



UNIVERSITY
of York

Department of Electronic Engineering

BEng Project Report

2019/20

Benjamin Duncan

Sounds Weird! Unphysical Modelling of Musical
Instruments

Supervisors:

Dr Frank Stevens

Dr Helena Daffern

Abstract

Physical modelling synthesis has historically been designed with accuracy, stability and efficiency as measurements of success. This study takes a different approach by attempting to use mathematical models to create unphysical behaviour. Primarily, this was implemented by considering modal analysis of an N-degrees of freedom mass-spring system in MATLAB. Using an approach in three phases, key functions were created and tested, eventually leading to the development of an interactive app. Unphysical behaviour was introduced in several ways. First of all, novel Vector Profiles for initial conditions and mass, spring and damping were designed and tested. Then, by using the amplitude response of all masses, the mass 'pickup' was moved using sweeps and with a feedback-type character by using the response of a mass to describe the movement of the pickup position within the system. The results for changes to the parameters of the system resulted in a range of interesting sounds, and the moving pickup demonstrated a strong baseline of additional sound possibilities. Overall, the generation of sounds greatly benefited from a strong interactive application, along with useful features such as importing and exporting use cases, user presets, and clear plotted visualisations.

Key Notation

Notation	Definition
ζ	Damping coefficient
ω	Natural Frequency (radians per second)
ω_d	Damped frequency (radians per second)
Φ	Normalized Modal Mass Matrix
Ψ	Eigen Vector
τ	Tension
ρ	Density
λ	Eigen Value
x	Displacement
v, \dot{x}	Velocity (ms^{-1})
a, \ddot{x}	Acceleration (ms^{-2})
f_s	Sample Rate (Hz)
x	Displacement (m)
t	Time (s)
$x(t)$	Displacement at time t .
m	Mass (kg)
k	Spring stiffness constant (Nm)
F	Force (N)
$[M]$	Mass Matrix
$[K]$	Stiffness Matric
$[C]$	Damping Matrix
$\{F\}$	Force Vector
$\{x\}$	Displacement Vector
$\{v\}$	Velocity Vector

1 CONTENTS

2	Introduction.....	1
2.1	Mass-Spring Systems	2
2.2	An Unphysical Model	3
3	Literature Review.....	3
3.1	Historical Context	4
3.2	Physical Modelling	6
4	Software Implementation	11
5	Theoretical Background.....	12
5.1	Modal Analysis	12
1.2	Normal Mode Shapes	16
6	Methodology.....	17
6.1	Aims	17
6.2	Phase One: Initial Experiments.....	18
6.3	Phase Two: Interactive App	20
6.4	Phase Three: Unphysical Model	24
6.5	Planning	28
7	Phase One: Initial Experiments.....	31
7.1	Overview of Experiments	31
8	Phase Two: Interactive App.....	32
8.1	Design Interactive System and Integrate System Components	32
8.2	Vibrating String	37
9	Phase Three: Unphysical Model.....	40
9.1	Design Unphysical Behaviour and Integrate Components	40
9.2	Unphysical Behaviour.....	43
9.3	Combined Unphysical	49
10	Further Work	51
10.1	Pickup improvements.....	51
10.2	Linear time-variant (LTV) parameters.....	52
10.3	Forced excitation	52
10.4	Efficiency Improvements.....	53
10.5	Listening Tests	53
11	Conclusion.....	54
11.1	Planning	54
11.2	Implementation and Iteration	54
11.3	Demonstration of Theory	54
11.4	Testing, Analysis and Discussion	55
11.5	Review of Project Impact / Viability.....	55
12	References	56
13	Statement of Ethics	58
Appendix A: Experiments		59
A.1	Experiment Set 1.....	59
A.2	Experiment Set 2.....	62
A.3	Experiment Set 3.....	67
A.4	Experiment Set 4.....	73
Appendix B: MATLABFunctions.....		76
B.1	pm_mdof_free_vibration.m	76
B.2	pm_mdof_free_vibration_modified.m.....	76
B.3	pm_mdof_create_kmatrix_linear.m	77

B.4	m_mdof_free_damped_vibration.m	78
B.5	pm_mdof_create_zeta_vector.m	79
B.6	pm_mdof_triangle_pluck_condition.m	80
B.7	pm_create_initial_condition_vector.m	80
B.8	pm_mdof_format_fft.m	83
B.9	pm_mdof_time_variant_pickup.m	83
B.10	pm_analyse_audio.m	85
Appendix C: Forced Excitation.....		86
Appendix D: Table of Vector Profiles		87

2 INTRODUCTION

This project focuses on what is described as unphysical modelling. Essentially, an unphysical model is any system which would be impossible, impractical or expensive to create in the real world. This concept contrasts with physical modelling, which is typically concerned with the accurate representation of real-world objects or systems [1]. However, an unphysical model can use the same principles as a physical model, so it is essential to fully understand the concepts of physical modelling.

For physical modelling, one area of interest is in the simulation of musical instruments, commonly known as physical modelling synthesis, which uses mathematical principles to describe the physical properties of a given instrument and ultimately create sound. In this respect, many physical modelling synthesisers can be said to operate from first principles, wholly modelling the sounds generated, as opposed to using recorded samples of an instrument [2, p. 554].

For many physical modelling instruments, there are three features to their produced sound [3]:

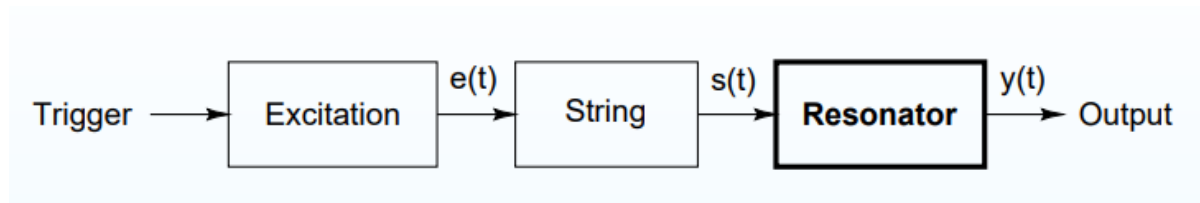


Figure 2.1: Block diagram showing the key components of a string physical model [3].

1. An excitation input, which can include any force applied or initial conditions. Examples of excitation include striking a drum membrane, blowing the reed of a woodwind instrument or bowing a violin.
2. A resonant behaviour, which will generally describe the system's emphasis on certain frequencies and harmonics. This distribution of energy can drastically transform an input, creating rich, complex timbres as well as acoustic amplification. Some instruments have several resonating or vibrating components: a piano has both strings and a surrounding soundboard contributing to its sound.

3. An output, which can include the transfer of energy from a vibrating body to the surrounding air, and ultimately to the listener's ear. An output can also include some form of recording using a microphone, an electric guitar pickup or a contact microphone for more direct measurement of vibration.

An additional final component to a physical model can sometimes be additional resonant behaviour other than the instrument itself, such as room reverberation.

2.1 MASS-SPRING SYSTEMS

One fundamental component of this project is the mass-spring approximation of a continuous system [4]. This assumption allows a vibrating system, such as a string, to be modelled using a network of interconnected masses and springs [5]. The number of masses in a system corresponds with its number of degrees of freedom (DOF). Such a system will demonstrate equilibrium; therefore, some force or displacement must be applied to generate a response or vibration. Also, assumptions for transfer of energy result in a decay in the signal, meaning a mass-spring system should have some form of decaying response.

Given an excitation, a mass-spring system will have a vibrational response, where the motion of each mass will generate forces in adjacent masses. There can also be fixed points, creating boundary conditions for the vibration and therefore, reflections [6]. This motion can be modelled through discrete time steps, resulting in a response for each mass over time.

Another critical consideration for any physical model is how its output is chosen, and for a mass-spring system, there is a simple solution: the displacement-time response of a chosen mass. Given the correct mass and spring stiffness values and an appropriate sample rate, this response can translate directly into a digital sound. Additionally, a complex system can offer a variety of sounds depending on which mass is used as the output.

2.2 AN UNPHYSICAL MODEL

There is potential to move from the paradigm of the physical model, where accuracy, stability and efficiency are measurements of success [7], to an unphysical model. The Physical characteristics of mass-spring systems can be interfered with to generate new behaviours and sounds. We can refer to Figure 2.1 to consider ways to introduce unphysical behaviour in each block.

1. Other than external forces, vibration can be achieved by describing the position and velocity of every mass in a system, typically at $t = 0$. It is, therefore, possible to describe a displacement vector of every mass that would be difficult to achieve. The closest comparison could be the effects of overtones in string instruments, which are used emphasise specific frequencies in the harmonic series.
2. Rather than assuming mass, spring and damping values are the same, abstract descriptions of these parameters could be created. Such a mass-spring system would no longer attempt to approximate a string or any one-dimensional physical object.
3. While an instrument's output could be described as the sound captured by a static microphone, there could be some function that describes how the microphone moves over time. In the mass-spring system, this might be varying the chosen displacement-time response for a given mass over time.

However, it is not always necessary to consider the real-world counterpart to an unphysical model, and some additional exploration can be achieved with a creative approach.

3 LITERATURE REVIEW

Before evaluating Physical Modelling techniques, and by extension, Unphysical Modelling, it is useful to consider the relevant physics fundamentals, primarily in relation to a vibrating string.

3.1 HISTORICAL CONTEXT

Physical modelling can be traced back to several important discoveries and their corresponding equations, beginning with Hooke's Law for an ideal spring, discovered in 1660 and published in 1678:

$$F = -kx \quad (3.1)$$

Isaac Newton soon after formed the basis for classical mechanics in his 1687 work "*Philosophiæ Naturalis Principia Mathematica*".

$$\text{Newton's First Law:} \quad \sum F = 0 \quad (3.2)$$

$$\text{Newton's Second Law} \quad F = ma \quad (3.3)$$

The modelling of a vibrating string was first considered by Daniel Bernoulli in 1727, with the idea that it could be approximated and discretised as a finite number of equally spaced beads connected by elastic springs and considered by extension, to consist of an infinite number of beads. In 1742, he proposed a theory that an acoustic vibration should be considered as a superposition of simple modes and expressed mathematically as the sum of sinusoids. This was later extended by Joseph Fourier, whereby any signal is composed of the sum of sinusoids with varying frequencies, amplitudes and phases [4].

In 1746, the mathematical solution for a vibrating string, based on initial and boundary conditions, was published by Jean D'Alembert as an essay in the "*Réflexions sur la cause générale des vents*" using partial differential calculus, and is known as the one dimensional (1-D) wave equation and uses principles by Brooke Taylor and Newton's Second Law Eq. (3.3).

For a displaced string (Figure 3.1) with tension τ and density ρ , $y(\mathbf{x}, \mathbf{t})$ is the transverse displacement of the string at position x and time t [8].

$$\frac{\partial^2 y}{\partial t^2} = c^2 \frac{\partial^2 y}{\partial x^2}, \quad c^2 = \frac{\tau}{\rho} > 0 \quad (3.4)$$

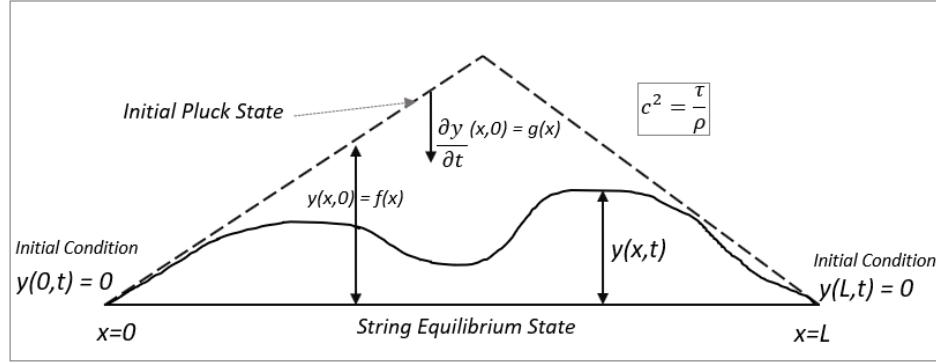


Figure 3.1: Plucked String governed by D'Alembert Eq. (3.4)

The boundary condition (fixed at both ends) (3.5) and given initial conditions (3.6), (3.7).

$$y(0,t) = y(L,t) = 0 \quad (3.5)$$

$$y(x,0) = f(x) \quad (3.6)$$

$$\frac{\partial y}{\partial t}(x,0) = g(x) \quad (3.7)$$

Where f and g are arbitrary functions, for right and left travelling waves, travelling in the $+x$, $-x$ directions at speed of c [9].

$$y(x,t) = f(x-ct) + g(x+ct) \quad (3.8)$$

And the 1-D Wave Equation:

$$y(x,t) = \frac{1}{2}y_0(x-ct) + \frac{1}{2}y_0(x+ct) + \frac{1}{2c} \int_{x-ct}^{x+ct} v_0(s) ds \quad (3.9)$$

Another milestone in sound synthesis physical modelling can be drawn from the work published in 1877 by Lord Rayleigh in the “*The Theory of Sound*”, in which he detailed the physical mathematics of vibrating systems including strings, plates, bars and membranes and how mass, stiffness and damping interrelate. A central feature of his work was the concept of modal analysis, or vibrating modes, which he applied to both to undamped and proportionally viscous damped systems. These theories were further developed in the 19th century, by, amongst others, Hermann von Helmholtz,

in 1862, with his work “*On the Sensations of Tone*” described apparatuses known as Helmholtz resonators, which were used to identify the presence of specific frequencies in complex sounds.

3.2 PHYSICAL MODELLING

Moving forward to the 20th century, physical modelling synthesis was introduced, and several milestones were reached:

- 1961: Kelly, Lochbaum and Mathews created what is considered the first computer-based application of physical modelling synthesis using vowel data synthesized as speech [10].
- 1971: Hiller and Ruiz developed the first digital string using computational methods based on mass and spring meshes [11].
- 1983: Karplus and Strong; plucked string digital synthesis [12].
- 1983: Jaffe and Smith further developed the Karplus-Strong Plucked-String Algorithm[13].
- 1987: Julius O. Smith provided a generalization of the Karplus-Strong model, introducing digital waveguides [14].
- 1994: Yamaha worked closely with the Centre for Computer Research in Music and Acoustics (CCRMA) at Stanford University to develop the first commercial physical modelling instrument, the Yamaha VL1 virtual acoustic modelling synthesiser, using waveguide synthesis [15].

1.1.1 Karplus Strong Plucked String Synthesis

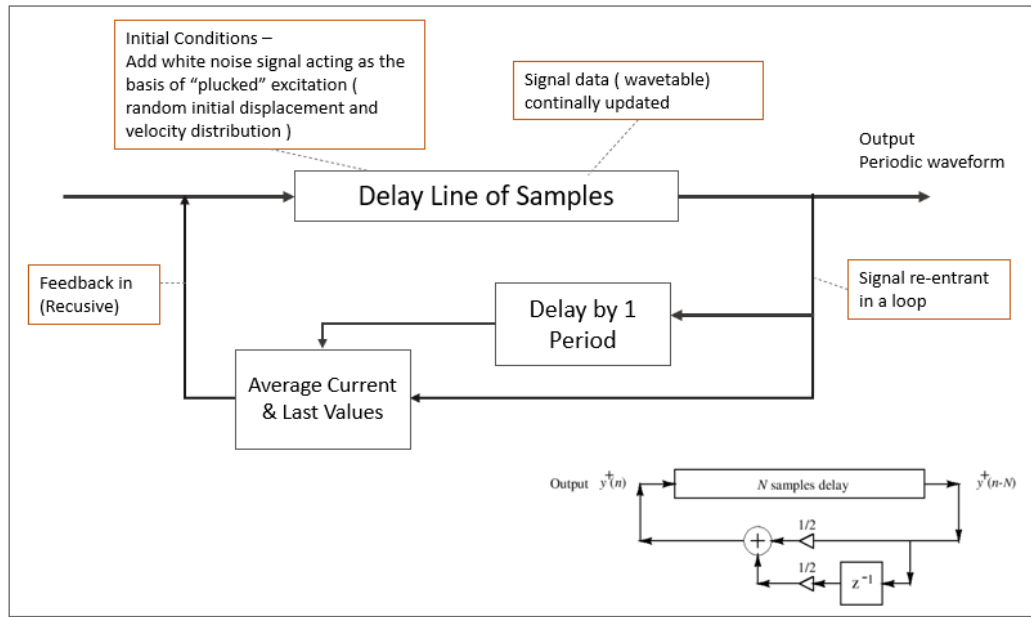


Figure 3.2: Karplus-Strong Algorithm

The Karplus-Strong algorithm (Figure 3.2) is a simple and effective process designed to computationally generate a plucked string sound synthetically. Starting with an initial data set consisting of a random white noise signal in the range -1, +1, it feeds into a moving average process by combining the current and previous versions of the signal by adding the upper and lower delay lines and acts as a low pass cut-off signal filter resulting in an efficient plucked string sound.

1.1.2 Digital Waveguide and Rigidly Terminated Ideal String

D'Alembert's 1-D wave equation of plucked string was shown to be the superposition of the left and right travelling waves (Eq. 3.8). As it is fixed at both ends the waves reflected at the boundaries are considered as travelling in a closed recursive loop. The 1-D digital waveguide (DWG) uses this method which translates to a pair of bidirectional delay lines to simulate the progressive waves (Figure 3.3) [14].

Algorithmically, it is a simple and efficient method and is typically implemented using a computer memory circular buffer where the signal sample involves a memory read, memory write and advancements of a pointer. The buffers (delay lines) are

initialised with half of the amplitude of intended string displacement (as the overall output is summed) and the initial displacements are also band limited to half frequency sample rate.

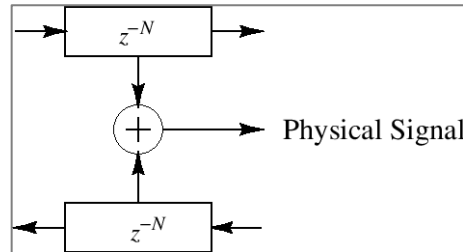


Figure 3.3: Lossless DWG with summed output

DWGs may not necessarily be useful for unphysical modelling due to how a delay line assumes a homogenous medium and would have to account for wave scattering due to a change in medium. For example, a string which has increasing density along its length would have to be discretised into many delay lines, potentially making the waveguide’s efficiency redundant

1.1.3 Lumped Networks

Another class of physical models used in sound synthesis are lumped physical models (or discrete models) which consist of masses, springs, dampers (dashpots) and non-linear components.

