Computational Physics Mini-Report: Damped Oscillator

Patrice Harapeti

Abstract

The investigation of damped oscillators offers physicists a steppingstone to understand and visualise more complex systems that can be described as one or more partial differential equations. A partial differential equation describing a system can often be broken up into a set of coupled ordinary equations. As PDEs become more complex, analytical solutions to such systems become impossible to derive due to non-linearity and various other factors. Due to this limitation, physicists must solve complex systems using numerical techniques which involves the discretization and quantization of domains to be input as data into a computer or algorithm. The simplest numerical method used to numerically solve coupled systems of equations includes the Euler's method. In this mini-report, Euler's method will be utilized to numerically solve for the Underdamped, Critically Damped and Overdamped cases of the damped oscillator. Validation and converge tests are run on the numerical solution against the analytical solution to ensure the numerical solution is valid and accurately represents the motion and behaviour of the damped oscillator. The report also contains exploration techniques such as Fourier Analysis to determine aspects regarding the central frequency and width of the time-varying signal.

Introduction

A damped oscillator is a type of harmonic oscillator (Harmonic Oscillator, n.d.) which possesses an identifiable characteristic known as dampening, which is resistive or frictional force that opposes the direction of motion of the oscillator and aims to minimise the total energy of the system – note that this force is non-conservative. There are namely three types/cases of damped oscillators. The type of a damped oscillator is determined by the value of the oscillator's dampening ratio - which is often expressed as the determinant in the characteristic equation which describes the partial differential.

The first case is where the determinant is positive; also known as the Overdamped case. In the Overdamped case, the system exponentially decays to its final steady state without oscillating. The larger the dampening ratio, the longer it takes for the oscillator to reach equilibrium. The following equation describes the motion of a damped oscillator in this case:

$$y = Ae^{\lambda_1 t} + Be^{\lambda_2 t}$$

The second case is where the determinant equals zero; this is known as the Critically damped case. In this case, the system reaches equilibrium as quickly possible without oscillating. The following equation described the motion of a damped oscillator in this case:

$$y = (A + Bt)e^{\lambda_1 t}$$

The final case is where the determinant is negative; also known as the Underdamped case. This case describes an oscillator which oscillates at a given frequency and eventually reaches equilibrium. The rate at which the oscillator reaches equilibrium is further determined by the dampening ratio. The following equation describes the motion of a damped oscillator in this case:

$$y = e^{rt}(A\cos s\,t + B\sin s\,t)$$

The angular frequency or natural frequency of an undamped oscillator can be determined by the following equation (Damped Oscillations, 2020):

$$\omega_0 = \sqrt{\frac{k}{m}}$$

The angular frequency of a damped oscillator is determined by: (Damped Oscillations, 2020)

$$\omega = \sqrt{\omega_0^2 - (\frac{\gamma}{2m})^2}$$

The Q-Factor or Quality Factor is a scalar value which can be computed to describe how long it takes for a given oscillator to dampen its motion to equilibrium. The following definition of the Q-Factor (Schwartz, n.d.) described how as the dampening ratio is minimised, the oscillators' ability to resist energy loss increases:

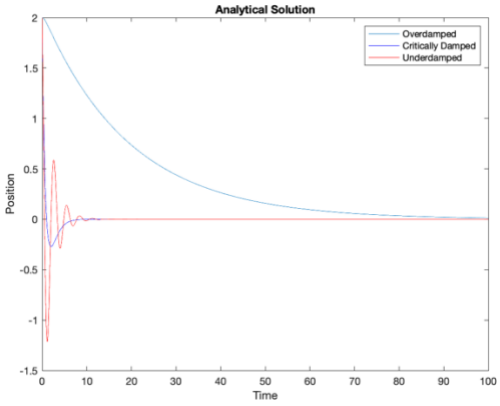
$$Q = \frac{1}{2\xi}$$

Problem Specification

The following partial differential equation describes a damped oscillator. This report will the derivation, implementation, analysis and validation of a numerical solution for this system. The derivation of ordinary differential equations of this partial differential can be found in Part 7 of Appendix A.

$$-\frac{d^2x}{dt^2} - \gamma\frac{dx}{dt} - kx = 0$$

We can visualise the behaviour of this damped oscillator by selecting three combinations of parameters (γ, k) such produce a discriminant of the characteristic equation that from each case – underdamped, critically damped and overdamped. The following graph visualises the solutions for the $\{2, 0.1\}$, $\{2, 1\}$ and $\{1, 5\}$ parameter cases.



