid
grid on;
ax = gca;
ax.GridColor = [0.5, 0.5, 0.5];  % Grey color for the grid
%% TASK 2

disp('-- Task 2 --')

%% Phase portrait - Preprocess

% Define moment of inertia
theta = 1/2 * m * r^2;

% Define kinetic energy
T = 1/2 * theta * dphi^2;

% Compute derivative of T with respect to dphi and replace dphi with ddphi
dT = diff(T, dphi);
dT = subs(dT, dphi, ddphi);

Leq = dT + dU_dphi;
Leq_simp = dU_dphi * 1/theta;

%% Phase Portrai - Plotting
% Define the Cauchy transformation
dx_dt = @(phi, y) y;
```

```
Leq_simp_num = subs(Leq_simp, [L, a, r, s, m], [L_num, a_num, r_num, s_num,
    m_num]);
v = diff(Leq_simp_num, phi);

% Convert the symbolic expression to a function handle
v_function = matlabFunction(v, 'Vars', {phi});

% Define a grid for plotting
[phi_grid, dphi_grid] = meshgrid(linspace(-20, 20, 70), linspace(-20, 20, 22));

% Compute v values on the grid
v_values = v_function(phi_grid);

% Plot the phase portrait
figure;
quiver(phi_grid, dphi_grid, dx_dt(phi_grid, dphi_grid), v_values, 'b');
xlabel('\phi');
ylabel('v (d\phi/dt)');
title('Phase Portrait');
%% Trajectories
% Number of trajectories
num_trajectories = 8;

% Generate random initial conditions within the specified mesh grid
initial_conditions = rand(num_trajectories, 2) .* [40, 40] - [20, 20];

% Define the ODE system
ode_system = @(t, y) [y(2); v_function(y(1))];

% Time span for integration
tspan = [0, 10]; % Adjust as needed

% Plot the phase portrait
figure;
quiver(phi_grid, dphi_grid, dx_dt(phi_grid, dphi_grid), v_values, 'b');
hold on;
xlabel('\phi');
ylabel('v (d\phi/dt)');
title('Phase Portrait');

% Plot random trajectories within the specified mesh grid
for i = 1:num_trajectories
    [t, y] = ode45(ode_system, tspan, initial_conditions(i,:));
    % Check if the trajectory lies within the specified mesh grid
    if all(y(:,1) >= -20) && all(y(:,1) <= 20) && all(y(:,2) >= -20) && all(y
   (:,2) <= 20)
        % Plot the trajectory in red
        plot(y(:,1), y(:,2), 'r');
    end
end
hold off;

%% TASK 3
disp('-- Task 3 --')
```

```
%% Bifurcation Diagram
% Define array to store bifurcation points and their stability
bifurcation_points = [];
stable_points = [];
unstable_points = [];

% Define L values to vary
L_values = linspace(L_min_num, L_max_num, 150);
dU_dphi_num = subs(dU_dphi, [a, r, s], [a_num, r_num, s_num]);

% Loop through L values
for L_val = L_values
    % Substitute L value into dU_dphi
    dU_dphi_L = subs(dU_dphi_num, L, L_val);

    % Solve for phi
    phi_solution = solve(dU_dphi_L == 0, phi, 'Real', true);

    % Convert symbolic solution to numeric and append to bifurcation_points
    for i = 1:numel(phi_solution)
        phi_solution_numeric = double(phi_solution(i));
        if isreal(phi_solution_numeric)
            bifurcation_points = [bifurcation_points; L_val, phi_solution_numeric];

            % Calculate second derivative of U with respect to phi
            d2U_dphi2 = diff(dU_dphi_L, phi, 1);
            d2U_dphi2_val = subs(d2U_dphi2, phi, phi_solution(i));

            % Classify stability based on the sign of second derivative
            if double(d2U_dphi2_val) > 0
                stable_points = [stable_points; L_val, phi_solution_numeric];
            elseif double(d2U_dphi2_val) < 0
                unstable_points = [unstable_points; L_val, phi_solution_numeric];
            end
        end
    end
end

% Plot bifurcation diagram
figure;
hold on;
plot(stable_points(:, 1), stable_points(:, 2), 'g.');
plot(unstable_points(:, 1), unstable_points(:, 2), 'r.');
hold off;
xlabel('L [m]');
ylabel('$\varphi$ [rad]', 'Interpreter', 'latex');
title('Bifurcation Diagram');
legend('Stable', 'Unstable');