The modelling aim is to create a simplified version of a system by dividing into discrete parts to model an approximation of the system (as theorized by Bernoulli).

Typically, the models make assumptions that certain physical properties are unimportant, such as an “ideal” spring having elastic properties but no mass. The effect of such trade-offs is that equations for lumped network systems can be described with coupled ordinary differential equations (ODE) instead of partial differential equations (PDE), making lumped physical models easier to analyse than distributed systems [7].

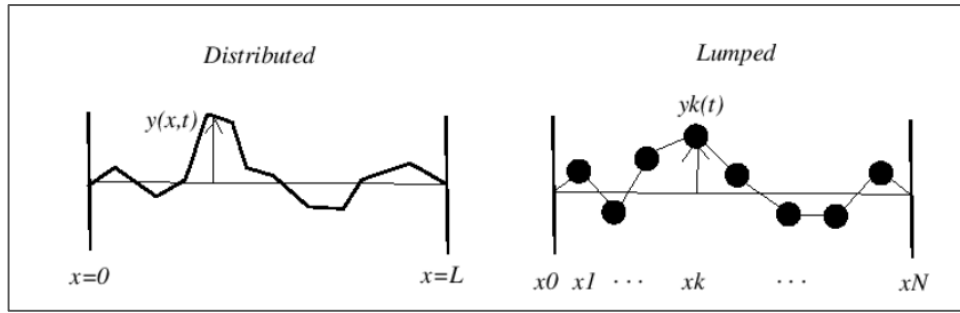


Figure 3.4: Distributed and Lumped Systems [7]

Practical implementation of this mass-interaction paradigm include CORDIS-ANIMA, introduced in 1993, as a real-time modelling and simulation system [16] and mass-interaction physical models into the FAUST programming language [17] and gesture-based virtual instruments [18].

For this project, lumped networks, in combination with modal analysis, provide flexibility, and the ability to create a wide variety of one-dimensional systems, at the expense of performance.

1.1.4 Modal Analysis and Modal Synthesis

Modal analysis aims to determine the features of physical systems through the mathematical modelling of a system's natural frequencies, mode shapes and damping characteristics [19]. For a linear time-invariant system, its behaviour can be described as the sum of its natural modes, where every mode exhibits simple harmonic motion (Figure 3.5).

This method can be applied to structural engineering, mechanical systems, and seismic systems and has also been extended beyond traditional fields of engineering to acoustics and musical instruments.

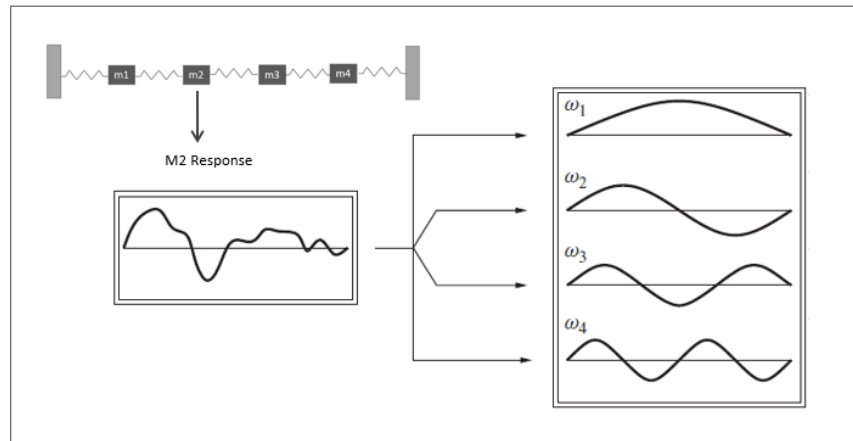


Figure 3.5: Response of a mass-spring system as a series of normal modes [7]

In terms of sound synthesis, modal analysis was first implemented by Adrien (1988) and developed into the MODALYS [20] system with notable research into a generation of 2-D modal models (surface networks) for modal synthesis [21].

Modal analysis was the mathematical technique used for this project, and the rationale for its use are:

- Opportunity to create physically inspired sounds.
- Discrete element approximation decomposes the problem and reduces complexity.
- Constituent modes can be manipulated.
- Natural and intuitive interaction with the physical control, hence the sound can be contextualised based on the expression of inputs and deformations.

The disadvantages were also considered:

- Changes to the system structure require a complete re-computation.
- Degraded performance as the sampling rate and the number of degrees of freedom increases, resulting in a delayed response. Parallel processing methods have been applied to MSS computations [22], and is an option to consider as the processing of each mode could run concurrently.
- Numerical accuracy is dependent on the sampling rate.

4 SOFTWARE IMPLEMENTATION

MATLAB was chosen as the implementation toolset for this project for several reasons:

1. The author has existing expertise with the software.
2. MATLAB strongly aligns with the matrix algebra that modal analysis requires.
3. Provides a rich library of functions and features for sound and digital signal processing and visualizations that is a requirement for the end-state interactive application.
4. The MATLAB introduction of the “App Designer” allows for rapid User Interface development.
5. Strong active community of users for help and advice.
6. Help and documentation is mature with comprehensive examples and tutorials.

The intention was to build the functions and modules that are required for this project from first principles, and therefore there is no strong dependency on specialist libraries for modal analysis geared towards mass-spring systems that feature in others tool suites such as MODALYS for MAX/MSP, Next Generation Sound Synthesis (NESS) [23] or Synthesis ToolKit (STK) for C++ . As MATLAB is natively an interpreted language, the execution of the system is expected to be slower than a compiled language such as C or C++. However, there is an option to use the “Application Compiler” to compile the application to a targeted operating system which is an option.

5 THEORETICAL BACKGROUND

5.1 MODAL ANALYSIS

For the purpose of this project, we considered a vibratory string modelled as a linear chain of coupled mass spring oscillators with mass, stiffness and damping properties and we will explore nonphysical properties of their production of displacement as an audio response (Figure 5.1). This contrasts with a uniform vibratory string with an unvarying distribution of its mass and elastic properties. It is noted that a uniform string could be modelled by this proposed method [6].

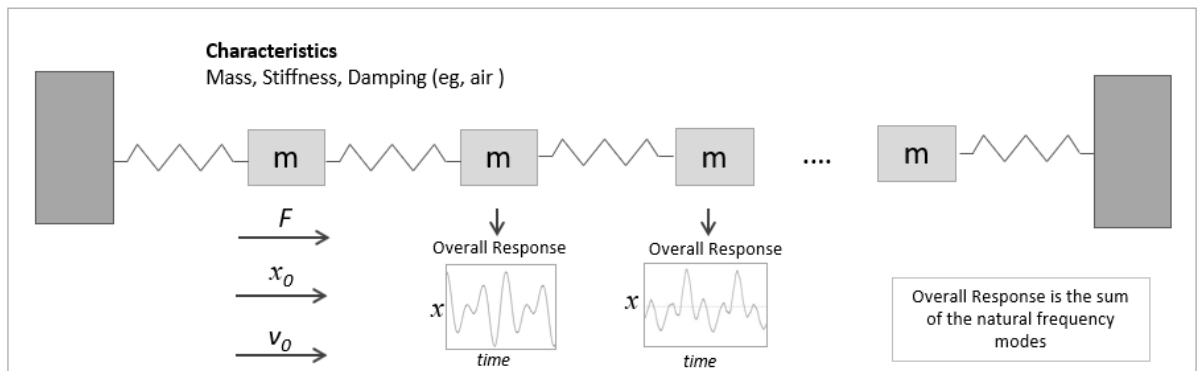


Figure 5.1: N-DOF Coupled Mass Spring Oscillators

A mass-spring system with N masses can be described as having N modes where each mode has a natural frequency, damping ratio and shape (characteristic pattern of displacement). The system is said to have N degrees-of-freedom (N-DOF).

Thereby, each natural mode contributes to the overall vibration response by virtue of the mode shape and nature of the excitation. Essentially, this is like a complex waveform that is comprised of a series of sine waves.

Through Modal Analysis, the properties of a system can be decomposed into sets of matrices to describe the equations of motions, involving matrix inverses, symmetry and eigenvalue problems [19].

5.1.1 Theory of a 2-DOF System Under Free Vibration

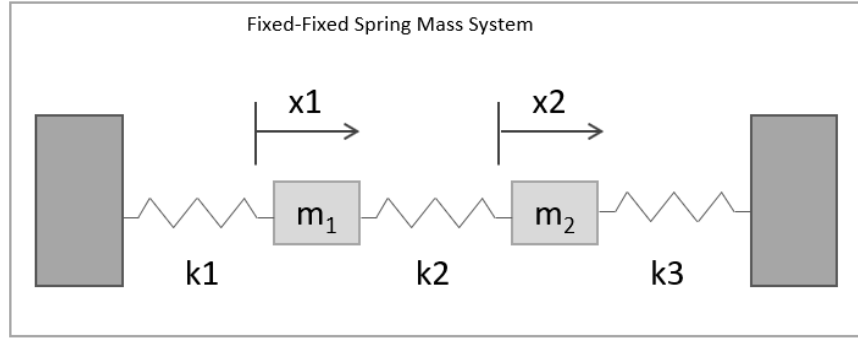


Figure 5.2: Fixed-Fixed 2-DOF Spring Mass System

A 2-DOF system is the simplest form of an N-DOF System (Figure 5.2). The masses are connected by springs with stiffness k and boundary conditions such that the end springs are connected to fixed points. Free vibration is initiated by an initial displacement. Applying Hooke's Law and Newton's second law of motion for the masses m_n displaced by x_n , results in the following simultaneous equations [24]:

$$m_1 \ddot{x}_1 + (k_1 + k_2)x_1 - k_2x_2 = 0 \quad (5.1)$$

$$m_2 \ddot{x}_2 + (k_2 + k_3)x_2 - k_2x_1 = 0 \quad (5.2)$$

These equations can be decoupled in matrix form as follows:

$$\begin{bmatrix} m_1 & 0 \\ 0 & m_2 \end{bmatrix} \ddot{x} + \begin{bmatrix} k_1 + k_2 & -k_2 \\ -k_2 & k_2 + k_3 \end{bmatrix} x = 0 \quad (5.3)$$

$$\text{where, } \ddot{x} = \begin{bmatrix} \ddot{x}_1 \\ \ddot{x}_2 \end{bmatrix} \text{ and } x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$[M]\ddot{x} + [K]x = \{0\} \quad (5.4)$$

$$[M]\ddot{x} + [K]x = F \text{ (zero for free vibration)} \quad (5.5)$$

Equation 5.4 can be reduced by eigenvalue decomposition with the following summarised modal analysis procedure:

$$([K] - \lambda[M])\{\psi\} = \{0\} \quad (5.6)$$

$$([K] - \omega^2[M])\{\psi\} = \{0\} \quad (5.7)$$

Assuming Equation 5.6 consists of n eigenvalues λ , where $\lambda = \omega_i^2$, ω is the natural frequency and $\{\psi\}_i$ eigenvectors where $i = 1, 2, \dots, n$. The Mass $[M]$ and stiffness $[K]$ matrices are symmetrical.

Mode shapes are then normalised to give the mass-normalised mode shape $[\Phi]$, to ensure a unique solution to the Modal Analysis:

$$[\phi] = [\psi][m_i]^{-1/2} \quad (5.8)$$

When the eigenvectors are normalised with respect to the mass matrix, such that

$$\{\psi\}_i^T [M] \{\psi\}_i = 1 \text{ and } \{\psi\}_i^T [K] \{\psi\}_i = \omega_i^2 = \lambda \quad (5.9)$$

The solution of the motion in decoupled modal coordinates is:

$$x(t) = \sum_{i=0}^{n=2} \{\psi\}_i \left(\{\psi\}_i^T [M] x(0) \cos \omega_i t + \frac{1}{\omega_i} \{\psi\}_i^T [M] \dot{x}(0) \sin \omega_i t \right) \quad (5.10)$$

Where $x(0)$ and $\dot{x}(0)$ are the initial conditions displacement and velocities vectors.

5.1.2 Multiple Degrees of Freedom and Boundary Conditions

For the N-DOF System with N masses, the matrices for Equation 5.5 are [25]:

$$[M] = \begin{bmatrix} m_1 & 0 & 0 & 0 & \dots & 0 \\ 0 & m_2 & 0 & 0 & \dots & 0 \\ 0 & 0 & m_3 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & \dots & 0 \\ \dots & \dots & \dots & \dots & m_{n-1} & 0 \\ 0 & 0 & 0 & 0 & 0 & m_n \end{bmatrix} \quad (5.11)$$

$$[K] = \begin{bmatrix} k_1 + k_2 & -k_2 & 0 & \dots & 0 & 0 \\ -k_2 & k_2 + k_3 & -k_3 & \dots & 0 & 0 \\ 0 & -k_3 & k_3 + k_4 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & 0 \\ 0 & 0 & \dots & \dots & k_{n-1} + k_n & -k_n \\ 0 & 0 & 0 & \dots & -k_n & k_n + k_{n+1} \end{bmatrix} \quad (5.12)$$

So far, we have considered the edge masses fixed at both ends (by springs). For a fixed-free or free-free system, the uncoupled equation of motions can be easily adjusted by considering the absence of spring(s). For a Fixed-Free N-DOF System $k_{n+1} = 0$, and for Free-Free N-DOF System $k_1 = k_{n+1} = 0$.

5.1.3 Theory of Damped System under free vibration

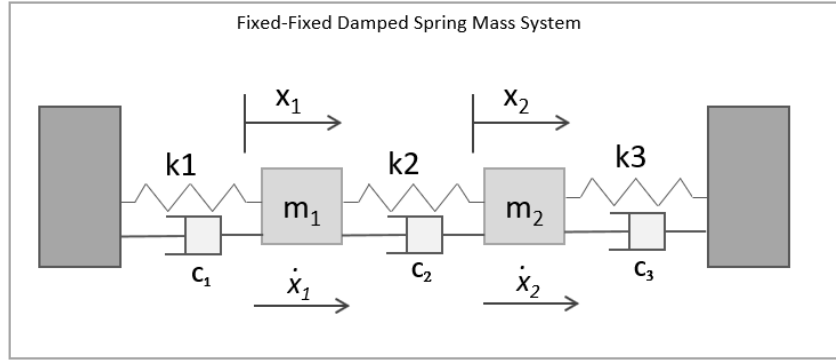


Figure 5.3: Fixed-Fixed Damped Spring Mass System

For the 2-DOF system in Figure 5.3, with damping, the equations of motion now become the following two coupled second-order differential equations.

$$m_1 \ddot{x}_1 + (c_1 + c_2)\dot{x}_1 - c_2\dot{x}_2 + (k_1 + k_2)x_1 - k_2x_2 = 0 \quad (5.13)$$

$$m_2 \ddot{x}_2 + (c_2 + c_3)\dot{x}_2 - c_2\dot{x}_1 + (k_2 + k_3)x_2 - k_2x_1 = 0 \quad (5.14)$$

Once damping is introduced to the free-vibration system it becomes difficult to uncouple the equations of motion. However, when assuming viscous damping, the mode shapes are identical. When viscous damping is added to Equation 5.4, the equation of motion becomes (5.15); where $[C]$ denotes the damping matrix and the damping force is proportional to the mass velocity.

$$[M]\ddot{x} + [C]\dot{x} + [K]x = \{0\} \quad (5.15)$$

If we assume viscous damping matrix $[C]$ is proportional to mass and stiffness matrices, then it can be written as:

$$[C] = \alpha [M] + \beta [K] \quad (5.16)$$

Thereby,

$$[M]\ddot{x} + [\zeta]\dot{x} + [K]x = \{0\} \quad (5.17)$$

Where ζ is the modal damping matrix applied independently to each mode.

Equation 5.17 consists of N uncoupled equations and the damped natural frequency of a mode is.

$$\omega_{di} = \omega_i \sqrt{1 - \zeta^2} \quad (5.18)$$

In summary, the equation of motion in uncoupled modal coordinates for a proportionally damped system is:

$$x(t) = \sum_{i=0}^n e^{-\zeta \omega_i t} \{\psi\}_i \left(\frac{\{\psi\}_i^T [M] x(0) \cos \omega_i t}{\sqrt{1 - \zeta^2}} + \frac{\{\psi\}_i^T [M] \dot{x}(0) \sin \omega_i t}{\omega_i \sqrt{1 - \zeta^2}} \right) \quad (5.19)$$

When no damping is applied equation 5.19 reverts to equation 5.10.

1.2 NORMAL MODE SHAPES

From the modal-analysis, it follows that an N-DOF mass-system has N normal modes, each of which is a unique modal shape and the overall response of the system will be a sum of its modes, and depending on the initial conditions, will have varying phase and amplitudes for each mode (Figure 5.4).

When the initial amplitude conditions for each mass are a factor of a normal mode, the output will be a single sinusoid, where the frequency response of all the masses is equal to the natural frequency for that eigenmode [26].

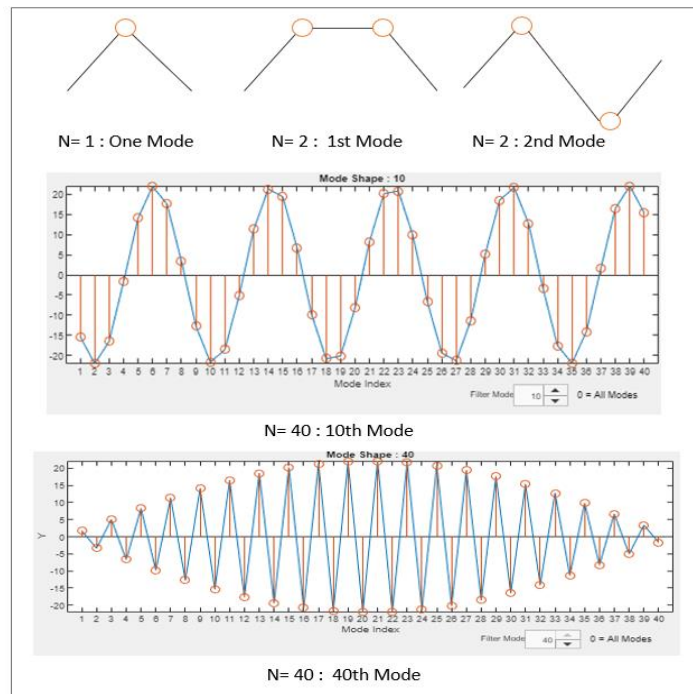


Figure 5.4: Normal Modes Shapes

6 METHODOLOGY

The main objective of this project was to examine the behaviour of an unphysical system. In order to achieve this, several aims set out:

6.1 AIMS

- Research materials to satisfy the core foundations of the project's physical model.
- Demonstrate knowledge and understanding of theory through MATLAB implementation of physical model components.
- Develop working model/s to implement this theory correctly.
- Iterate upon models and gain further control over more system parameters.
- Discuss possible effects of these parameters, with the goal of finding unphysical behaviour.
- Overall try and adopt an agile and iterative way of running the project.