By inspecting the graph, we can see that the Overdamped case exponentially dampens the motion of the oscillator. The Critically damped case dampens the motion at a lesser degree and allows a single period oscillation of the harmonic oscillator. Lastly, the Underdamped case minimally dampens the oscillator, and we can visually identify multiple oscillations before the oscillator reaches equilibrium.

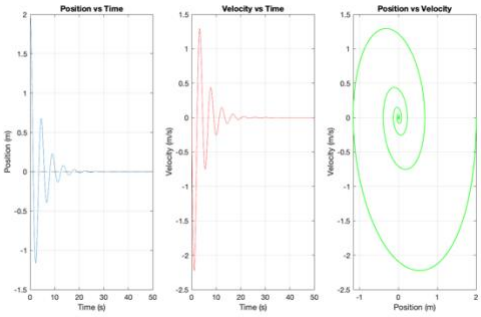
Numerical Solutions to Systems

A variety of physics problems and systems are not able to be derived in terms of an analytical answer, due to complexity or non-linearity. In an effort to understand these complex systems, physicists have turned to numerical solutions which quantize and discretize parameters and/or data, such that it can be processed in a digital system such as a computer. Since the inception of the technique to numerical solve problems, many numerical methods have gained popularity due to their ability to provide consistent and convergent solutions to a complex system. Numerical methods can also be classified in various types, but in an effort to keep this report brief selects the Euler's method as our numerical method.

The Euler's method (Euler Method, n.d.) is a widely adopted numerical method which is able to numerically solve one or more ordinary differential equations. The main advantage of using this method is the method's simplicity and ease of implementation. The Euler's method is a first-order method, which means that the local error is proportional to the square of the step size, and the global error is proportional to the step size. This means that the level of discretization used when determining a step size in the implementation of the Euler's method can be finely chosen to reach a sufficient minimum error criterion. The main limit to minimising the step size will lie in the computational power available during runtime.

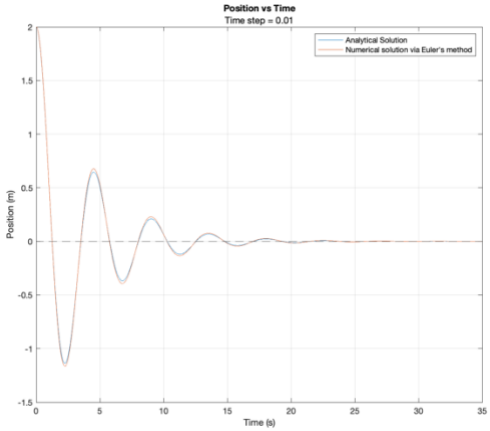
We must also consider the accuracy and validity of our numerical solution. A numerical solution is only valid if the accuracy of the solution is known, compared against other criteria such as other numerical solutions, an analytical solution, experimental results, intermediate results or convergence.

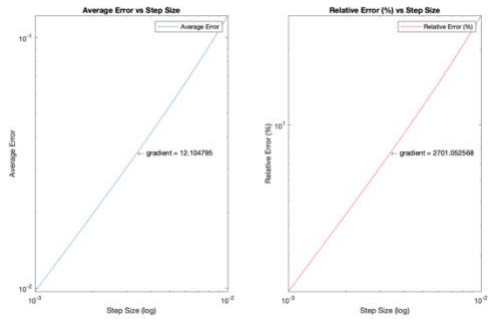
The following figure represents how the numerical solution for the Underdamped case behaves over time and space (where γ and k are 0.5 and 2 respectively). By inspecting the position vs time and velocity vs time graphs, we can clearly see how this solution reaches equilibrium at approximately 25 seconds.