%% ------ TASK 4 ------ %%
disp('-- Task 4 --')
%% Time period
```

```
omega_n_lin = (subs(diff(Leq_simp, phi), phi,0))^(1/2);
omega_n_lin_num = double(subs(omega_n_lin, [s, m, L, a, r],[s_num, m_num, L_num
    , a_num, r_num]));

fd3 = subs(diff(Leq_simp, phi, 3), phi,0);
mu = 1/6 * double(subs(fd3, [s, m, L, a, r],[s_num, m_num, L_num, a_num, r_num
    ]));

disp('omega_n_lin_num :')
disp(omega_n_lin_num)
disp('mu :')
disp(mu)

%% Break 4.1
syms Amp

omega_n2 = omega_n_lin_num^2 + 3/4*mu* Amp^2;
Tp = 2*pi /(omega_n_lin_num^2 + 3/4 * mu * Amp^2)^(1/2);
disp('Tp :')
disp(Tp)

%% Break 4.2
% Define the ODE function
ode_func = @(t, y) [y(2); (80000*(((sin(y(1))/20 - 3/25)^2 + (cos(y(1)) - 1)
    ^2/400)^(1/2) - 3/25)*((sin(y(1))*(cos(y(1)) - 1))/200 - (cos(y(1))*(sin(y
    (1))/20 - 3/25))/10))/((sin(y(1))/20 - 3/25)^2 + (cos(y(1)) - 1)^2/400)
    ^(1/2)];

% Define the initial conditions
initial_conditions = [pi/32; 0];

% Define the time span
t_span = [0, 1]; % Adjust the end time as needed

% Solve the ODE using ode45
[t, phi] = ode45(ode_func, t_span, initial_conditions);

% Plot the results
figure;
plot(t, phi(:, 1));
xlabel('Time (t)');
ylabel('Angle (\phi)');
title('Solution of ODE: Angle vs Time');

%%
% Define the threshold time
threshold_time_up = 0.4;
threshold_time_down = 0.2;

% Initialize variables to store the maximum x value and its corresponding time
max_x_value = 0;
max_x_time = -1;

% Iterate through the phi values
```

```
for i = 1:length(phi)
    % Check if the current time is greater than the threshold
    if t(i) < threshold_time_up
        if t(i) > threshold_time_down
        % Check if the current phi value is greater than the maximum found so
   far
            if phi(i, 1) > max_x_value
                % Update the maximum x value and its corresponding time
                max_x_value = phi(i, 1);
                max_x_time = t(i);
            end
        end
    end
end

% Display the results
if max_x_time ~= -1
    disp(['First maximum x value after x > 0.1: ', num2str(max_x_value)]);
    disp(['Time corresponding to the first maximum x value: ', num2str(
   max_x_time)]);
else
    disp('No maximum x value found after x > 0.1.');
end
FirstPoint_x = max_x_time;
FirstPoint_y = max_x_value;
disp('----')
%%
% Define the threshold times
threshold_time_up = 0.4;
threshold_time_down = 0.2;

% Define initial conditions
initial_conditions_list = [[pi/16; 0], [pi/8; 0], [3*pi/16; 0], [pi/4; 0], [5*
   pi/16; 0], [3*pi/8; 0], [7*pi/16; 0], [pi/2; 0]];

% Initialize a cell array to store max_x_time values
max_x_times = cell(size(initial_conditions_list));

% Iterate over each set of initial conditions
for i = 1:size(initial_conditions_list, 2)
    % Define the ODE function
    ode_func = @(t, y) [y(2); (80000*(((sin(y(1))/20 - 3/25)^2 + (cos(y(1)) -
   1)^2/400)^(1/2) - 3/25)*((sin(y(1))*(cos(y(1)) - 1))/200 - (cos(y(1))*(sin(y
   (1))/20 - 3/25))/10))/((sin(y(1))/20 - 3/25)^2 + (cos(y(1)) - 1)^2/400)
   ^(1/2)];

    % Define the initial conditions
    initial_conditions = initial_conditions_list(:, i);

    % Solve the ODE using ode45
    [t, phi] = ode45(ode_func, t_span, initial_conditions);

    % Initialize variables to store the maximum x value and its corresponding
   time
```

```
    max_x_value = 0;
    max_x_time = -1;

    % Iterate through the phi values
    for j = 1:length(phi)
        % Check if the current time is within the threshold
        if t(j) < threshold_time_up && t(j) > threshold_time_down
            % Check if the current phi value is greater than the maximum found
   so far
            if phi(j, 1) > max_x_value
                % Update the maximum x value and its corresponding time
                max_x_value = phi(j, 1);
                max_x_time = t(j);
            end
        end
    end