These aims were broken down in the following flow and sequences (Figure 6.1):

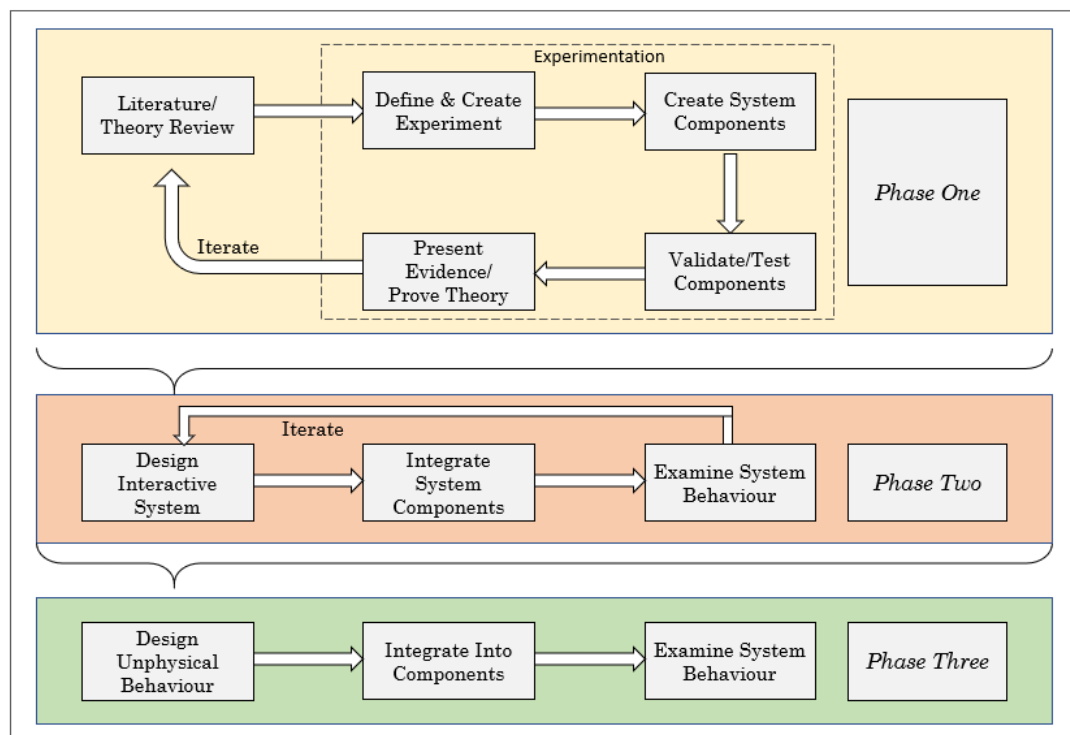


Figure 6.1: Methodology Cycle

6.2 PHASE ONE: INITIAL EXPERIMENTS

6.2.1 Foundations - Literature/ Theory Review

Relevant material and background theory were identified (*Literature Review and Theoretical Background*) for methods in which to implement the features, including formulae and matrix structures for a physically modelled mass-spring system.

Once confident with foundation theory, the first phase of the project began with the simplest systems, specifically a 2-DOF, adding more degrees-of-freedom and increasing complexity up to a proportional damped N-DOF system through an iterative cycle of experimentation and validation. Finally adding characterisation parameters that would be considered unphysical of an oscillating mass-spring system using Modal Analysis.

6.2.2 Experimentation

As shown in Figure 6.1, Methodology Cycle, the experimentation stage consists of several activities. The aim was to realise the modal theory, leveraging MATLAB library functions to determine the eigenvalues and eigenvectors and create bespoke functions and test cases scripts to validate these functions. This methodical approach is common in software systems and can be classified as Test Driven Development (TDD) [27].

The process (including results) was managed in Atlassian JIRA. The intent was also to ensure there was a demonstrated deliverable at the end of each iteration cycle, a common practice in Agile methods.

6.2.3 Define & Create Experiment

The overall aim for an experimental iteration was captured in an Experimentation Set Template (Table 1).

Experiment Set	Description
Aim	The aim of the experimentation iteration with expected outcomes.
JIRA Task	Manage and link the methods in JIRA.
MATLAB Changes	Description of the modular functions and test case scripts developed.
Overall Summary	Brief description of the results performed against objectives.

Table 1: Experimentation Set Template

6.2.4 Create System Components

This activity involved the development of the MATLAB functions to implement the mathematical components of the system and modify existing functions where necessary. The early experiments focussed on the core modal function for the eigenvalue problem such as the `pm_mdof_free_vibration.m` (Appendix B.1), `pm_mdof_free_damped_vibration.m` (Appendix B.4) (once damping was introduced) and supporting functions for uniform and novel initial conditions.

6.2.5 Validate/ Test Components

For all the modular component functions, supporting test scripts were created to verify their behaviour was correct. These were captured as individual experiments and formed part of the overall experiment set. A secondary Experimentation Template (Table 2) was created for this activity.

Item	Description
Experiment ID	Self-explanatory.
Test	Description of the test to be performed.
Input parameters	The control parameters for the experiment/test.
Other Parameters	Other parameters to support the experiment/theory.
MATLAB Changes	Details of new MATLAB functions or updates to existing functions to support experiments and create modular re-usable code for the main application. Details of the test case MATLAB function acting as unit tests (requiring manual execution).
Results	Summary of the results, including evidenced graphical and runtime outputs.
Conclusion	Description of the behaviour and to validate the outcomes against expectations.

Table 2: Experimentation Template

6.2.6 Present Evidence/ Prove Theory

The results of each experiment were written up in the test template (Table 2) and an overall summary was fed back up the experiment set. The evidence also included relevant graphical plots or tabular runtime output where appropriate.

6.3 PHASE TWO: INTERACTIVE APP

For this project, the optimal way to test and interact with the unphysical modelling behaviour was through an interactive application with a variety of unphysical characteristic inputs and the observed visual and audio outputs. Primarily it allowed for a more efficient investigation into the system's behaviour under test conditions through clarity of inputs and faster collection of results.

6.3.1 Design Interactive System

Phase 1 of the Project successfully captured and verified the approach and theory with a set of modular Physical Model Functions from experimentation in which to integrate into the design and development an interactive system and extend with additional features and realise the algorithm (Table 3).

Algorithm
Interactive initial conditions and boundary conditions
Determine the matrices for mass and stiffness
Establish the modal shapes using modal analysis (eigenvalues/vectors)
Calculate the mode responses over the time sample based on initial conditions.
Perform superposition of modes for the overall response for each degree of freedom (mass).
Apply pickup method and create sound synthesis and analyse signal response.

Table 3: Physical Model Algorithm

Figure 5.2 shows the first iteration design (with default values) of Interactive GUI application constructed using the MATLAB App Designer Toolset (Design View).

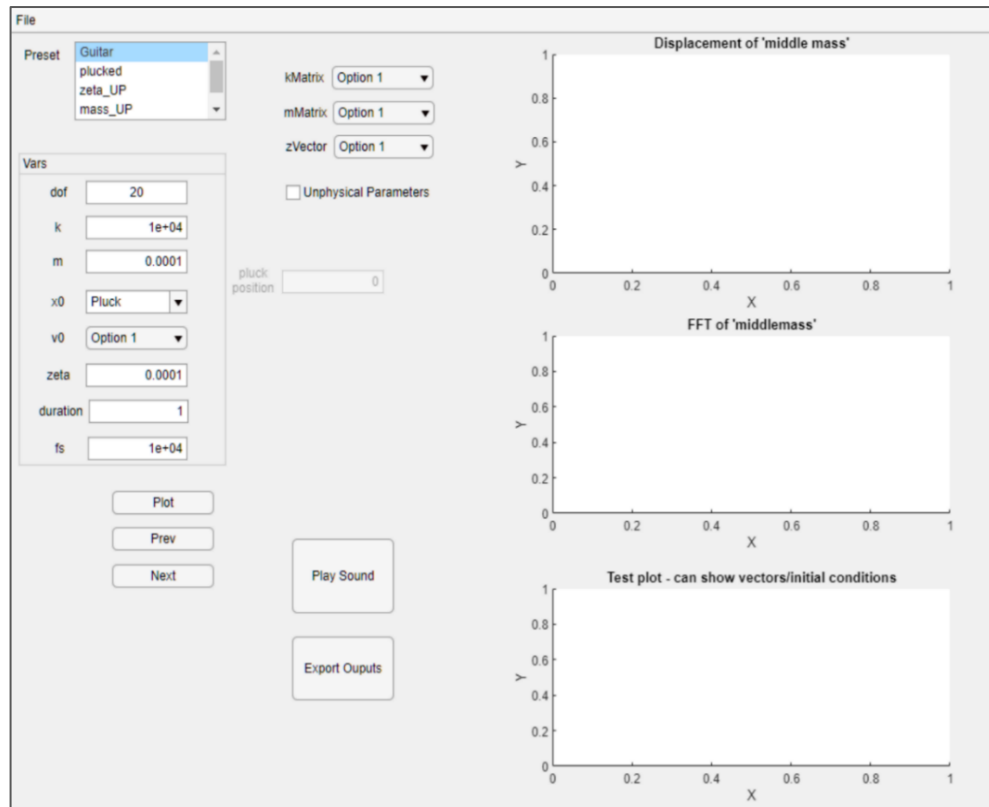


Figure 6.2: First iteration of Interactive app

6.3.2 Integrate System Components

The main user interface application code underlying the MATLAB Design View is the project “.mlapp” code file (pm_interactive.mlapp). The project’s modular functions from Phase 1 were integrated as linkage to the user interface components through callback functions and plot updates (Axes Components).

Iteration 1 Features:

- Surfaced all control parameters from the Physical Model Functions developed during Phase 1 experimentation to the user interface with associated plots.
- Integrated the MATLAB functions: % From the general form :

```
% ( [M] * v0 ) + ( [K] * v0 ) = 0 ( free vibration - no damping )
% derive the stiffness KMatrix for a linear M-Dof system FIXED at both ends
% FIX:k1-m1-k2-m2-k3-m3-k4:FIX
% Input is an array (row ) of stiffness K's
% the array size has  dof + 1 elements
% Example 3 -dof
% [  k1+k2    -k2      0
%    -k2     k2+k3    -k3
%      0     -k3     k3 +4 ]
```

```

% pm_mdof_create_kmatrix_linear( [1,1,1,1] );
%Returns
% [ 2    -1    0
%   -1    2   -1
%    0   -1    2 ]

function [KMatrix] = pm_mdof_create_kmatrix_linear( kVector)

    dof = length( kVector) -1 ;
    % create a dof length vector of values k1+k2, k2+k3, k3+k4
    addedKs= zeros(dof,1);
    for ( i = 1:dof )
        addedKs(i) = kVector (i) + kVector(i+1);
    end
    % make a diagonal out of the k1+k2, k2+k3, k3+k4
    KMatrix = diag(addedKs);
    % make vector of negative k's starting at position 2 ; -k2, -k3,-k4 etc
    negativeKs = zeros(dof-1,1);
    for ( i = 2:dof)
        negativeKs(i-1) = -kVector (i);
    end
    % now apply these diagonally either size of the k matrix
    for ( i = 1:dof -1 )
        KMatrix( i, i+1 ) = negativeKs(i);
        KMatrix( i+1, i ) = negativeKs(i);
    end
    if ( issymmetric( KMatrix ) == 0 )
        error('**** Error. \nCalculated [K] Matrix is incorrect' )
    end
end

```

- m_mdof_free_damped_vibration.m,
-
- pm_mdof_create_kmatrix_linear.m, pm_mdof_create_zeta_vector.m,
pm_mdof_triangle_pluck_condition.m (Appendix B).
- Added placeholder options for unphysical parameters to set up the initial conditions to the control variants; masses, springs, displacements, velocities and damping.
- Added placeholders to scroll back and forth through the runtime executions.
- For the middle mass, playback of audio, plot FFT, amplitude response and initial displacement conditions.
- Option to export as MATLAB Figure (.fig) the plots for the displacement, FFT and save audio as a .WAV file

6.3.3 Examine System Behaviour

In order to validate the system, the first Phase 2 interactive task was to establish baseline characteristics for a simple plucked string (without a resonant chamber or other forms of impedance).

This ensured the response of the system under normal conditions matched theoretical results. This facilitated later when attempting to differentiate between changes in output due to its inherent behaviour and deliberate changes due to unphysical behaviour.

The Interactive application design and development underwent several iterations in Phase Two, improving its usability, adding non-physical profile vectors, software improvements, custom pickups, import/export and reporting features.

Figure 6.3 shows a feature of the final application; a generated textual report, FFT (Fast Fourier Transforms) and Spectrogram Plots (.png), Audio output file (.wav) when test run was exported.

A useful addition was the introduction of the PMUseCase, a MATLAB class that holds all the details of the input control parameters for a given test (known as a Plot on the interactive interface). This was saved as data file `UseCase.mat` and could also be imported between stop and starts and shared in the final project deliverables.

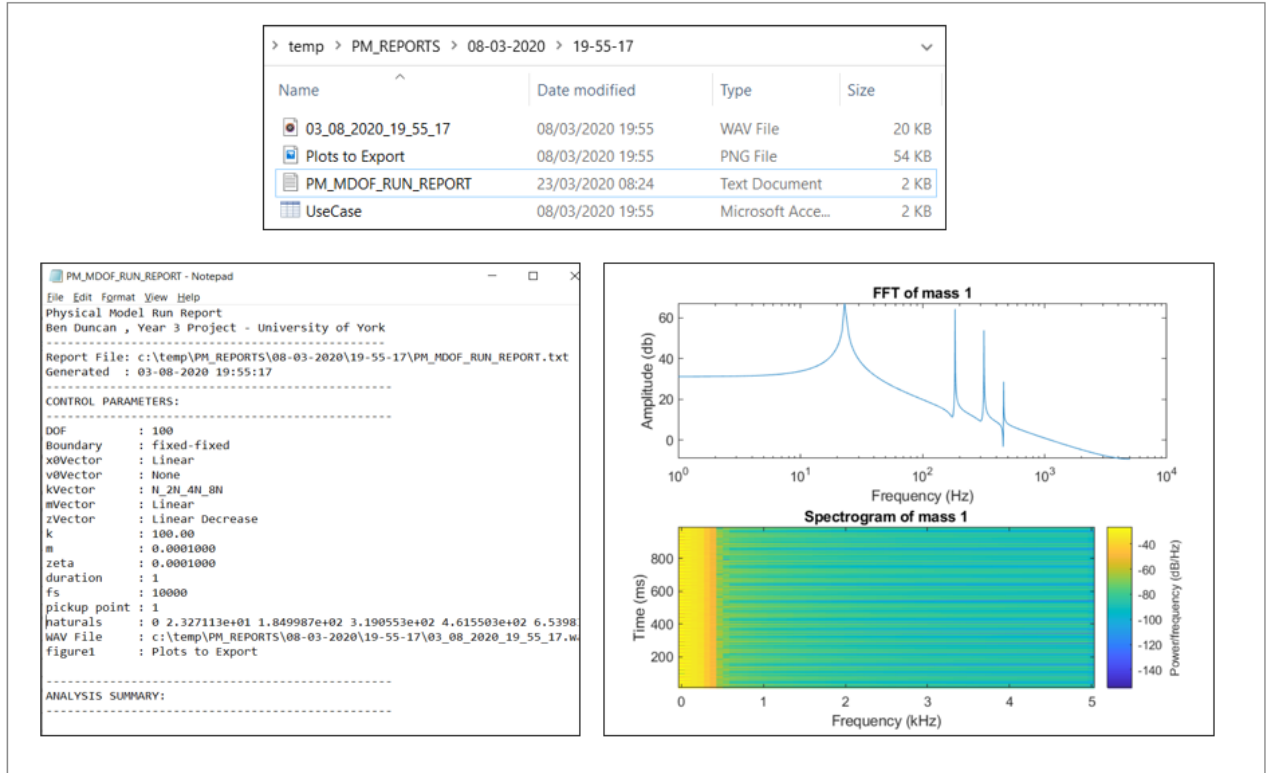


Figure 6.3: Exported Use Case Artefacts

6.4 PHASE THREE: UNPHYSICAL MODEL

Phase Three builds upon the application from Phase Two specifically to include additional functionality to explore unphysical behaviour of the system.

6.4.1 Design Unphysical Behaviour

Extensions to existing functions were made to add novel behaviour. Experimentation to research and examine the effects of this behaviour were self-documenting by the addition of the Use Case functionality, which allowed for on-screen written feedback and the option to mark the significance of a run. Table 4, shows the final set of interactive control parameters

Control Parameter	Description
DOF	Number of masses (degrees of freedom)
K	Spring constant
M	Mass
X	Displacement initial conditions for mass(es)
V	Velocity initial conditions for mass(s)
Z	Modal damping coefficient applied to the normal mode(s)
K-Vector	A set of 'profiles' that can be applied to each spring using the k-value.
M-Vector	A set of 'profiles' that can be applied to the masses using the m-value.
X-Vector	A set of displacement 'profiles' that can be applied to mass using the x value.
Z-Vector	A set of modal damping 'profiles' that can be applied to normal-modes using the z-value.
Apex position	Used in conjunction with triangular-shaped profiles
Pickup Method	A set of profiles for the time-varying pickup.
Sweeps	Used in conjunction with a time-varying pickup, to specify the number of sweeps across the mass-amplitudes response matrix.
Stereo	Applies the Left and Right pickups
T	Duration
Fs	Sample rate
Boundary conditions	Fixed-fixed, fixed-free or free-free spring boundary connections.
Pickup point(s)	Mass(es) where an output is measured.

Table 4: Model Control Parameters

Of note is the concept of Vector Profiles, which are the names of profile shapes to be applied to masses, springs, modal damping and the displacements and velocities; one example is the Bell Shape. To aid the interface feedback, the profile shape is displayed visually on the user interface (Figure 6.4). Appendix D contains a table of all vector profiles.