Further, the Position vs Velocity phase-space diagram on the right clearly shows how the oscillator reaches equilibrium the spiralling behaviour is minimised to a point – at which the position and velocity is zero.

The following graph compares the analytical and numerical solution (from above).

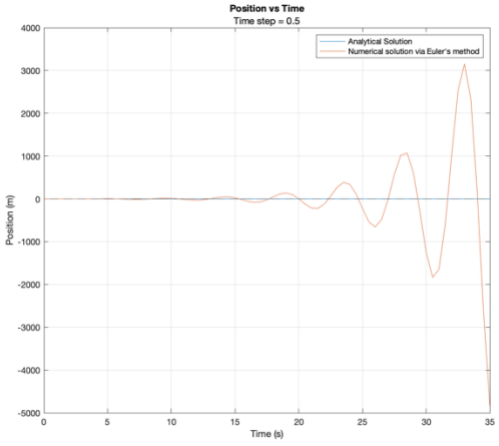




Through further inspection, we can see that the average error between the analytical and the numerical solution linearly grows as the step size increases. A logarithmic plot is used to highlight this relationship – resulting in a gradient approximately near the value of 10.

If we analyse the error between our two solutions at step sizes larger than 0.01 seconds, we find that the error grows at an exponential rate and becomes non-linear. This is due to the Euler's Method limitations of modelling the conservation of energy in a given system.

The following graph compares the analytical and numerical solution of the Underdamped case with a large step size of 0.5 seconds. As time progress, the local error calculated by the Euler's method is carried through to each sequential computation – resulting in a growing and non-linear error which does accurately model the physical system. Alternative numerical methods such as Runge-Kutta may be utilised to avoid these forms of energy conservation issues.



Validation and Convergence of Numerical Solutions

Some types of numerical solutions are best suited for particular kinds of validation (Jauregui & Silva, n.d.). For example, using an analytical solution to validate a numerical solution is only feasible if the analytical solution is simple to implement and derive. In reality, most systems are highly complex and require experimental data to sufficiently validate any of its numerical solutions. Luckily in our case, we are able to derive a simple analytical solution for our system - allowing us to compare the numerical solution against the analytical system by techniques such as convergence.

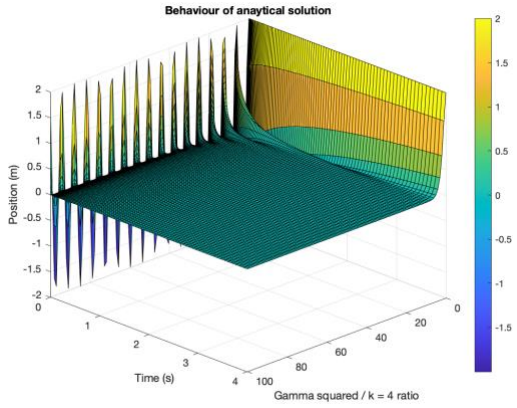
As discussed in the paragraphs above, the local error found in a numerical solution generated from the Euler's method grows proportionally with the step size used when discretizing the input dataset. Therefore, we would expect the local error to decrease as the step size decreases.

By looking at the following graph, we can see that the Average Absolute Error and Relative Error percentage both decrease as the step size is decreased. This proves that the Euler's method is an effective numerical method in solving the system, in comparison to the analytical solution. Further, as both types of error decreases as the dataset is more granularly discretized, it is clear that the analytical and numerical solution both converge.

Exploring solutions

Systems are often analysed through various techniques relevant to the type of system and behaviour present in the system.