    % Store the max_x_time value
    max_x_times{i} = max_x_time;
end

% Display the max_x_times values
disp('Max_x_times for different initial conditions:');
disp(max_x_times);

%%
sim_tp_y_values = [max_x_times{1}-0.005,
    max_x_times(2,1),
    max_x_times(1,2),
    max_x_times(2,2),
    max_x_times(1,3),
    max_x_times(2,3)];

sim_tp_x_values = [pi/16,
    pi/8,
    3*pi/16,
    pi/4,
    5*pi/16,
    3*pi/8];

%% Time Period vs Amplitude plot
% Define values for amplitude A
A_values = linspace(0, 3*pi/8, 100);

% Calculate Tp for each A value
Tp_values = zeros(size(A_values));
for i = 1:length(A_values)
    Tp_values(i) = double(subs(Tp, Amp, A_values(i)));
end

% Define custom tick locations and labels for x-axis
x_ticks = [pi/32, pi/16, pi/8, 3*pi/16, pi/4, 5*pi/16, 3*pi/8, 7*pi/16, pi/2];
x_tick_labels = {'\pi/32','\pi/16', '\pi/8', '3\pi/16', '\pi/4', '5\pi/16', '3\
   pi/8', '7\pi/16', '\pi/2'};
```

```matlab

% Plot Tp as a function of A
figure;
plot(A_values, Tp_values);
xlabel('Amplitude (A) [rad]');
ylabel('Time Period (Tp) [s]');
title('Time Period vs Amplitude');
xticks(x_ticks);
xticklabels(x_tick_labels);
grid on;
grid minor;
ax = gca;
ax.GridColor = [0.5, 0.5, 0.5];

% Plot the points sim_tp_x_values and sim_tp_y_values
hold on;
for i = 1:numel(sim_tp_y_values)
     plot(sim_tp_x_values(i), sim_tp_y_values{i},'ko', 'linewidth', 2,'
    MarkerFaceColor','k', 'MarkerSize', 4);
end
plot(pi/32, FirstPoint_x+0.001, 'ko', 'linewidth', 2,'MarkerFaceColor','k', '
    MarkerSize', 4)
legend('Tp vs A', 'Simulated Tp Points');

%% Task 5
disp('-- Task 5 --')

%% Amplification diagram plot
syms Om A phi

% Constants and calculations
Leq_simp_num = subs(Leq_simp, [L, a, r, s, m], [L_num, a_num, r_num, s_num,
    m_num]);
fd3 = subs(diff(Leq_simp_num, phi, 3), phi, 0);
mu = 1/6 * subs(fd3, [s, m, L, a, r], [s_num, m_num, L_num, a_num, r_num]);
mu = double(subs(mu, m, m_num));
om2 = omega_n_lin_num^2;
alpha = double(sqrt(om2));

% Function definitions
ellipse_fun = @(x, y) x^2 / om2 - y^2 / ((4 * om2) / (3 * mu)) - 1;
linesP_fun = @(x, y, f) (3/4) * mu * y^3 + om2 * y - (f * om2 + x^2 * y);
linesN_fun = @(x, y, f) (3/4) * mu * y^3 + om2 * y - (-f * om2 + x^2 * y);

% Plotting
figure;
set(gcf, 'color', 'white');
hold on;
grid on;

xlimmax = 2 * alpha;
ylimmax = 2 * (2 * sqrt(om2) / (sqrt(3 * abs(mu))));

% Plot amplification diagrams
```

```
colors = {'b', 'r', 'k'}; % Define colors for each f element
for idx = 1:numel(colors)
    f = [0, 0.2, 0.4];
    f_ellipse = @(x, y) ellipse_fun(x, y);
    f_linesP = @(x, y) linesP_fun(x, y, f(idx));
    f_linesN = @(x, y) linesN_fun(x, y, f(idx));

    fimplicit(f_ellipse, [0, xlimmax, 0, ylimmax], colors{idx}, 'LineStyle', '
    --');
    fimplicit(f_linesP, [0, xlimmax, 0, ylimmax], colors{idx});
    fimplicit(f_linesN, [0, xlimmax, 0, ylimmax], colors{idx});
end

xlabel({'$\varphi$ [rad/s]'}, 'Interpreter', 'latex');
ylabel('A [rad]', 'Interpreter', 'latex');
set(gca, 'TickLabelInterpreter', 'latex');

%% End :) %%
```

Listing 1: Nonlinear Vibrations - Homework 2 : MATLAB code