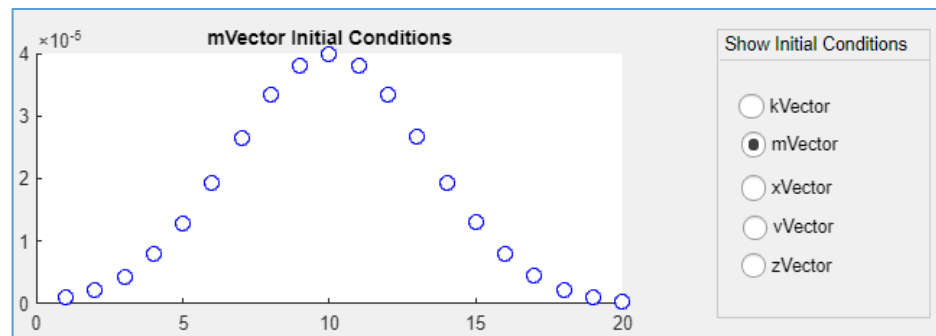


Figure 6.4: Bell Shape distribution applied to mass sizes (20 -DOF)

6.4.2 Integrate Components

The main changes were the development of the vector profiles functions and time-varying pickups, resulting in the `pm_create_initial_condition_vector` (for the Vector Profiles) and `pm_mdof_time_variant_pickup` functions and their integration into the interactive application.

6.4.3 Examine System Behaviour

There were several ways to evaluate the system, of which the sound produced was the most important to consider. The sounds were compared against real sound recordings of instruments, especially in relation to their timbre. Also, it was useful to compare to synthetic sounds, such as those created by other physical modelling systems. In both cases, this helped to evaluate whether the sounds created by the system were observably different from existing sounds.

Another aspect of the system that was evaluated was the validity of its implementation. It would have been easier to consider any interesting sounds produced as valid, but it was important that the behaviour was fully understood. This would mean that the system would be fully reproducible, and with enough knowledge of the behaviour, it was easier to use and generate sounds.

Finally, for each test of the system, it was important to consider the significance of its results. This also helped to map the overall domain of behaviours that were relevant and interesting, and discard those which were not significant including the combinations of stiffness and mass sizes resulting in a frequency response below the range of human hearing. Table 5 shows some of the key functions integrated and an example of their usage by the GUI based on user input.

Function/ Description	Example Unphysical Behaviour
pm_mdof_create_kmatrix_linear.m	
Creates a K matrix based on an input K-vector, which consists of the values of each spring stiffness.	Each spring has an alternating stiffness: $k(1) = 1 \times 10^6 Nm^{-1}$ $k(2) = 2 \times 10^6 Nm^{-1}$ $k(3) = 1 \times 10^6 Nm^{-1}$
pm_mdof_create_Mmatrix_linear.m	
Creates an M matrix based on an input mass-vector, which consists of the values of each mass.	Each mass has an increasing value: $m(1) = 1 \times 10^{-5} kg$ $m(2) = 2 \times 10^{-5} kg$ $m(n) = n \times 10^{-5} kg$
pm_mdof_create_zeta_vector.m	
Creates a damping vector. This will be used in the main calculation to add damping to each mode.	Higher modes have less damping: $\zeta(1) = 0.99$ $\zeta(n-1) = 0.1$ $\zeta(n) = 0.01$
<p>% From the general form :</p> <pre> % ([M] * v0) + ([K] * v0) = 0 (free vibration - no damping) % derive the stiffness KMatrix for a linear M-Dof system FIXED at both ends % FIX:k1-m1-k2-m2-k3-m3-k4:FIX % Input is an array (row) of stiffness K's % the array size has dof + 1 elements % Example 3 -dof % [k1+k2 -k2 0 % -k2 k2+k3 -k3 % 0 -k3 k3 +4] % pm_mdof_create_kmatrix_linear([1,1,1,1]); %Returns % [2 -1 0 % -1 2 -1 % 0 -1 2] function [KMatrix] = pm_mdof_create_kmatrix_linear(kVector) dof = length(kVector) -1 ; % create a dof length vector of values k1+k2, k2+k3, k3+k4 addedKs= zeros(dof,1); for (i = 1:dof) addedKs(i) = kVector (i) + kVector(i+1); end % make a diagonal out of the k1+k2, k2+k3, k3+k4 KMatrix = diag(addedKs); % make vector of negative k's starting at position 2 ; -k2, -k3,-k4 etc negativeKs = zeros(dof-1,1); for (i = 2:dof) negativeKs(i-1) = -kVector (i); end % now apply these diagonally either size of the k matrix for (i = 1:dof -1) KMatrix(i, i+1) = negativeKs(i); KMatrix(i+1, i) = negativeKs(i); </pre>	

<pre> end if (issymmetric(KMatrix) == 0) error('**** Error. \nCalculated [K] Matrix is incorrect') end end </pre> <p style="text-align: center;">m_mdof_free_damped_vibration.m</p>	
Calculates the motion of the system using modal analysis, based upon its initial conditions, mass and spring stiffness values and any damping.	Quantisation in mode calculations ¹ Signal distortion e.g. clipping Omission of certain mode components
pm_create_initial_condition_vector.m	
Creates Profile Vectors to be applied to masses, springs, damping, displacements, velocities (Appendix D).	<i>Xth-Eigen</i> for Displacement An eigenvector of the system is used as the initial displacement vector. <i>Example</i> “Bell” applied to masses and ‘Triangle’ applied to displacement
pm_mdof_time_variant_pickup.m	
The moving pickup traverses the amplitudes response table (DOF matrix) in sweep profiles or driven by an amplitude response feedback	<i>Example</i> , “Moving Linear”, 8 Sweeps

Table 5: Key Functions implemented in interactive MATLAB Application

6.5 PLANNING

Several planning tools and project management packages were considered, but the final choice was Atlassian JIRA, which has widespread use across the agile software development community. Table 6 summaries the reasons for its adoption.

Advantages	Disadvantages
Flexibility. It proved to a general overall tool for stories, issues, documentation and defects.	Harder to implement as an individual due to the lack of a product owner.
Provides easy to use visual dashboard for reporting of the project and task progress.	Requires familiarity to a degree with agile methods and Kanban.

¹ While not directly related to how the physical model is set up, this tests the possible effects of quantisation such as “errors” in stages of calculation.

Able to track time spent on each task.	
Able to easily prioritise tasks related to the project.	
Cloud hosting makes it easily accessible with good support community and knowledge articles.	
Supports agile paradigms.	
Free for personal use.	

Table 6: JIRA Advantages and Disadvantages

A Project was setup in JIRA for the PM (Physical Model) and used for overall task management and tracking project features in the immediate pipeline such as creating Epics for larger goals, which were typically the experiment sets (Figure 6.5).

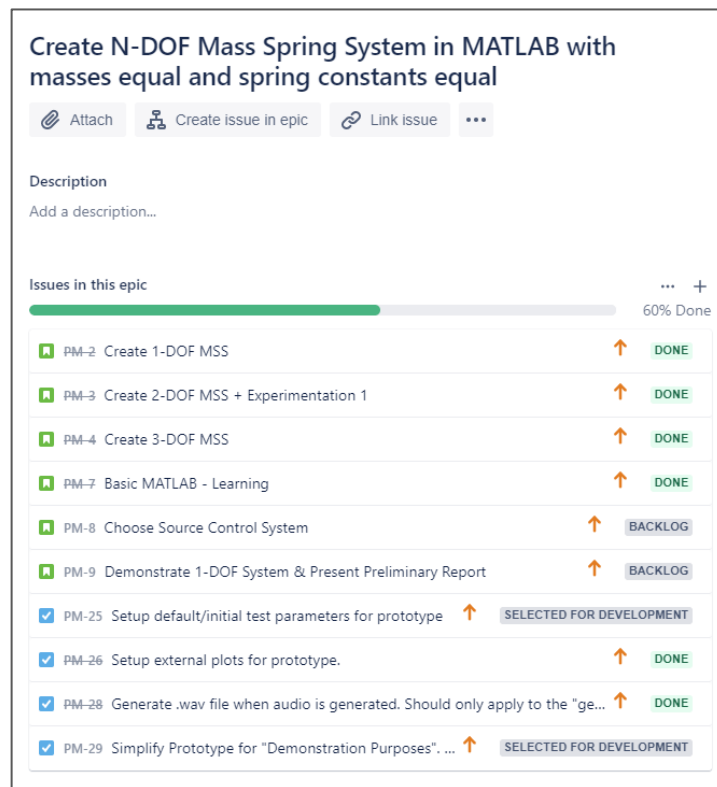


Figure 6.5: Example of a Project "Epic" and its sub-tasks

The Epics were broken down into smaller task or User Stories (Figure 6.6), together with the estimated effort they should take to complete, and then each task was prioritised to be completed in the next week's project cycle.
















Key		Summary	
	PM-2	DONE	Create 1-DOF MSS
	PM-3	DONE	Create 2-DOF MSS + Experimentation 1
	PM-4	DONE	Create 3-DOF MSS
	PM-7	DONE	Basic MATLAB - Learning
	PM-10	DONE	Setup Development and Tooling Environment
	PM-16	DONE	Add FFT plot for N DOF (1 mass only) and see if main freq is relevant to Normal Frequency
	PM-19	DONE	Weekly update/Retrospective - EMAIL UNI+PROJECT SUPERVISOR/S (FRIDAY)
	PM-28	DONE	Generate .wav file when audio is generated. Should only apply to the "generate figures" button
	PM-30	DONE	Test Viability of 3-D Plots e.g. FFT for each seperate mass.
	PM-37	DONE	PM Experiment 2 - 3 DOFS
	PM-38	IN PROGRESS	Experiment 3 - N-DOF (with dispersion & Pluck)
	PM-42	IN PROGRESS	Methodology
	PM-43	SELECTED FOR DEVEL...	Add references to Word document
	PM-47	DONE	Review Report Writing Techniques/Strategies
	PM-5	IN PROGRESS	Create a 1st Draft Template of the Report Structure
	PM-15	BACKLOG	Normal Modes (Eigen Modes)
	PM-26	DONE	Setup external plots for prototype.
	PM-6	BACKLOG	Research and References - Includes any relevant source material

Figure 6.6: Excerpt of Jira User Stories

Progress (Figure 6.7) was evaluated on a regular basis, adding new tasks, closing completed, updating task progress and updating or removing existing tasks and evaluating time allocation. Also, GitHub was used for tracking software changes, and as a backup for all project materials.

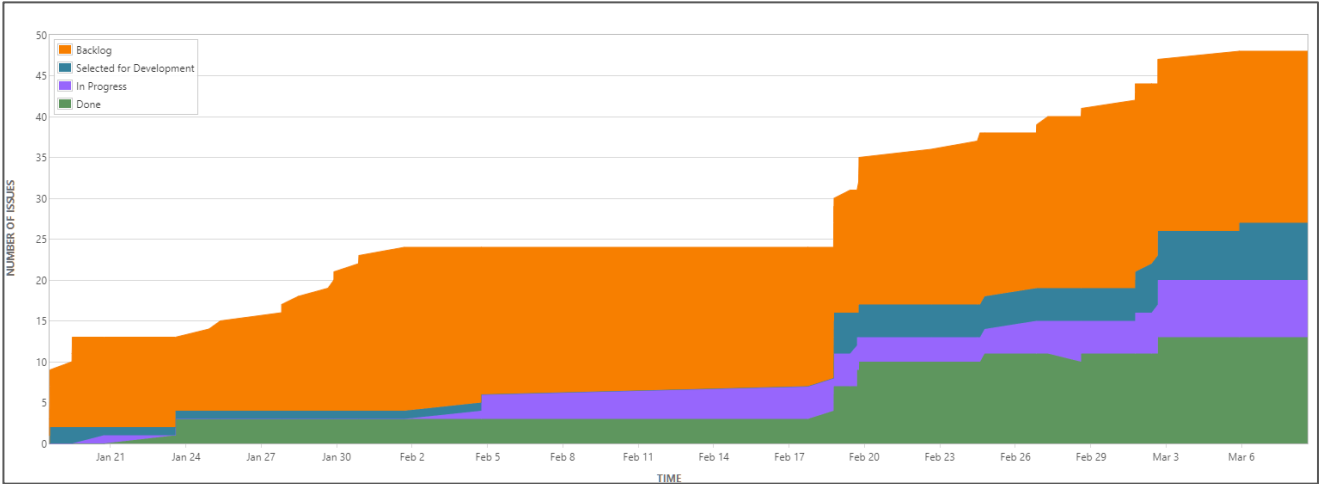


Figure 6.7: JIRA Overview of PM progress

7 PHASE ONE: INITIAL EXPERIMENTS

This is a breakdown of the Experimentation Phase as described in the Methodology, *Phase One: Initial Experiments*. See Appendix A for complete experiment testing.

7.1 OVERVIEW OF EXPERIMENTS

Experiment	Description
PM_EX01	Test the validity of a 2-DOF System with multiple test cases using MATLAB
Based on the Free Vibration Modal Equation (Modal 1-D Wave Equation) (5.10), a standalone MATLAB function was developed with test cases to validate the theory with demonstrated output, showing the amplitude responses for initial conditions of velocity and displacement for 2-DOF system.	
PM_EX02	Test the validity of a 3-DOF System's normal modes using MATLAB.
<p>A variant of the equation from Experiment 1 was developed to verify the superposition of modes for a 3-DOF system. This equation was not integrated into the final interactive application.</p> <p>Additional test cases were developed to:</p> <ul style="list-style-type: none"> • Validate the modal shapes. • Show the effects when a modal-shape was used as initial displacement input, resulting in the same frequency response for all three masses (with different amplitudes) • Demonstrate frequency response decreases (non-linearly) with mass size increase. • Demonstrate the frequency response increases (non-linearly) with spring stiffness increase. 	
PM_EX03	Extend the mass-spring model to a programmatic N-DOF System.
<p>Extended the system for multi degrees of freedom for N masses (N-DOF). Functions created to initialise the N-DOF mass and stiffness matrices with verified test case.</p> <p>Demonstrated a 20-DOF system with amplitude responses for the 1st, 10th, 20th masses using a non-stiff spring constant ($k=1$, $m=1$), showing slow energy propagation along the chain.</p> <p>Demonstrated the dispersive properties of the system for a 50-DOF system, as the modal frequencies converge to a maximum value.</p> <p>Created an initial condition function for a triangle-shaped pluck and verified the shape propagation along the mass chain and reflected for a 50-DOF system.</p> <p>Analysed the frequency response (using FFT) of a single mass in the 50-DOF system</p>	

PM_EX04	Test the validity of a damped System with multiple test cases using MATLAB.
<p>Created a MATLAB function based on equation Free Vibration Proportionally Damped Modal Equation (5.19). Demonstrated the effect of modal damping applied to each mode on a 2-DOF system.</p> <p>Created a MATLAB function to apply non-linear damping and demonstrated against a 20-DOF system for increasing damping and decreasing damping applied across the modes.</p>	

Table 7: Experimentation Summary

8 PHASE TWO: INTERACTIVE APP

As described in the Methodology, *Phase Two: Interactive App* the purpose of this phase was 1) The iterative design and development of the interactive application and 2) demonstrate the mass-spring system as a representation of vibrating string against a true-life example, before commencing into the unphysical behaviour Phase Three.

The *Methodology Cycle* (Figure 6.1) shows a clear demarcation between Phases 2 and 3, but there was some overlap as the non-physical functions that were developed in Phase 3 were then rolled back into this phase of application integration.

8.1 DESIGN INTERACTIVE SYSTEM AND INTEGRATE SYSTEM COMPONENTS

8.1.1 Component Model

Figure 8.1 shows a component breakdown of the Interactive System. The Inputs detail the User action activities with the system, with specific categories for the Control Parameters that affect the response of which a subset is specific to the core modal calculations. The Outputs are both visual in terms of plots such as FFT, Spectrogram and Initial Conditions and physical such as the audio playback and file system artefacts such as reports, images, WAV files and data files. Although shown in the component design, the Modal Shape Plots did not make it into the final system as the User Interface became crowded. However, these were demonstrated in Phase One. The main application itself manifests as the 'PM Application' that integrates and interacts with all the system components.

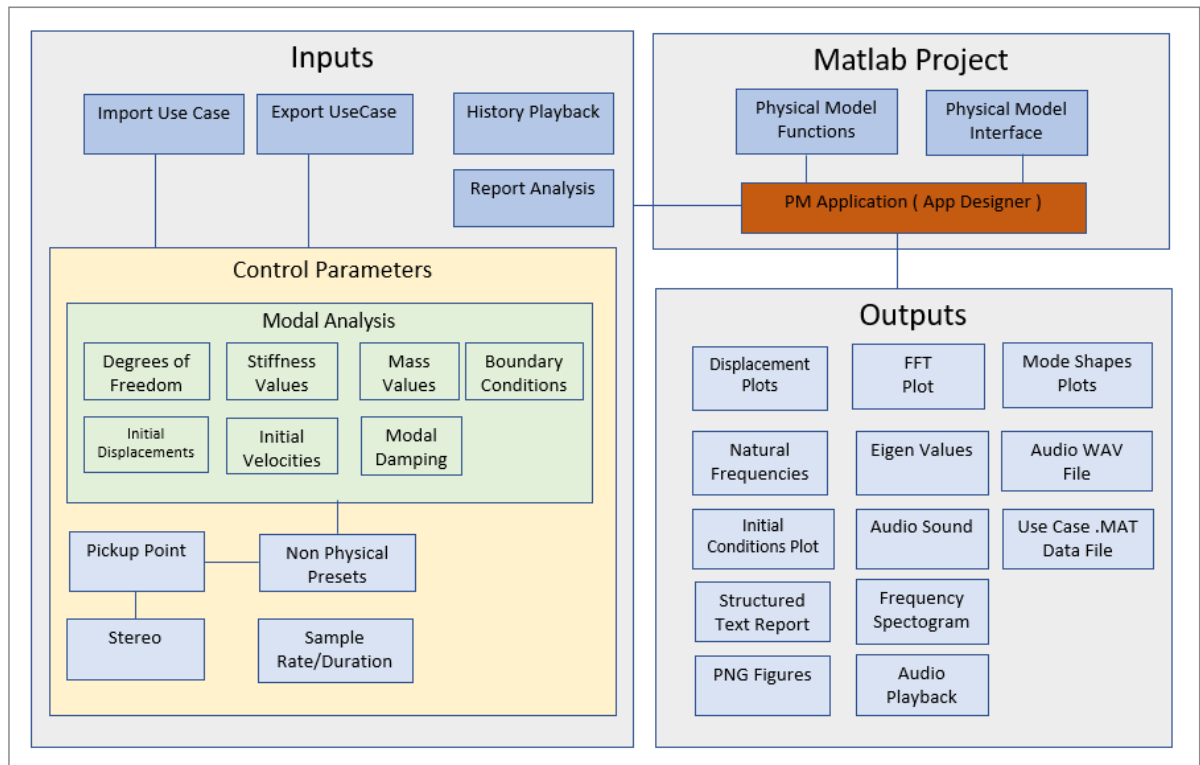


Figure 8.1: Project Functional Components

8.1.2 Intermediate Application

Figure 8.2 shows the Iteration 2 of the application. The features that were enhanced since Iteration 1 (*Integrate System Components*) are:

- Changes to the user interface components and positioning.
- Extended the controls inputs to align with Vector Profiles for the initial conditions distributed to the control variants; masses, springs, displacements, velocities and damping and integration with `pm_create_initial_conditions.m` (developed in Phase 3).
- Introduced a `PMUseCase`, a serialised structured class to capture the applied control parameters and calculated responses.
- Implemented the history playback functionality for `PMUseCase`. This enabled scrolling through the previous and next `PMUseCase` and aided usability by monitoring incremental changes to the controls.
- Playback of audio at the designated pickup point.