As described above, the discriminant of the characteristic equation is the key contributor to the behaviour of the damped oscillator. By indexing over a range of parameter combination, we can gain an understanding of how the damped oscillator behaves across a range of values for γ and k. The following diagram renders a plot of the position vs time for a set of γ and k values. It is clear to see that the damped oscillator achieves equilibrium in shorter time as the $\frac{\gamma^2}{k} = 4$ ratio increases.

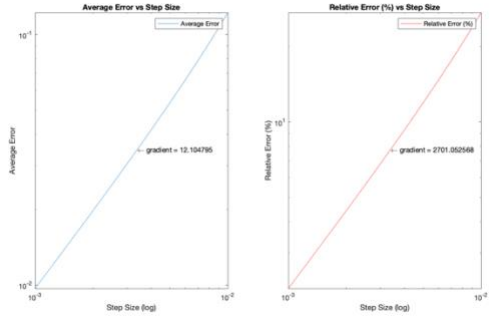


Another interesting exercise would be to investigate the behaviour of the Damped Oscillator in terms of its imaginary and real components. For example, a plot of position (imaginary) vs angle for an underdamped case would undoubtably provide another methodology to derive the frequency of the damped oscillator.

Fourier Analysis

The damped oscillator can also be analysed through the lens of Fourier Analysis. In short, Fourier Analysis is the study of representing a function as a sum of trigonometric functions, other through the Fourier Series.

Performing Fourier Analysis via MATLAB can be greatly simplified by using the Fast Fourier Transform method, which produces a Fast Fourier Transformation matrix, which can be applied to a set of input domain to transform the domain into one of frequency and space. Fourier Analysis can be used on our system to identify key features such as the power, centre frequency and decay time of our damped oscillator.



By analysing the figure above, we can see that the power of the signal oscillates from an initial peak and eventually reaches equilibrium after $t_0 + 15$ seconds. Further, we can conclude that the resonant or central frequency of the signal is approximately 49.75 Hz. This value corresponds to the resonant frequency as determined by the analytical solution for this particular case.

