**Computational Physics Mini-Report: Damped Oscillator**

Patrice Harapeti

**Abstract**

**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. There are namely three types 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:

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:

The last 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:

The angular frequency of an undamped oscillator can be determined by the following equation:

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 described how as the dampening ratio is minimised, the oscillators’ ability to resist energy loss increases:

**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.

**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.

**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 graph below, 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. In our case, the damped oscillator can 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.

Discuss FWHM and frequency (compare this with calculated oscillation frequency from above)

**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

x\_initial = 2; % initial position (metres)

v\_initial = 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, x\_initial));

hold on;

% Critically Damped case

plot(timeSeries, generateAnalyticalSolution(timeSeries, 2, 1, x\_initial), 'g');

hold on;

% Underdamped case

plot(timeSeries, generateAnalyticalSolution(timeSeries, 1, 5, x\_initial), '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, x\_initial);

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

[numerical\_position, numerical\_velocity] = generateNumericalSolution(timeSeries, 0.5, 2, x\_initial, v\_initial, timestep);

% Plot Position vs Time

figure('NumberTitle', 'off', 'Name', 'Numerical Solution of Underdamped case');

subplot(1, 3, 1);

plot(timeSeries, numerical\_position);

% 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, numerical\_velocity, '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(numerical\_position, numerical\_velocity, 'g', 'LineWidth', 1.2);

grid on;

title('Postion vs Velocity');

xlabel('Position (m)');

ylabel('Velocity (m/s)');

%% Part Four: Comparing Analytical and Numerical Solution

% Generate analytical solution for the underdamped case

analytical\_position = generateAnalyticalSolution(timeSeries, 0.5, 2, x\_initial);

% Generate numerical solution using same parameters

[numerical\_position, numerical\_velocity] = generateNumericalSolution(timeSeries, 0.5, 2, x\_initial, v\_initial, timestep);

% Plot Position vs Time

figure('NumberTitle', 'off', 'Name', 'Analytical vs Numerical Side by Side');

plot(timeSeries, analytical\_position, timeSeries, numerical\_position);

% 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 = linspace(0.001, 1);

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, x\_initial);

% Generate numerical solution with same timeseries

[numericalPos, ~] = generateNumericalSolution(varyingTimeSeries, gammaErrorAnalysis, kErrorAnalysis, x\_initial, v\_initial, stepSizes(i));

% Calculate error at the current step size

[averageError, relativeError] = calculateError(analyticalPos, numericalPos);

averageErrorAtStepSize(i) = averageError;

relativePercentErrorAtStepSize(i) = relativeError .\* 100;

end

% 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');

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, x\_initial);

% Define new domain to transformed into frequency space

x = linspace(-1,1,length(timeSeries)).' \* 10;

power\_real = abs(analytical\_position).^2;

% Plot real power of analytical solution vs x

figure('NumberTitle', 'off', 'Name', 'Fourier Analysis of critically damped case');

subplot(1, 3, 1);

plot(x, power\_real, 'Color','#008000');

title('Power vs x');

xlim([-3, 3]);

xlabel('x');

ylabel('Power');

legend('Power');

% Perform Fast Fourier Transform on Analytical Solution

N = length(x); % Number of samples

Y = fft(analytical\_position); % Compute Fast Fourier Transformation

dx = mean(diff(x)); % 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

power\_freq = abs(Y) .^ 2; % Calculate absolute power in frequency space

% Plot Power vs Frequency

subplot(1, 3, 2);

plot(frequency, power\_freq);

title('Power vs Frequency');

xlabel('Hz');

ylabel('Power');

legend('Power');

% Limit power (in frequency domain) to be positive for frequency analysis

frequencyPositive = frequency .\* (frequency > 0);

powerPositive = power\_freq .\* (power\_freq > 0);

% Plot power (frequency domain) vs frequency for positive frequencies

subplot(1, 3, 3);

plot(frequencyPositive, powerPositive, '-r');

title('Power vs Frequency');

subtitle('For positive frequencies');

xlabel('Hz');

xlim([-100, 500]);

ylabel('Power');

legend('Power');

hold on;

% Estimate center frequency in frequency domain and include in plot

powerSum\_freq = sum(powerPositive);

weightedPowerSum\_freq = sum(powerPositive .\* frequencyPositive.');

expectedCenterFrequency\_freq = weightedPowerSum\_freq ./ powerSum\_freq;

% 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> (Grayling, 2014)

if discriminant == 0 % critically damped

% Solve for A and B constants

A = x\_init;

B = x\_init .\* 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 = -x\_init ./ 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, x\_initial, v\_initial, stepSize)

% Generate vectors

position = nan(size(timeSeries));

velocity = nan(size(timeSeries));

% Define initial values

position(1) = x\_initial;

velocity(1) = v\_initial;

% Inspired by http://www.astro.yale.edu/coppi/astro520/solving\_differential\_equation.pdf (Barnes, 1987)

% 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> (SUTHERLAND, 2009)

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