- Added functionality for the user to input the significance of an executed Use Case, provide written analysis and to export the Use Case as Structured Text Report, Use Case MAT data file, PNG files for plots and audio WAV file for the pickup point to the local file system. It was decided to partition the PMUseCase structure such that the matrices and amplitudes responses were not saved. This significantly reduced the file size from MBs to KBs and aided their portability.
- Option to import a saved Use Case from the file system and calculate the response. This underlying method also served as the basis for loading Use Case Presets when the system was initiated.

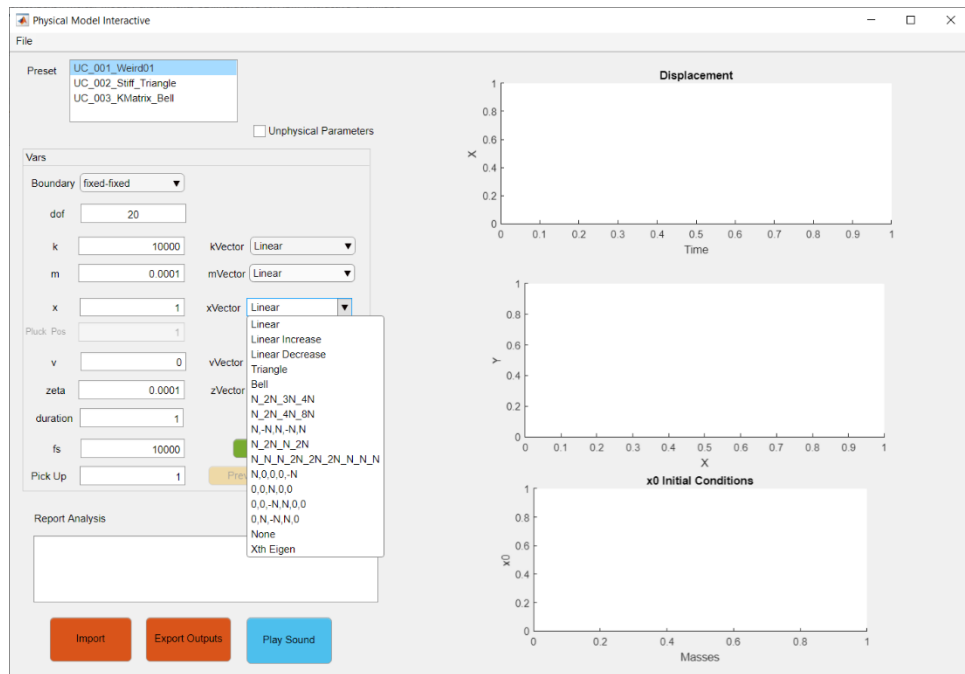


Figure 8.2: PM Application (MATLAB App Designer) version 0.2

8.1.3 Final Application

Figure 8.3 shows the final version of the interactive application. At this stage, all the non-physical functional features had been completed in Phase 3. The main features that were introduced since Iteration 2 (*Intermediate Application*) are:

- Final layout to the user interface components and overall geometry, which significantly improved the interactive usability.

- Added exception handling and error checking to code and functions and relayed meaningful error messages to the User.
- Integrated the time-variant pickup function from Phase 3 with their sweep profiles.
- Added slider components for the Pickup points with an option for Stereo, including secondary (Right) pickup channel.
- Normalised the audio output.
- Updated the Initial conditions plots to include the profile Masses, Displacements, Velocities and Damping, Stiffness.
- Use cases that are marked as significant are also saved to the Presets folder (updated with by Refresh button).
- Significant functionality for the conditional rendering of User interface components (disabled or enabled) depending on the options selected.
- Disabled the Plot button after a test has been performed and re-enabled when control parameters values have changed.

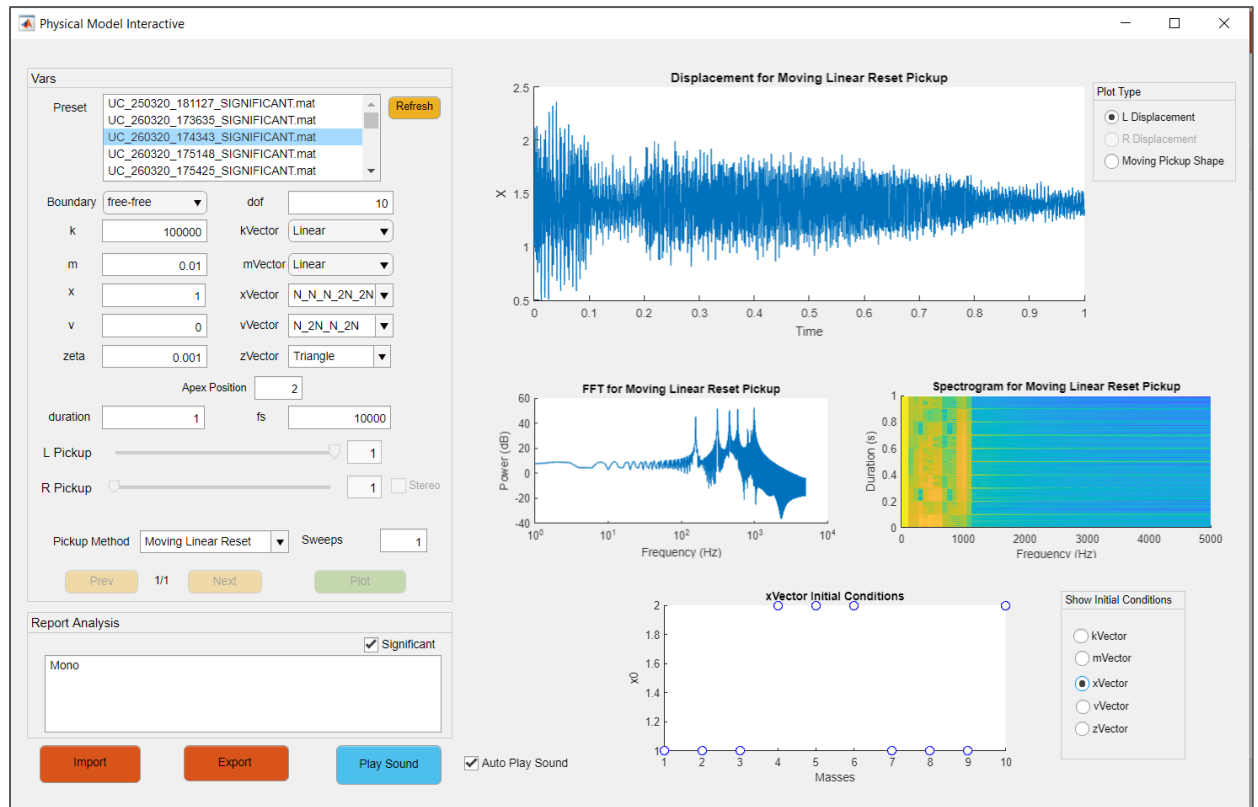


Figure 8.3: PM Application (MATLAB App Designer), Final version

8.1.4 Application Performance

Further analysis showed that the computational task for calculating the amplitude response in the function, % From the general form :

```
% ( [M] * v0 ) + ( [K] * v0 ) = 0 ( free vibration - no damping )
% derive the stiffness KMatrix for a linear M-Dof system FIXED at both ends
% FIX:k1-m1-k2-m2-k3-m3-k4:FIX
% Input is an array (row ) of stiffness K's
% the array size has dof + 1 elements
% Example 3 -dof
% [ k1+k2    -k2      0
%    -k2     k2+k3   -k3
%      0     -k3     k3+k4 ]
% pm_mdof_create_kmatrix_linear( [1,1,1,1] );
%Returns
% [ 2    -1     0
%   -1     2    -1
%    0    -1     2 ]
```

```
function [KMatrix] = pm_mdof_create_kmatrix_linear( kVector)
```

```
dof = length( kVector) -1 ;
% create a dof length vector of values k1+k2, k2+k3, k3+k4
addedKs= zeros(dof,1);
for ( i = 1:dof )
    addedKs(i) = kVector (i) + kVector(i+1);
```

```

end
% make a diagonal out of the k1+k2, k2+k3, k3+k4
KMatrix = diag(addedKs);
% make vector of negative k's starting at position 2 ; -k2, -k3,-k4 etc
negativeKs = zeros(dof-1,1);
for ( i = 2:dof)
    negativeKs(i-1) = -kVector (i);
end
% now apply these diagonally either size of the k matrix
for ( i = 1:dof -1 )
    KMatrix( i, i+1 ) = negativeKs(i);
    KMatrix( i+1, i ) = negativeKs(i);
end
if ( issymmetric( KMatrix ) == 0 )
    error('**** Error. \nCalculated [K] Matrix is incorrect' )
end
end
end

```

`m_mdof_free_damped_vibration.m` for all the timesteps is affected by DOFs, sample rate and duration (Table 8). The `tic` and `toc` MATLAB commands were used to measure these calculation times.

DOF	Timing (secs) fs = 44.1 kHz duration = 2	Timing (secs) fs = 22.05 kHz duration = 2	Timing (secs) fs = 10 kHz duration = 2
10	0.63	0.61	0.51
20	0.88	0.59	0.57
50	1.95	1.14	0.79
100	7.38	3.03	1.89
200	20	10.39	5.25
300	43	22.6	10.8
400	79	40	18.8
800	308	154	72

Table 8 : System Responsiveness

The calculation time appears to be proportional to the square of DOF, as an increase from 200 to 400 results in four-fold response time, while sample rate and duration increase calculation time linearly. When interacting with the system, a response of 5 seconds or less gives a more immersive experience, and the combinations shown in green are indicative of parameter ranges that satisfy these criteria (based on a dual core Intel i5 processor, 8GB RAM).

8.2 VIBRATING STRING

As discussed in the Methodology Phase Two, this section compared the system's representation of vibrating string against a sound recording. In this case, an acoustic

guitar sample of a G3 note was considered. For a contextual comparison, we need consider the properties of acoustic guitar string that contribute to its sound [6]:

- The material, tension and length of the string.
- The physical properties of the bridge (where the string terminates) such as its rigidity as it transfers energy to the acoustic body and the decay due to other physical elements and components (nut).
- The axes of the string's vibration in traverse, longitudinal and torsional axes.
- The material and geometry of the acoustic chamber.
- How the string is plucked, either with a plectrum or finger(s).

As this model simplifies this behaviour, the audio comparison to the synthesised string will not sound the same as it does not fully exhibit the properties of a real guitar, but we should expect the first harmonics and overtones to be similar. We know from PM_EX03_04 (Appendix A.3) that a triangle-shaped displacement vector reasonably approximates a pluck. Additionally, frequency-dependent damping was added, which will ensure higher harmonics decay faster. Since the natural frequencies of a string are defined by its tension and mass per unit length, the M-Vector and K-Vector were adjusted by trial and error to reach a fundamental frequency (Appendix A, PM_EX02_01 and PM_EX02_02).

The following control parameters were used to create a plucked string using the interactive app (Figure 8.4):

CONTROL PARAMETERS:	
DOF	: 400
Boundary	: fixed-fixed
x0Vector	: SoftTriangle
v0Vector	: None
kVector	: Linear
mVector	: Linear
zVector	: Linear
k	: 247100.00
m	: 0.0000100
zeta	: 0.0010000
duration	: 2
fs	: 44100
Lpickup point	: 45
Rpickup point	: 1
pickup Method	: Single Mass
Stereo	: 0

Figure 8.4: G3 Note Control Parameters

The guitar sample was analysed using `pm_analyse_audio.m` (Appendix B.10) (which outputs the same graphical plots as used in the main application), allowing for fast comparison with the plots generated by the interactive application (Figure 8.5).

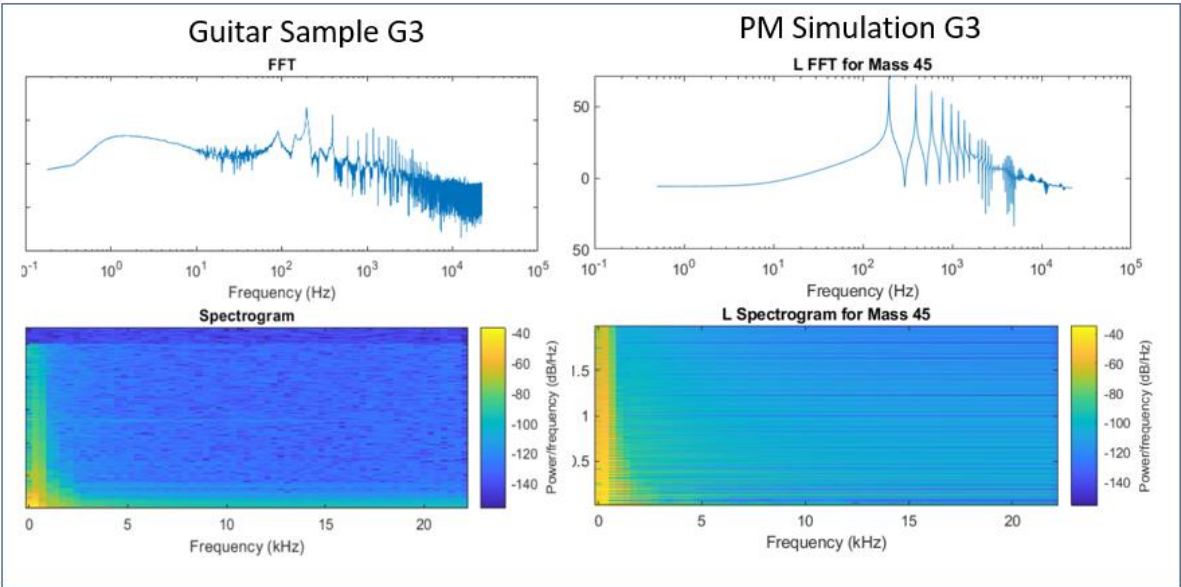


Figure 8.5: Frequency Analysis Comparison

The Guitar G3 FFT plot was marked up to measure the peaks harmonics and overtones (Figure 8.6).

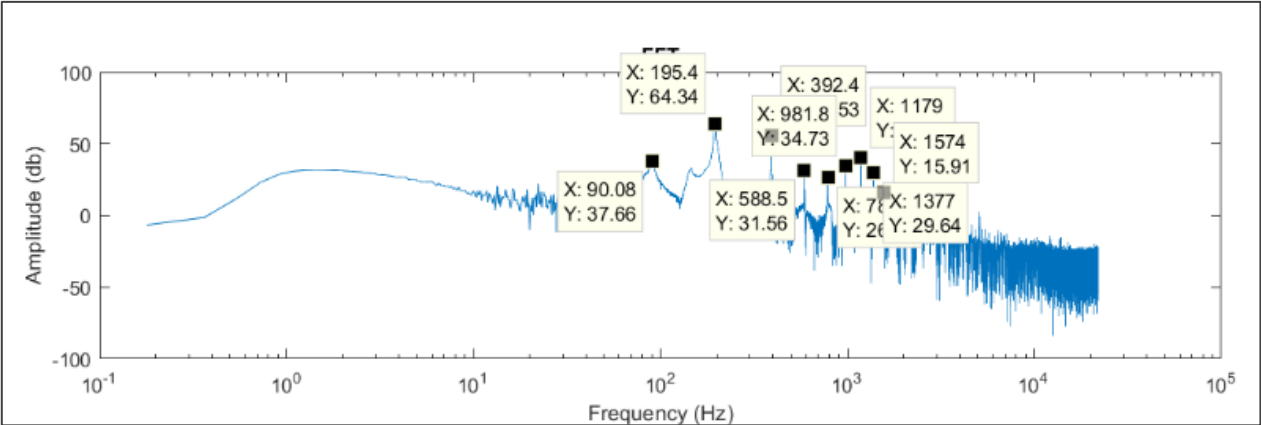


Figure 8.6: Guitar G3 Harmonics

The natural frequency modes for the simulation were then extracted from the textual use case report and compared with the guitar sample (Table 9).

Harmonic	Output Freq (Hz)	Guitar Sample (Hz)	% Difference
1	196	195.4	0.30

2	392	392.4	-0.10
3	587	588.5	-0.25
4	783	785.2	-0.28
5	979	981.8	-0.28
6	1176	1179	-0.25
7	1372	1377	-0.36
8	1568	1574	-0.38

Table 9: G3 Note Comparison

The natural frequencies from the application are all within 0.4% of the sample, which shows the harmonics for the physical model are accurate for a guitar string, although possibly demonstrating some inharmonicity. Additionally, the response of the harmonics series is similar, with the fundamental frequency being the loudest in both cases, as well as the decrease in amplitudes for higher harmonics appearing in both cases as well. Upon listening to both sounds, it is clear there is greater complexity in the sampled guitar, but enough accuracy in the physical model to demonstrate its validity as a simple string.

With this comparison in mind, it is now possible to consider ways to change this simple string model into an unphysical model.

9 PHASE THREE: UNPHYSICAL MODEL

The aim of Phase Three was to detail the methods for applying unphysical behaviour by using the Vector Profiles and Time-Varying Pickup in the interactive application.

9.1 DESIGN UNPHYSICAL BEHAVIOUR AND INTEGRATE COMPONENTS

9.1.1 Vector Profiles

Vector Profiles are names of profile shapes to be applied to the control parameters: M-Vector; K-Vector; Z-Vector; X-Vector; V-Vector. These profiles, documented in Appendix D, can be used either singularly to one of the control parameters or applied in combination to multiple parameters.

9.1.2 Time-Varying Pickup

The Time-Varying pickup (`pm_mdof_time_variant_pickup.m`, Appendix B.9) traverses the mass chain and samples a proportional “slice” of the overall amplitude response for each mass. The slices are then concatenated to give a resultant response as shown in Figure 9.1.