Appendix A: MATLAB Script

```
%% Part Zero : Setup

% Clear existing workspace
clear; clc; close all

% Setup parameters
timestep = 0.01; % timestep (seconds)
totalTime = 100; % total time of simulation (seconds)
timeSeries = 0:timestep:totalTime;

% Define initial conditions of system
xInitial = 2; % initial position (metres)
vInitial = 0; % intial velocity (metres / second)

%% Part One : Analytical Solution

% Plot analytical solution
figure('NumberTitle', 'off', 'Name', 'Analytical Solution of each case');
yline(0, '--');
grid on;

% Overdamped case
plot(timeSeries, generateAnalyticalSolution(timeSeries, 2, 0.1, xInitial));
hold on;

% Critically Damped case
plot(timeSeries, generateAnalyticalSolution(timeSeries, 2, 1, xInitial), 'Color', [0 0 1]);
hold on;

% Underdamped case
plot(timeSeries, generateAnalyticalSolution(timeSeries, 1, 5, xInitial), 'r');
title('Analytical Solution');
xlabel('Time');
ylabel('Position');
legend('Overdamped', 'Critically Damped', 'Underdamped');
hold off;

%% Part Two : Exploration of Analytical Solution
% Plot surface plot of gamma/k vs time to visualise how the
% ratio (dampening) of the parameters affect the function

% Fix value of parameter k while gamma changes
kExplore = 1;

% Determine number of points required in discretization
noPoints = 100;

timeSeriesExploration = linspace(0, totalTime, noPoints);
gammaSeries = linspace(0, 2, noPoints);
ratioSeries = nan(size(gammaSeries));
positionSeries = nan(size(gammaSeries));

% Compute position and time data for each gamma value
for i = 1:length(gammaSeries)
```

```
    % Find gamma value for iteration
    gamma = gammaSeries(i);

    % Calculate parameter ratio using gamma and kExplore
    ratioSeries(i) = gamma.^2 ./ kExplore;

    % Calculate position based on time and gamma ratio
    positionSeries(i, :) = generateAnalyticalSolution(timeSeriesExploration, gamma, kExplore, xInitial);
end

% Plot ratio of parameters vs position and time
figure('NumberTitle', 'off', 'Name', 'Function Behavioural Analysis');
surf(timeSeriesExploration, ratioSeries, positionSeries);

% Decorate surface plot
colorbar
title('Behaviour of anaytical solution')
xlabel('Gamma squared / k = 4 ratio');
ylabel('Time (s)');
zlabel('Position (m)');

% Adjust camera viewport
%view([-15 3 4]);

%% Part Three : Numerical Solution using the Euler's Method

% Generate numerical solution for the underdamped case
[numericalPosition, numericalVelocity] = generateNumericalSolution(timeSeries, 0.5, 2, xInitial, vInitial,
timestep);

% Plot Position vs Time
figure('NumberTitle', 'off', 'Name', 'Numerical Solution of Underdamped case');
subplot(1, 3, 1);
plot(timeSeries, numericalPosition);

% Draw horizontal line at y = 0 to represent convergence value
yline(0, '--');
grid on;
title('Position vs Time');
xlabel('Time (s)');
xlim([0, 50]);
ylabel('Position (m)');

% Plot Velocity vs Time
subplot(1, 3, 2);
plot(timeSeries, numericalVelocity, 'r');
grid on;
title('Velocity vs Time');
xlabel('Time (s)');
xlim([0, 50]);
ylabel('Velocity (m/s)');

% Plot Position vs Velocity
subplot(1, 3, 3);
plot(numericalPosition, numericalVelocity, 'g', 'LineWidth', 1.2);
grid on;
```

```
title('Position vs Velocity');
xlabel('Position (m)');
ylabel('Velocity (m/s)');

%% Part Four : Comparing Analytical and Numerical Solution

gammaComparison = 0.5;
kComparison = 2;

% Generate analytical solution for the underdamped case
analytical_position = generateAnalyticalSolution(timeSeries, gammaComparison, kComparison, xInitial);

% Generate numerical solution using same parameters
[numericalPosition, numericalVelocity] = generateNumericalSolution(timeSeries, gammaComparison,
kComparison, xInitial, vInitial, timestep);

% Plot Position vs Time
figure('NumberTitle', 'off', 'Name', 'Analytical vs Numerical Side by Side');
plot(timeSeries, analytical_position, timeSeries, numericalPosition);

% Draw horizontal line at y = 0 to represent convergence value
yline(0, '--');
grid on;
title('Position vs Time');
subtitle("Time step = " + timestep);
xlabel('Time (s)');
xlim([0, 35]);
ylabel('Position (m)');
legend('Analytical Solution', "Numerical solution via Euler's method");

%% Part Five : Analyis of error with varying step size