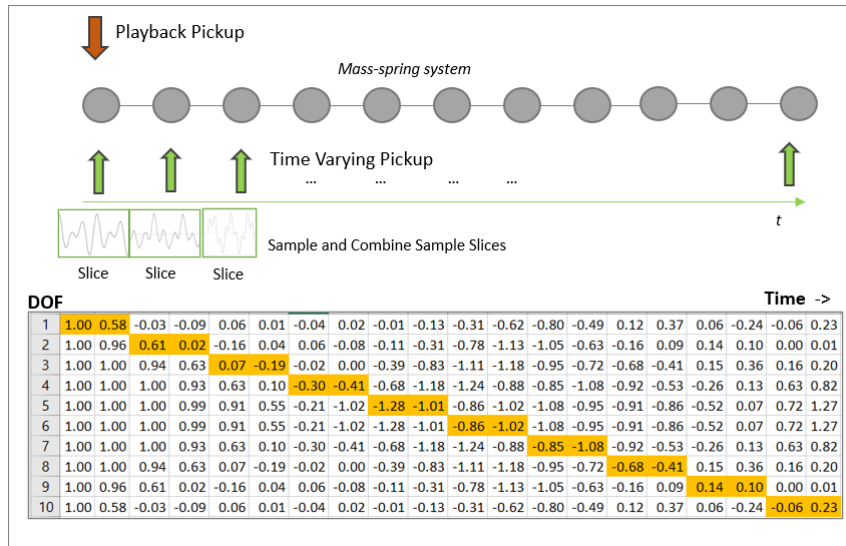


Figure 9.1: Time-Varying Pickup

The pickup can be applied in a single linear sweep or extended for multiple sweeps either resetting to the beginning or oscillating back and forth (Figure 9.2).

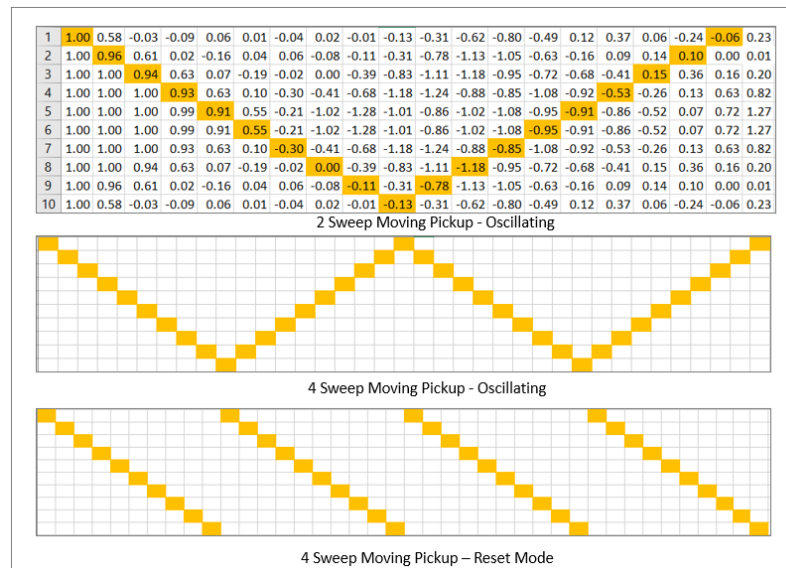


Figure 9.2: Time-Varying Pickup shapes

This mechanism offers an original method for applying non-physical behaviour to the system using the table of displacement/time for each mass.

9.1.3 Response-Driven Pickup

The Response-Driven Pickup extends the Time-Varying Pickup, whereby the amplitude response for a given mass describes the pickup's movement (Figure 9.3).

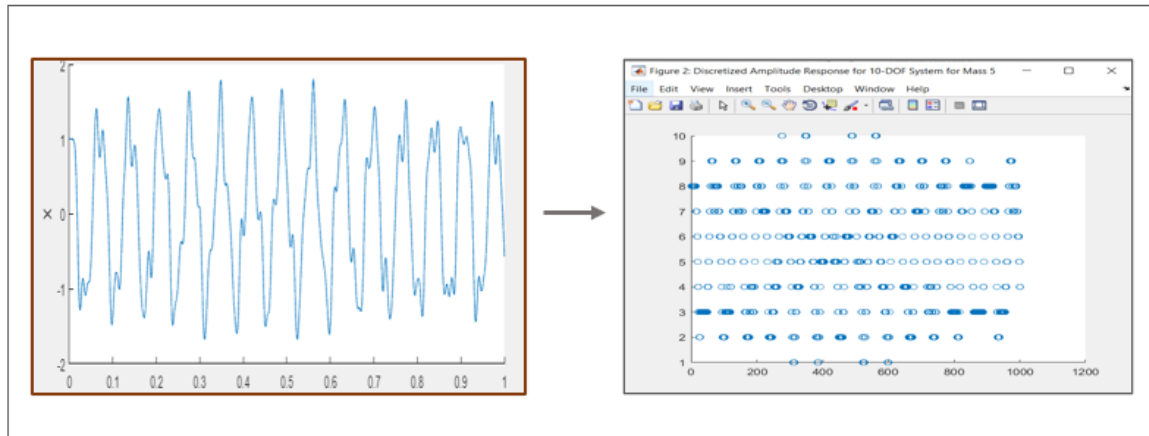


Figure 9.3: Amplitude Response Transformed

Figure 9.4, shows the algorithmic sequence to implement the Response Driven Pickup:

- User interacts with system and selects the mass which will drive the pickup position.
- System responds with an amplitude matrix of responses for every mass.
- The amplitude response for the pickup point is normalised in the range $\{-1$ to $+1\}$ and then discretised into integers and scaled, resulting in transformed values $\{1$ to $\text{DOF}\}$ (Figure 9.3).
- These discrete values dictate, for each time increment, where in the overall matrix the output values should be read to construct the response driven output.

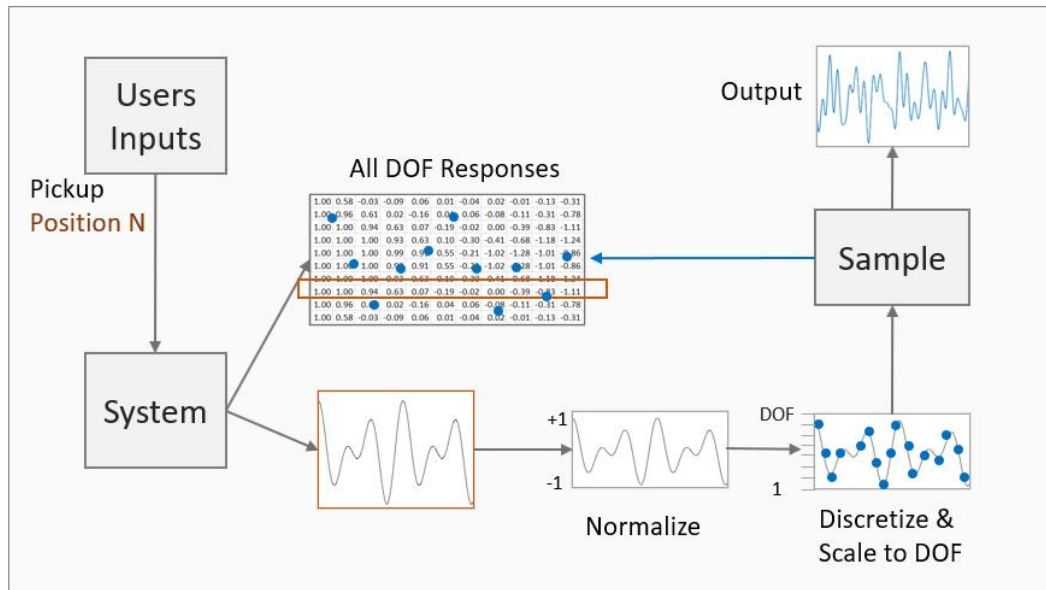


Figure 9.4: Responsive Driven Pickup Algorithm

9.2 UNPHYSICAL BEHAVIOUR

Each of the unphysical Vector Profiles were analysed separately for the X-Vector, Damping, K-Matrix and M-Matrix parameters, as were the Time-Varying Pickup types (Sweeps and Response-Driven). All cases were evaluated for their effectiveness at demonstrating unphysical behaviour.

9.2.1 Initial Displacement (X-Vector)

While a triangle-shaped initial displacement vector is more representative of a typical string pluck, there are other ways in which the initial conditions of the system can be used in an unphysical way. For this use case, the 'Step' vector was applied to the displacements (Figure 9.5). Also, stereo was used, with the pickups for left and right channels at Mass 9 and Mass 40 respectively.

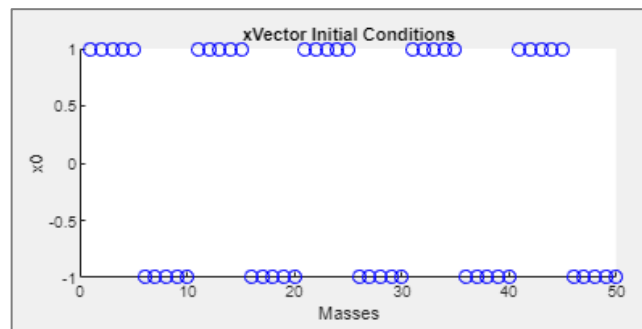
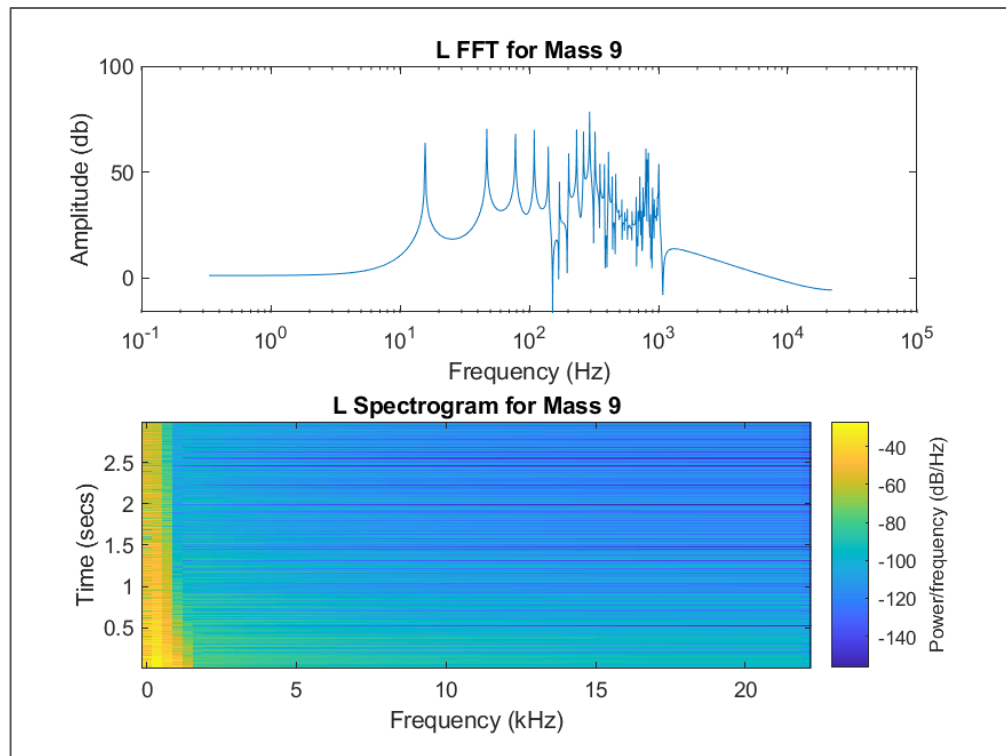


Figure 9.5: Initial Conditions for Unphysical X-Vector*Figure 9.6: Response for Unphysical X-Vector*

Since this vector roughly represents a square wave at approximately ten times the frequency of the fundamental, we could expect an emphasis on the tenth mode, and the even modes after (12th, 14th, 16th etc.). This is evident in Figure 9.6, where the highest amplitude is at approximately 293 Hz, the tenth mode.

9.2.2 Modal Damping Coefficients

Typically, modal damping for a simple string model will assume equal damping for each mode (Linear), or more accurately, greater damping for higher modes (Linear Increase). For an unphysical model, there could be the consideration that lower modes such as the fundamental frequency are damped more than the highest modes. This results in an interesting sounding model (Figure 9.7).

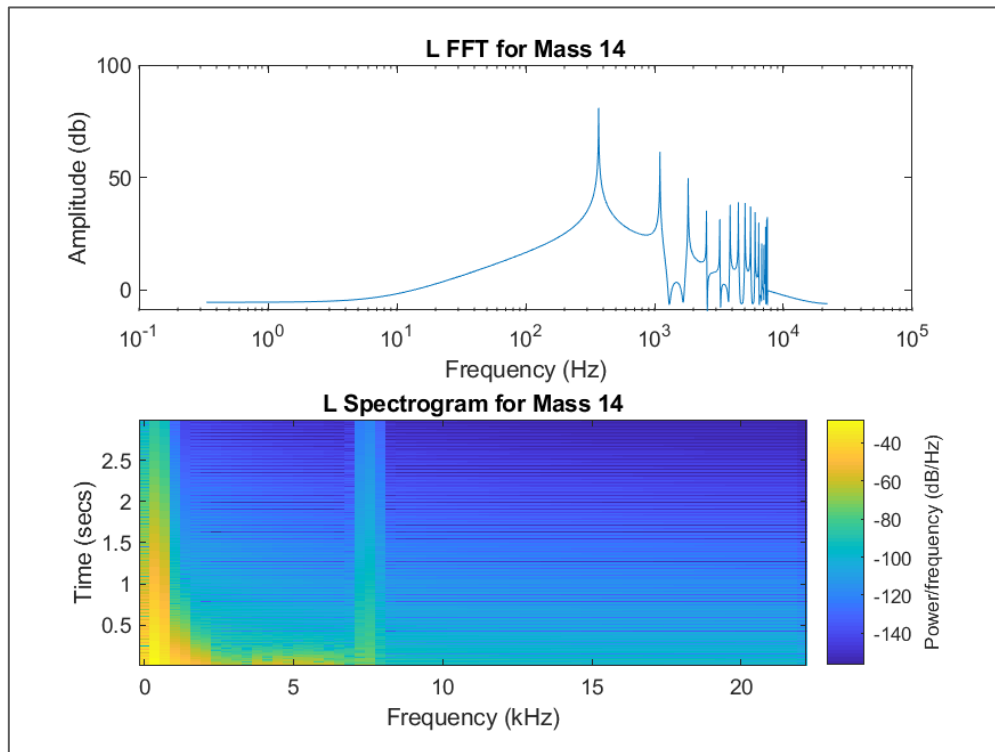


Figure 9.7: Response for Unphysical Modal Damping

As damping is reduced linearly for each mode, there is less decay at higher frequencies. This was most apparent for the highest natural frequency, 7.49 kHz. The sound resembled aliasing or down sampling, even though this is far below the Nyquist for this use case (22.05 kHz) or a more general high-frequency ringing. However, the overall response is unexpected given the damping is still lower for this highest mode than the fundamental, as seen in the spectrogram response near 0 Hz compared to 7.5 kHz. Despite this, the ringing sound is unexpected upon listening.

9.2.3 Stiffness K-Matrix/K-Vector

A string mass-spring model assumes equal spring stiffness across the entire system. This use case used the “N_2N_N_2N” vector for the spring stiffness values, meaning they alternate across the system (Figure 9.8).

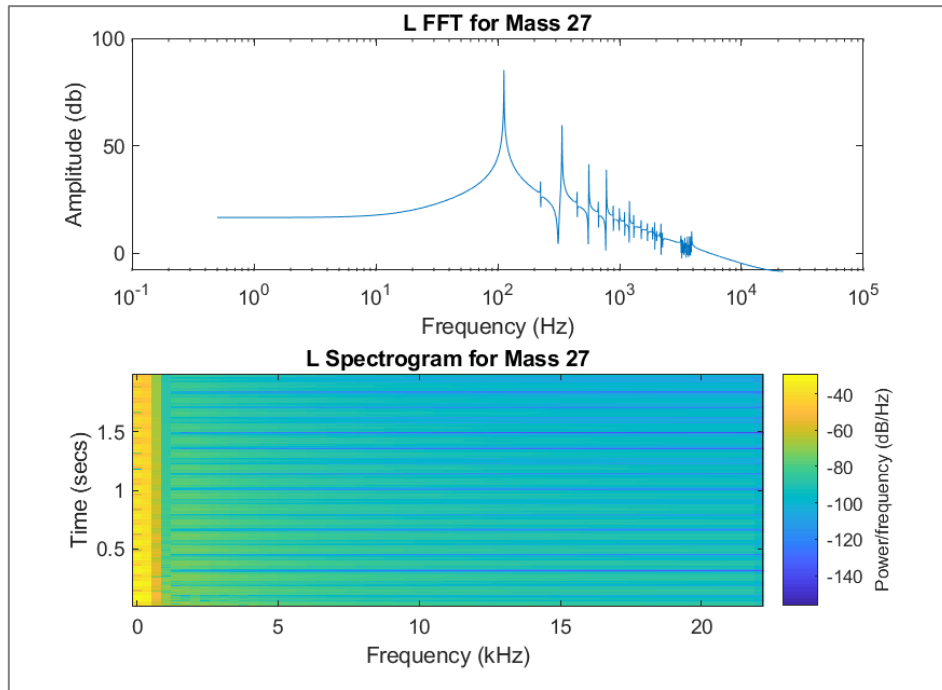


Figure 9.8: Response for Unphysical K-Matrix

The system appeared to behave like a string, where the alternating values average out for a high DOF system (100), meaning its response is not interesting.

9.2.4 Mass M-Matrix/M-Vector

A string mass-spring model assumes equal mass values. This use case applied the vector “N_2N_4N_8N” to a 100-DOF system (Figure 9.9). This is no longer a string-type model, as for an equivalent real string, the mass density would increase by a factor of $6.3e29$ along the length of the string, something that is clearly unphysical. The interesting factor to consider is how the natural frequencies are affected by this exponentially increasing density. It is important to note that the system has the same natural frequencies at all points, although the amplitudes will be significantly lower at heavier parts of the system.

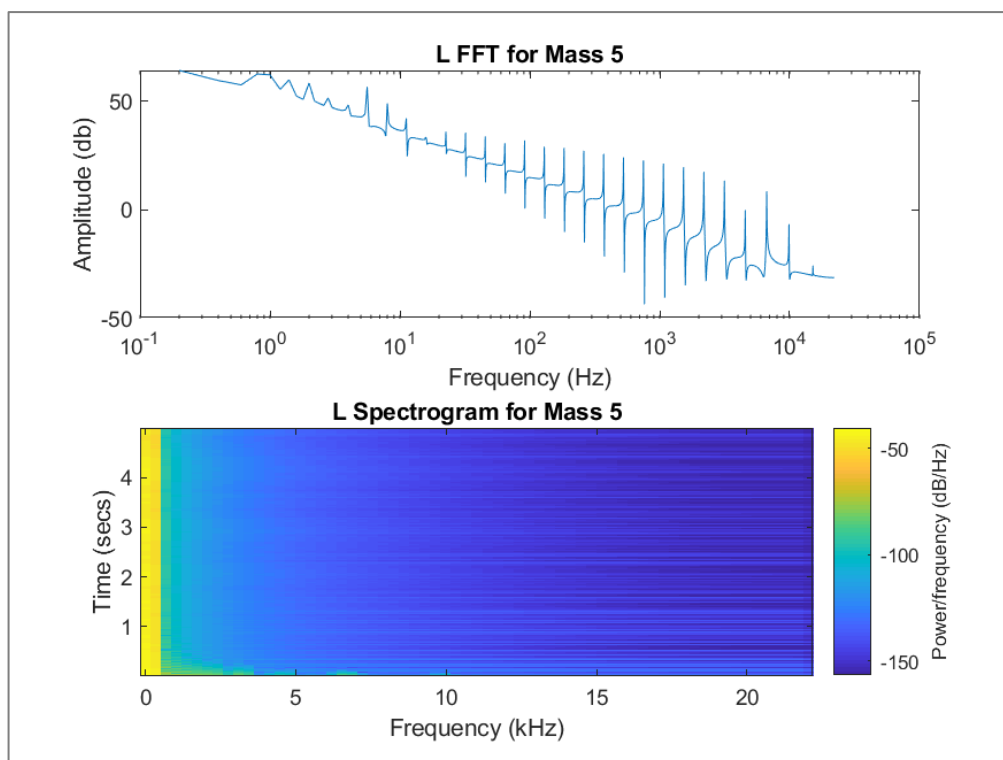


Figure 9.9: $M\text{-Vector} = N_2N_4N_8N$

The range of modes from the report range from 0 Hz up to 24.6 kHz, showing an exponential increase in the frequency of the natural modes. The sound generates resembles an FM synthesis-type quality.

9.2.5 Time-Varying Pickup

For a Time-Varying Pickup, the amplitude response for each mass will vary, resulting in a complex amplitude (Figure 9.10) and frequency response over time (Figure 9.11).

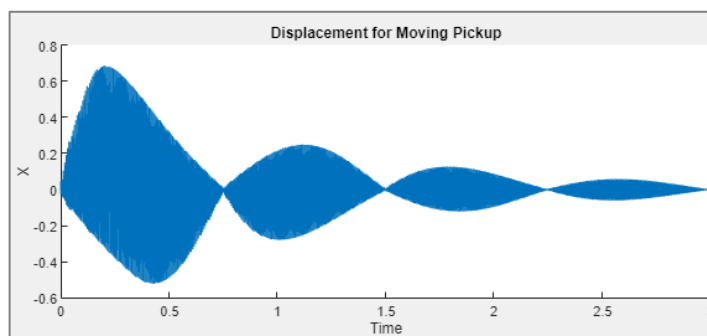


Figure 9.10: Amplitude Response for Time-Varying Pickup

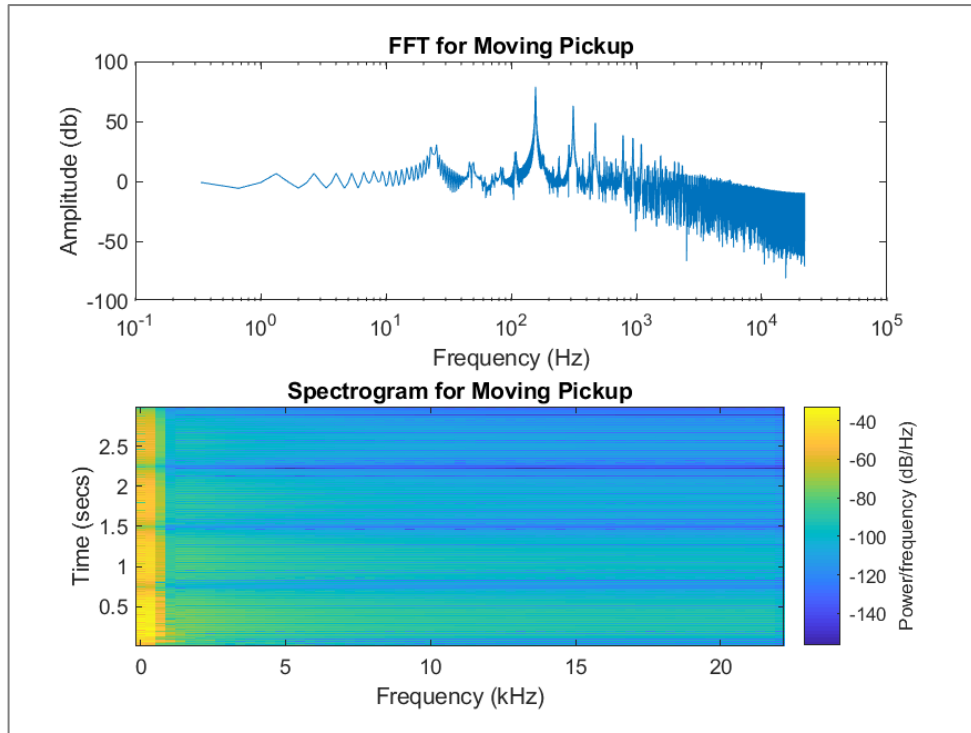


Figure 9.11: Response for Time-Varying Pickup

As apparent from Figure 9.10, there are four clear beats to the sound as the pickup moves along the system. The beating character is due to the masses closer to the boundaries having lower amplitudes to masses near the centre of the system. There is a large amount of additional noise as seen in the FFT plot, which is caused by the discrete movement and sharp transitions as the pickup moves between masses. Some form of interpolation could help address this issue (see Section 10.1: Pickup Improvements).

9.2.6 Response-Driven Pickup

This use case uses the second eigenvector as the initial displacement, which should result in a single sinusoid, in this case at 82.6 Hz. This condition dictates that the pickup moves as a sinusoid across the system at the same frequency, regardless of which mass is used for the pickup response.

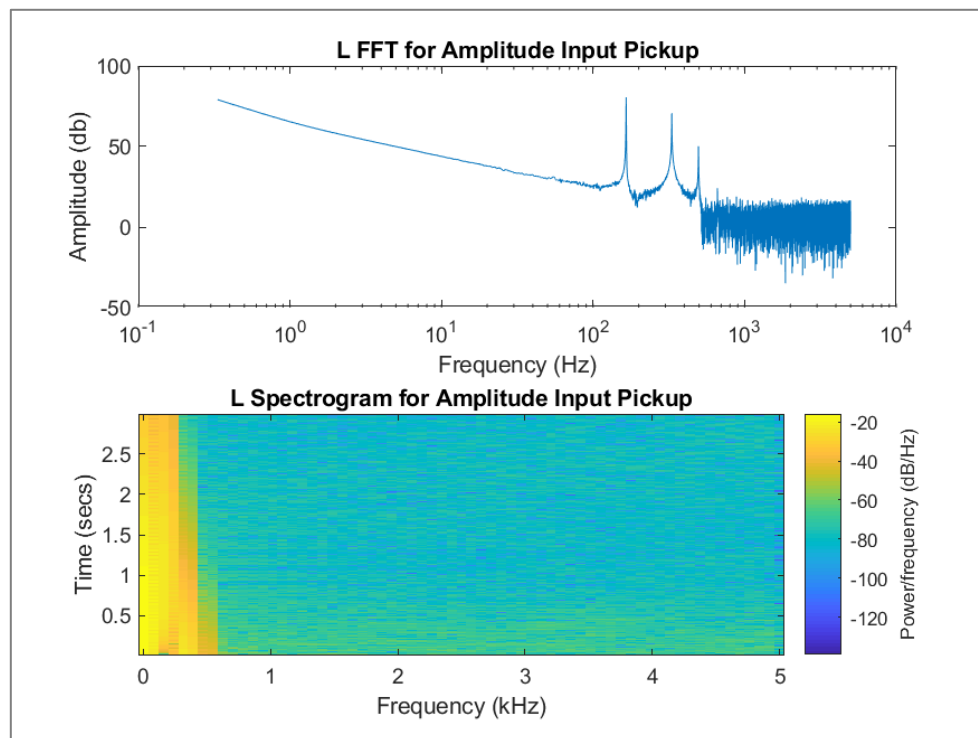


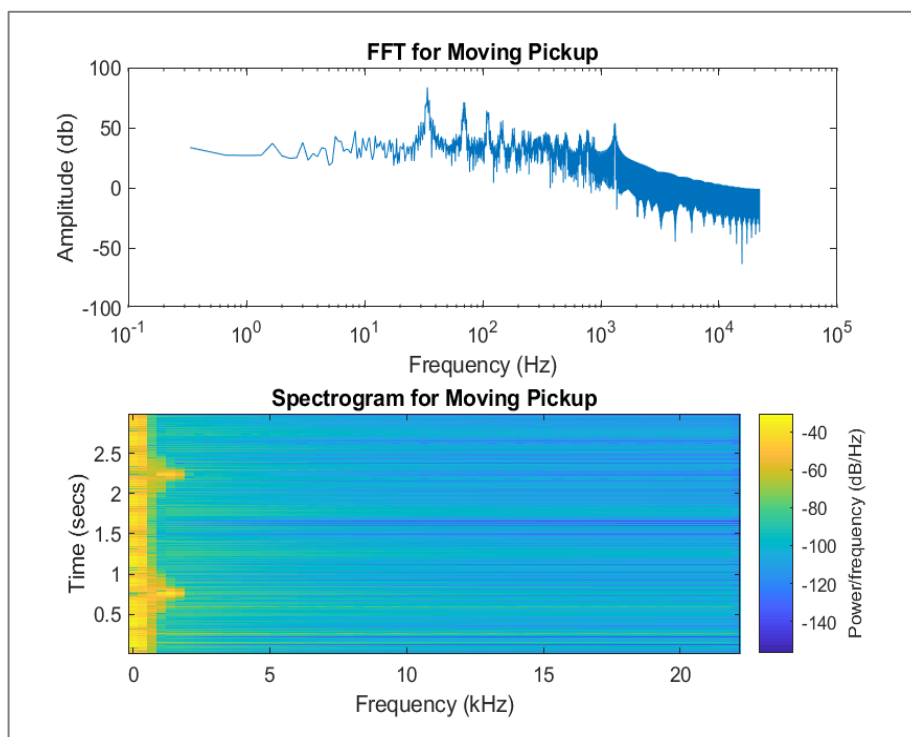
Figure 9.12: Response for Response Driven Pickup

Interestingly, there are resonant peaks at 165 Hz, 330 Hz, and 494 Hz, and the eigenvalue mode at 82.6 Hz is not apparent on the FFT. These other frequencies correspond to the 4th 8th and 12th natural modes for the system, which is unchanged by the pickup. This appears to be a form of amplitude modulation, where the signal and carrier wave are the same frequency. The rest of the sound is due to interpolation issues as discussed for the Time-Varying Pickup.

9.3 COMBINED UNPHYSICAL

Unphysical behaviours were combined to explore the sounds available from this application using the following Vector Profiles applied to multiple controls (Figure 9.14):

x0Vector	: Linear Increase
v0Vector	: 0,0,N,0,0
kVector	: N_N_N_2N_2N_2N_N_N_N
mVector	: Linear Decrease
zVector	: Linear Decrease
pickup Method	: Moving Linear

Figure 9.14: Vector Profiles for Combined Unphysical*Figure 9.15: Response for Combined Unphysical Model*

For this result (Figure 9.15), the sound is very abstract, in a similar sense to traditional sound synthesis and no clear relation to physical modelling. This demonstrates the sound design potential of unphysical modelling: a similar sound created using other synthesis techniques may take longer to produce a similarly complex sound, rather than using a set of unphysical behaviours.

10 FURTHER WORK

10.1 PICKUP IMPROVEMENTS

While not directly related to improving the unphysical behaviour of the system, improvements to the output would greatly improve the sound quality for any moving pickup output. This could be achieved using several techniques:

I. Introduce interpolation to the pickup response.

This would attempt to smooth out the response for any moving pickup by considering the continuous movement of the pickup. The output sound will be the sum of the two nearest masses displacements, based on their distance from the pickup on each sample (Figure 10.1). As the pickup moves, the coefficients of each mass displacement will change.

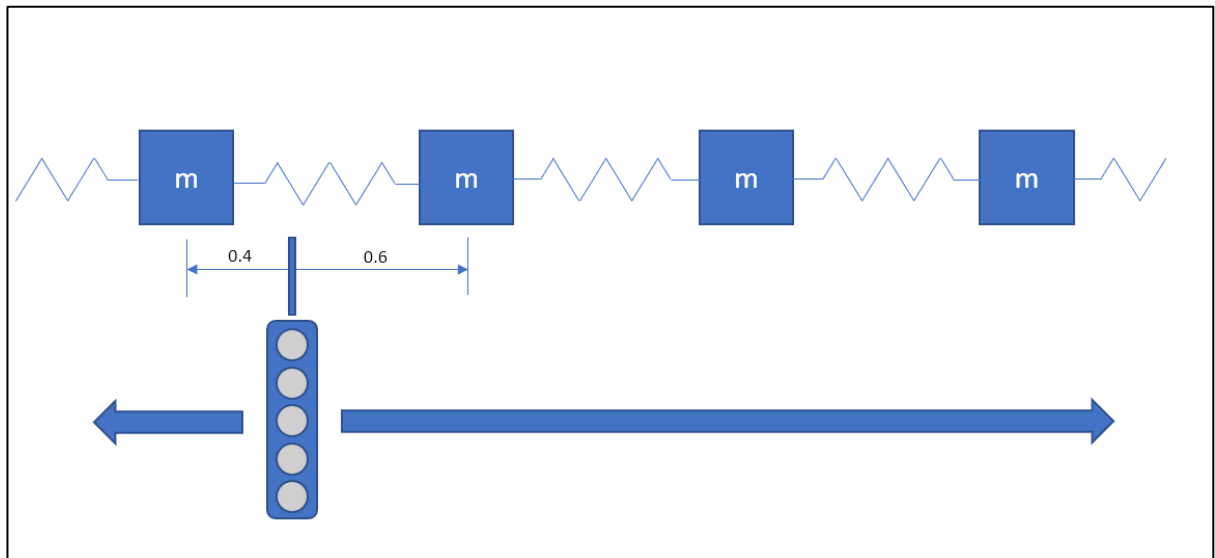


Figure 10.1: Interpolated Moving Pickup

II. Sum the pickup response shape.

A further improvement would be to consider more than two mass responses for the sound output. Several approaches could be taken as all mass amplitudes are calculated regardless of the desired output, such as using the Vector Profiles to describe the amount that each mass is used in the summed response. A Vector Profile could be used on a section of the system and then moved, utilising interpolation.

10.2 LINEAR TIME-VARIANT (LTV) PARAMETERS

This type of unphysical behaviour could include mass values changing over time. One simple example in a 1-DOF system would be measuring the oscillation of a leaky mass (such as a leaking bucket of water) attached to a spring [28]. As the mass decreases, the frequency of the SHM would increase over time. This could similarly be applied to an N-DOF system.

However, typical modal analysis is invalid for LTV calculations. This would require adapting the vibration calculations from modal to a state-space solution (such as for a forced excitation), which would allow the introduction of mass, spring and damping values as a function of time.

10.3 FORCED EXCITATION

Forced excitation is any external force applied to the system. For a string, this includes the use of a pluck, bow or hammer. Forces can often give a more accurate physical model, especially when considering the transient component of a sound. These can also be applied to an unphysical model with the introduction of more novel forces being applied to it. For example, a forced oscillation of a mass in an N-DOF system could greatly affect its output, especially for any frequencies close to or the same as any of the system's normal modes.

Work had begun to implement forced excitation into the app, but it was not possible to fully integrate due to time constraints. The forcing function below (pm_mdof_forced_vibration.m) used the MATLAB state-state `ss()` and simulation functions `lsim()` (See Appendix C for the initial experiment).

```
function [amplitudes]=pm_mdof_forced_vibration(M,K,Z,Forces,x0,v0,t)
    % create state space parameters
    zeroM = zeros(size(M));
    A=[zeroM, M ; -M\K, -M\Z];
    B=[zeroM;inv(M)];
    C=eye(2*size(M));
    D=0*B;
    sys=ss(A,B,C,D);
    y=lsim(sys,Forces',t',[x0';v0'],'foh'); % first order hold
    amplitudes=y(:,1:size(M,1))';
end
```

As this is no longer a modal analysis solution, the damping coefficients are now used as per the N-DOF mass-spring-damper system, where dampers are parallel to each spring. Therefore, the Z-matrix construction follows the same rules as the K-matrix. Appendix C shows a short cosine forced excitation to mass 5. The forced function could be changed to any shape, for example a simulated V shaped hammer impact across several masses, at any period range between 0 and T.

10.4 EFFICIENCY IMPROVEMENTS

Techniques could be used to improve the performance of the app which shows an exponential increase in calculation times (Table 8). For example, there is a clear possibility to implement parallel computation into the main modal calculation. The amplitude response for each mode is commutative so being able to calculate many modes simultaneously using multi-threading could help to address times using the MATLAB parallel computing toolbox. This was not possible to test, but one example would be to replace a for-loop with a parallel for-loop in `pm_mdof_free_damped_vibration.m`.

10.5 LISTENING TESTS

For a more robust system, a listening test could be devised to assess perceptions of unphysical modelling. Participants could be instructed to consider whether a sound is perceptible as a possible real-world sound or resembles impossible characteristics. Feedback given could also aid in considering the applications of unphysical modelling, such as in the context of audio-visual cues. An unphysical model paired with visual representation may seem illogical in some cases, or even offer no clear visual counterpart at all.

11 CONCLUSION

The project was evaluated against the aims described in section 6.1.

11.1 PLANNING

Project implementation was successfully completed well in advance of the submission date, with the remaining time left to test and generate results. In part, this was due to the use of an Agile methodology, and the use of Jira to track all programming-related work. This approach mostly allowed for the allocation and completion of smaller, achievable tasks with some attempts to log time spent on each task. This was especially apparent for Phase One, where the experimental approach to each function ensured all components behaved as expected.