% Pick particular case, underdamped in this case
gammaErrorAnalysis = 0.1;
kErrorAnalysis = 3;

% Generate range of step sizes
stepSizes = logspace(-3, -2);
averageErrorAtStepSize = nan(size(stepSizes));
relativePercentErrorAtStepSize = nan(size(stepSizes));

% Loop over each discretized step size
for i = 1:length(stepSizes)
    % Generate discretized time series for total time based on step size
    varyingTimeSeries = 0:stepSizes(i):totalTime;

    % Generate analytical solution with timeseries
    analyticalPos = generateAnalyticalSolution(varyingTimeSeries, gammaErrorAnalysis, kErrorAnalysis,
xInitial);

    % Generate numerical solution with same timeseries
    [numericalPos, ~] = generateNumericalSolution(varyingTimeSeries, gammaErrorAnalysis, kErrorAnalysis,
xInitial, vInitial, stepSizes(i));

    % Calculate error at the current step size
    [averageError, relativeError] = calculateError(analyticalPos, numericalPos);
```

```
    averageErrorAtStepSize(i) = averageError;
    relativePercentErrorAtStepSize(i) = relativeError .* 100;
end

% Determine gradient of average error vs stepsize
coefficients = polyfit(stepSizes, averageErrorAtStepSize, 1);
averageErrorGradient = coefficients(1);

% Determine gradient of relative error vs stepsize
coefficients = polyfit(stepSizes, relativePercentErrorAtStepSize, 1);
relativeErrorGradient = coefficients(1);

% Plot each type of error vs step size
% We expect the error to be reduced as the step size is minimised
figure('NumberTitle', 'off', 'Name', 'Error Analysis');
subplot(1, 2, 1);
loglog(stepSizes, averageErrorAtStepSize);
title('Average Error vs Step Size');
xlabel('Step Size (log)');
ylabel('Average Error');
legend('Average Error');
gradientText = sprintf('\\leftarrow gradient = %f', averageErrorGradient);
text(0.0034, 0.0348, gradientText);

subplot(1, 2, 2);
loglog(stepSizes, relativePercentErrorAtStepSize, 'r');
title('Relative Error (%) vs Step Size');
xlabel('Step Size (log)');
ylabel('Relative Error (%)');
legend('Relative Error (%)');

%% Part Six : Fourier Analysis of numerical solution

% Determine equation parameters for the underdamped case
gammaFourier = 0.5;
kFourier = 2;

% Generate positions of analytical solution
analytical_position = generateAnalyticalSolution(timeSeries, gammaFourier, kFourier, xInitial);

% Calculate power
powerReal = (abs(analytical_position) .^ 2);

% Plot real power of analytical solution vs time
figure('NumberTitle', 'off', 'Name', 'Fourier Analysis of underdamped case');
subplot(1, 3, 1);
plot(timeSeries, powerReal, 'Color','#008000');
title('Power vs Time');
xlim([0, 17]);
xlabel('Time (s)');
ylabel('Power');
legend('Power');

% Perform Fast Fourier Transform on Analytical Solution
N = length(analytical_position); % Number of samples
Y = fft(analytical_position); % Compute Fast Fourier Transformation
```

```
% Y = fftshift(analytical_position); % shift frequencies
Y = Y/N; % normalize for number of samples
dx = mean(diff(timeSeries)); % Determine sample spacing
df = 1/(N*dx); % Determine frequency spacing
fi = (0:(N-1)) - floor(N/2); % Generate unfolded index
frequency = df * fi; % Generate frequency vector
powerFreq = abs(Y) .^ 2; % Calculate absolute power in frequency space

% Plot Power vs Frequency
subplot(1, 3, 2);
plot(frequency, powerFreq);
title('Power vs Frequency');
xlabel('Hz');
xlim([-60, 60]);
ylabel('Power');
legend('Power');

% Plot power (frequency domain) vs frequency for positive frequencies
subplot(1, 3, 3);
frequencyPositive = frequency .* (frequency > 0);
powerPositive = powerFreq .* (powerFreq > 0);
plot(frequencyPositive, powerPositive, '-r');
title('Power vs Frequency');
subtitle('For positive frequencies');
xlabel('Hz');
xlim([48, 51]);
ylabel('Power');
legend('Power');
hold on;

% Estimate center frequency in frequency domain and include in plot
powerPositiveSum = sum(powerPositive);
weightedPowerSum_freq = sum(powerPositive .* frequency.');
expectedCenterFrequency_freq = weightedPowerSum_freq ./ powerPositive;

% Determine FWHM and include in plot


%% Part Seven : Function Definitions

% This function generates a numerical solution, given minimal parameters
function position = generateAnalyticalSolution(timeSeries, gamma, k, x_init)
    %   Derivation
    %   Let x = e^bt
    %   therefore... xdot = b * e^bt
    %   therefore... xddot = b^2 * e^bt
    %
    %   Plugging into the original PDE give us...
    %   -b^2 * e^bt - (gamma * b * e^bt) - ke^bt = 0
    %
    %   Pull e^bt out as common factor, this leaves us...
    %   e^bt (-b^2 - gamma*b - k) = 0
    %
    %   Therefore b^2 + gamma*b + k must equal 0
    %   Solving for b
    %   b = (gamma ± sqrt((gamma)^2 - (4k)) / -2
```

```
    %
    %   Overdamped when...          gamma^2 - 4k > 0
    %   Critically damped when...   gamma^2 - 4k = 0
    %   Underdamped when...         gamma^2 - 4k < 0

    % Calculate roots of characteristics equations
    b_1 = (-gamma + sqrt(gamma.^2 - (4 .* k))) ./ 2;
    b_2 = (-gamma - sqrt(gamma.^2 - (4 .* k))) ./ 2;

    % Define discriminant of characteristic equation
    discriminant = gamma.^2 - (4 .* k);

    % Define function in three cases based on the determinant of the roots
    % Reference: https://nrich.maths.org/11054
    if discriminant == 0 % critically damped
        % Solve for A and B constants
        A = x_init;
        B = - A .* b_1;

        position = (A + (B .* timeSeries)) .* exp(b_1 .* timeSeries);
    elseif discriminant > 0 % overdamped
        % Solve for A and B constants
        A = (x_init .* b_2) ./ (b_2 - b_1);
        B = (x_init .* b_1) ./ (b_1 - b_2);

        position = (A .* exp(b_1 .* timeSeries)) + ...
            (B .* exp(b_2 .* timeSeries));

    else % underdamped if discriminant is less than 0
        % Separate real and imaginary parts of roots
        alpha = real(b_1);
        beta = imag(b_2);

        % Solve for A and B constants
        A = x_init;
        B = (-A .* alpha)./ beta;

        position = exp(alpha .* timeSeries) .* ...
            (A.*cos(beta.*timeSeries) + B.*sin(beta.*timeSeries));
    end
end

% This function computes a numerical solution for a given dataset via the
% Euler's method.
function [position, velocity] = generateNumericalSolution(timeSeries, gamma, k, xInitial, vInitial, stepSize)
    % Generate vectors
    position = nan(size(timeSeries));
    velocity = nan(size(timeSeries));

    % Define initial values
    position(1) = xInitial;
    velocity(1) = vInitial;

    % Inspired by http://www.astro.yale.edu/coppi/astro520/solving_differential_equation.pdf
    % Numerically solve the position and velocity of the model system using
    % function arguments
```

```
    for i = 1:length(position) - 1
        acceleration = -(k .* position(i)) - (gamma .* velocity(i));
        velocity(i + 1) = velocity(i) + (stepSize .* acceleration);
        position(i + 1) = position(i) + (stepSize * velocity(i));
    end
end


% This function calculates the average absolute and relative error between
% two solutions
function [averageError, relativeError] = calculateError(analyticalSolution, numericalSolution)
    % Absolute error between analytical and numerical solution
    absoluteError = abs(analyticalSolution - numericalSolution);

    % Calculate Average Absolute Error
    % Reference: https://sutherland.che.utah.edu/wiki/index.php/Iteration_and_Convergence
    averageError = norm(absoluteError) ./ sqrt(length(analyticalSolution));

    % Calculate Relative Error
    relativeError = norm(absoluteError) ./ norm(abs(analyticalSolution));
end
```

References

Barnes, C. (1987). *Doing Classical Mechanics on a computer - II*. Retrieved from Yale Department of Astronomy: http://www.astro.yale.edu/coppi/astro520/solving_differential_equation.pdf

Euler Method. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Euler_method

Grayling, M. (2014, September). *Second Order Differential Equations* . Retrieved from NRICH: https://nrich.maths.org/11054

Harmonic Oscillator. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Harmonic_oscillator

Jauregui, R., & Silva, F. (n.d.). *Numerical Validation Methods*. Retrieved from Numerical Validation Methods : https://core.ac.uk/download/pdf/41765513.pdf

SUTHERLAND, J. C. (2009). *The University of Utah*. Retrieved from Iteration and Convergence: https://sutherland.che.utah.edu/wiki/index.php/Iteration_and_Convergence