11.2 IMPLEMENTATION AND ITERATION

The final interactive app proved to be stable for its purpose and enabled efficient sound design through features such as presets, previous/next controls and error handling. Also, the plots inside the app helped to evaluate the system's behaviour outside of the sounds produced. Deciding on creating a single final app also gave the opportunity to explore greater varieties of sound, beyond what would be possible if only one parameter could be explored.

However, efforts made into improving the app's usability could have been better spent on adding features to improve the quality of the results and sounds generated, which were overall a more important facet to this project.

11.3 DEMONSTRATION OF THEORY

Fortunately, modal analysis of mass-spring systems proved to be an ideal technique for this project. By utilising clear hierarchical programming, all mathematical calculation was implemented in distinct functions, with the GUI elements creating a separate front-end for interaction.

11.4 TESTING, ANALYSIS AND DISCUSSION

- Phase One introduced and tested each function used in the final Interactive App.
- Phase Two implemented these functions as a backend for GUI as well as testing the viability as an accurate simple string model, with the model appearing to demonstrate a reasonable level of accuracy required for this project.
- Phase Three tested unphysical behaviour on each parameter successfully, with each sounding unique and interesting. For the moving pickup, there was a strong initial concept, but improvements will need to be made to generate useful sounds.

11.5 REVIEW OF PROJECT IMPACT / VIABILITY

Due to the computational demands of modal analysis for sound synthesis, this model's usefulness only applies to its ability to generate sounds, not as an instrument. For this model to function as an instrument, there would have to be considerations for alternative methods of physical modelling to achieve enough performance without sacrificing the fidelity of the system.

It appears that for this system, physical modelling is a subset of the unphysical modelling. Furthermore, there is a specific domain for which the unphysical model is useful: the viability in shifting the paradigm of physical modelling from accuracy, stability, efficiency and performance to focus on sound design and creativity.

12 REFERENCES

- [1] J. A. Duarte, (2007), 'NEW TRENDS ON PHYSICAL MODELING OF MUSICAL INSTRUMENTS', [Online]. Available:
<https://ir.nctu.edu.tw/bitstream/11536/81946/1/651901.pdf>
- [2] S. S. Rao, *Mechanical vibrations*, 5th ed. Upper Saddle River, N.J: Prentice Hall, 2011.
- [3] J. O. Smith III, (2006), 'Physical Modeling Sound Synthesis', [Online]. Available:
<https://ccrma.stanford.edu/~jos/pdf/AES-Masterclass.pdf>
- [4] O. Darrigol, 'The acoustic origins of harmonic analysis', *Arch. Hist. Exact Sci.*, vol. 61, no. 4, pp. 343–424, Jun. 2007, doi: 10.1007/s00407-007-0003-9.
- [5] M. Ruzzene, 'One-Dimensional (1D) and Two-Dimensional (2D) Spring Mass Chains', [Online]. Available:
http://www.ruzzene.gatech.edu/CISM_Course/Course_Slides_files/Ruzzene02.pdf
- [6] M. D. Tuttle, (2007), 'Plucked Instrument Strings', [Online]. Available:
<https://www.semanticscholar.org/paper/Plucked-Instrument-Strings%3A-A-Combined-Frequency-to-Tuttle/499de3015bb29adebbb905ae493fd9056b4f5d30>
- [7] S. Bilbao, *Numerical Sound Synthesis*. Chichester, UK: John Wiley & Sons, Ltd, 2009.
- [8] D. Jaksch, 'Waves'. 2006, [Online]. Available:
<http://www.physics.ox.ac.uk/groups/qubit/tutes/Waves.pdf>.
- [9] Thomas D. Rossing, *Principles of Vibration and Sound*, Second Edition. Springer-Verlag, 2004.
- [10] J. O. Smith III, 'Singing Kelly-Lochbaum Vocal Tract'. [Online]. Available:
https://ccrma.stanford.edu/~jos/pasp05/Singing_Kelly_Lochbaum_Vocal_Tract.html.
- [11] L. Hiller and P. Ruiz, 'Synthesizing Musical Sounds by Solving the Wave Equation for Vibrating Objects: Part 1', *J. Audio Eng. Soc.*, vol. 19, no. 6, pp. 462–470, 1971.

- [12] K. Karplus and A. Strong, ‘Digital Synthesis of Plucked-String and Drum Timbres’, *Computer Music Journal*, vol. 7, no. 2, p. pp.43-55, 1983.
- [13] Smith III, J. O. and Jaffe, D., ‘Extensions of the Karplus-Strong Plucked-String Algorithm’, *Computer Music Journal*, vol. 7, no. 2, p. pg. 56-69, 1983.
- [14] Julius O. Smith, ‘A Basic Introduction to Digital Waveguide Synthesis (for the Technically Inclined)’. CCRMA, Feb. 23, 2006, [Online]. Available: <https://ccrma.stanford.edu/~jos/swgt/>
- [15] A. Crute, (2019) ‘Understanding physical modelling synthesis’, *MusicTech*, [Online]. Available: <https://www.musictech.net/guides/essential-guide/understanding-physical-modelling-synthesis/>.
- [16] C. Cadoz, A. Luciani and J. Florens, “CORDIS-ANIMA: A Modeling and Simulation System for Sound and Image Synthesis: The General Formalism,” *Computer Music Journal*, vol. 17, no. 1, pp. 19-29, 1993.’
- [17] L. Leonard and J. Villeneuve, (2019), ‘FORMALIZING MASS-INTERACTION PHYSICAL MODELING IN FAUST,’ [Online]. Available: <https://ccrma.stanford.edu/~rmichon/publications/doc/LAC-19-MI.pdf>.
- [18] D. M. Howard and S. Rimell, ‘Real-Time Gesture-Controlled Physical Modelling Music Synthesis with Tactile Feedback’, *EURASIP J. Adv. Signal Process.*, vol. 2004, no. 7, p. 830184, Dec. 2004, doi: 10.1155/S1110865704311182.
- [19] J. He and Z.-F. Fu, *Modal analysis*. Boston: Butterworth-Heinemann, 2001.
- [20] J. Morrison and J. Adrien, “MOSAIC: A Framework for Modal Synthesis,” *Computer Music Journal*, vol. 17, no. 1, pp. 45-46, 1993.
- [21] P. Djoharian, “Generating Models for Modal Synthesis,” *Computer Music Journal*, vol. 17, no. 1, pp. 57-65, 1993.
- [22] J. Georgii and R. Westermann, “Mass-Spring Systems on the GPU,” *Computer Graphics & Visualization Group, Technische Universitat Munchen*, 2005.
- [23] S. Bilbao *et al.*, “The NESS Project: Physical Modeling, Algorithms and Sound Synthesis”, [Online]. Available: <https://www.ness.music.ed.ac.uk/project>

- [24] P. G. BAKIR, 'Two degree of freedom systems.pdf'. [Online]. Available: <https://web.itu.edu.tr/~gundes/2dof.pdf>.
- [25] D. Morgan and S. Qiao, 'Analysis of Damped Mass-Spring Systems for Sound Synthesis', *EURASIP J Audio Speech Music Process*, vol. 2009, no. 1, p. 947823, 2009, doi: 10.1155/2009/947823.
- [26] T. R. Teja and S. K. Chaitanya, 'Computation Of Natural Frequencies Of Multi Degree Of Freedom System', *International Journal of Engineering Research*, vol. 1, no. 10, p. 5, 2012.
- [27] O. Hazzan and Y. Dubinsky, *Agile Software Engineering*, Springer, 2008.
- [28] H. Rodrigues, N. Panza, D. P. Jr, and A. Soares, 'A model of oscillator with variable mass', *Rev. Mex. Fis.*, p. 8, 2014.

13 STATEMENT OF ETHICS

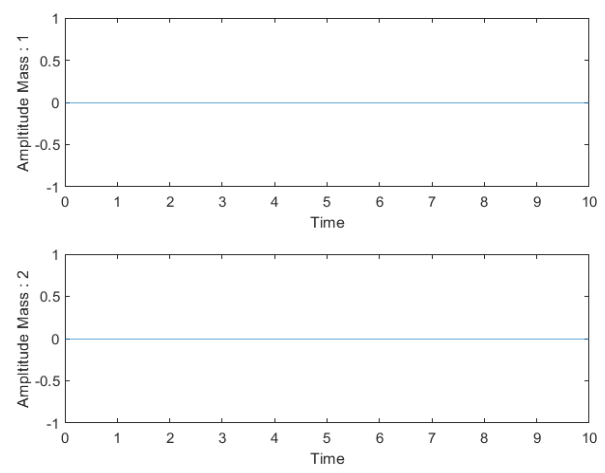
No ethical issues were identified for this project

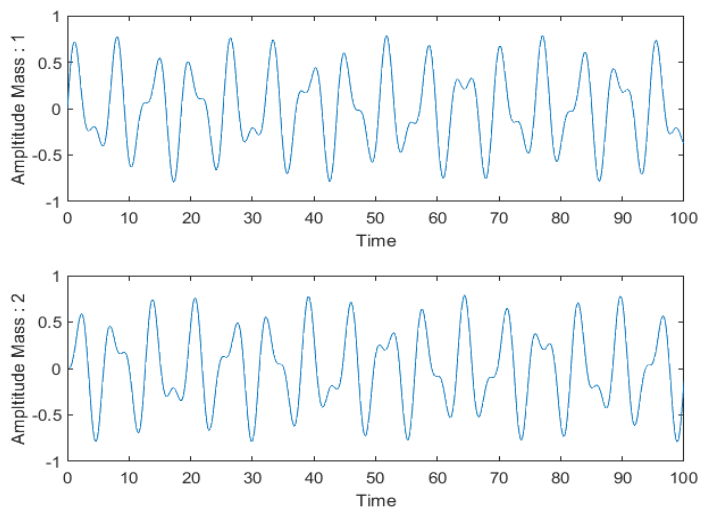
Appendix A: EXPERIMENTS

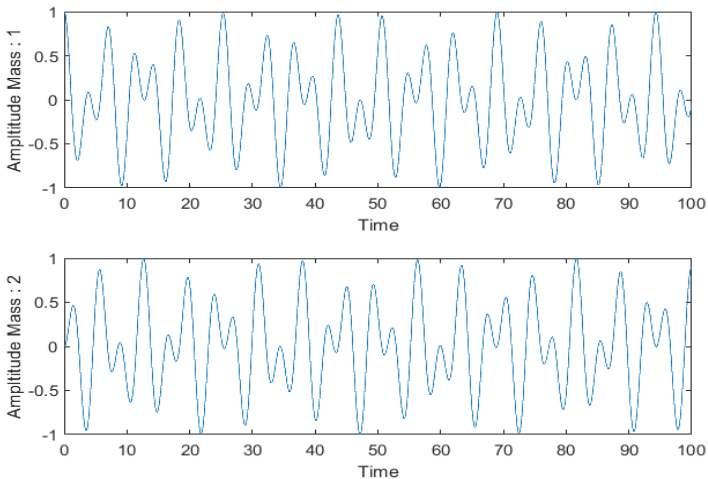
A.1 EXPERIMENT SET 1

Experiment Set	PM_EX01
Aim	<p>Develop a MATLAB function to apply and verify the following equation of motion.</p> <p><i>Free Vibration Modal Equation</i></p> $x(t) = \sum_{i=0}^{n=2} \{\psi\}_i (\{\psi\}_i^T [M] x(0) \cos \omega_i t + \frac{1}{\omega_i} \{\psi\}_i^T [M] \dot{x}(0) \sin \omega_i t)$
JIRA Task	JIRA Link: PM-3 Create 2 DOF MSS
MATLAB Changes	Error! Reference source not found., Error! Reference source not found.
Overall Summary	Based on the test cases the created MATLAB function is correctly computing the motion of the 2-DOF system.

Experiment ID	PM_EX01_01
Test	Validate the Response is zero for the Initial Conditions of no displacements or velocities.
Input parameters	x0(0,0), v0(0,0)
Other Parameters	As per the experiment invariants Definition PM_EX01.
MATLAB Changes	<p>Create a new MATLAB test function pm_mdof_free_vibration_ex01_test01.m. For complete definition see REPORT_SUBMISSION\Experiments\pm_exp01</p>

Results	 <p>The figure consists of two vertically stacked plots. The top plot is titled 'Amplitude Mass : 1' and the bottom plot is titled 'Amplitude Mass : 2'. Both plots have 'Time' on the x-axis (ranging from 0 to 10) and 'Amplitude' on the y-axis (ranging from -1 to 1). In both plots, a single horizontal blue line is drawn at y=0, indicating zero amplitude for both masses over the entire time range.</p>
Conclusion	The response is as expected, with no initial conditions there is no amplitude displacements of either masses.

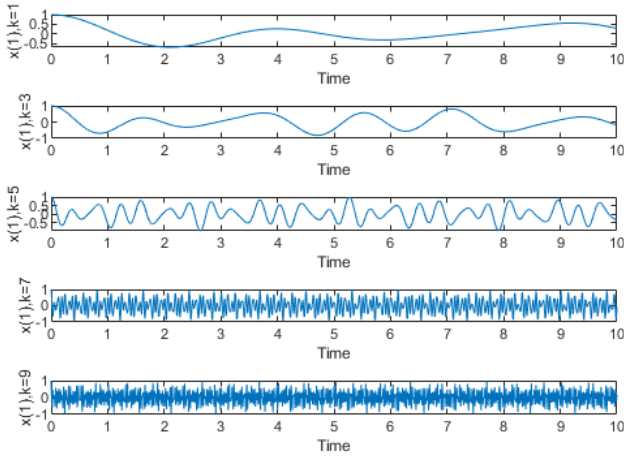
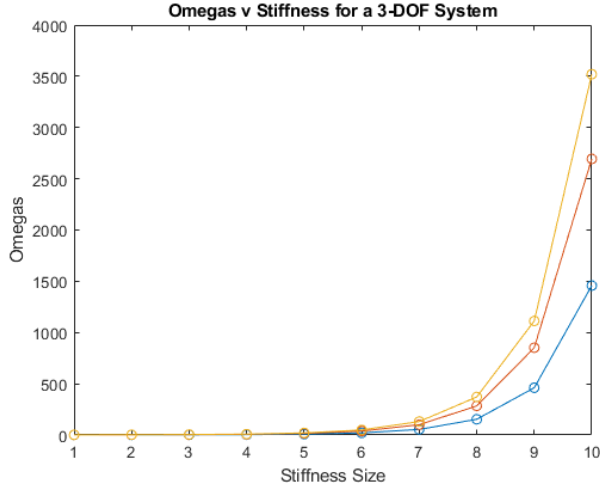
Experiment ID	PM_EX01_02
Test	Validate the Response is non-zero for the initial conditions of no displacements but with a velocity of 1 for the first mass.
Input parameters	$x_0(0,0)$, $v_0(1,0)$
Other Parameters	As per the experiment invariants.
MATLAB Changes	Create a new MATLAB test function <code>pm_mdof_free_vibration_ex01_test02.m</code> .
Results	 <p>The figure consists of two vertically stacked plots. The top plot is titled 'Amplitude Mass : 1' and the bottom plot is titled 'Amplitude Mass : 2'. Both plots have 'Time' on the x-axis (ranging from 0 to 100) and 'Amplitude' on the y-axis (ranging from -1 to 1). Both plots show complex, non-periodic oscillations. The oscillations appear to be a superposition of multiple frequencies, with the amplitude varying between approximately -0.8 and 0.8.</p>
Conclusion	The Amplitude Response is of each is non periodic and through visual inspection this appears a superposition of the velocity component of the equation (two sine waves of different frequencies).

Experiment ID	PM_EX01_03
Test	Validate the Response is non-zero for the Initial Conditions of displacements of 1 for the first mass but with no velocities.
Input parameters	$x_0(1,0)$, $v_0(0,0)$
Other Parameters	As per the experiment invariants PM_EX01.
MATLAB Changes	Create a new MATLAB test function <code>pm_mdof_free_vibration_ex01_test03.m</code> . For complete definition see REPORT_SUBMISSION\Experiments\pm_exp01
Results	
Conclusion	The Amplitude Response is of each is non periodic and through visual inspection this appears a superposition of the displacement component of the equation (two cosine waves of different frequencies). The mass 1 correctly shows an amplitude of 1 at $t=0$ and mass 2 correctly shows an amplitude of 0 at $t=0$.

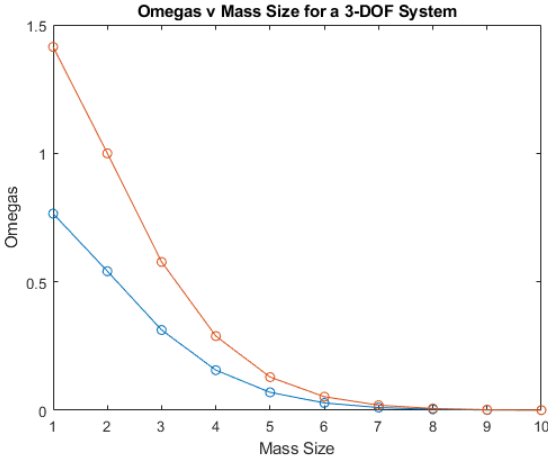
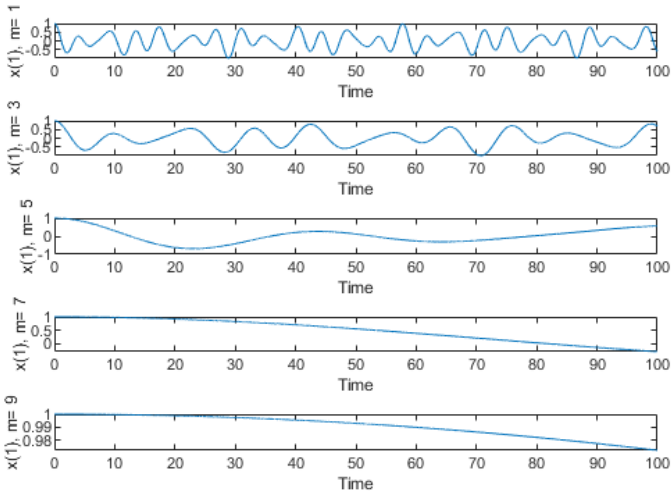
A.2 EXPERIMENT SET 2

Experiment Set	PM_EX02
Aim	Test the validity of a 3-DOF System's normal modes using MATLAB, using the <code>pm_mdof_free_vibration.m</code> and demonstrate normal modes
JIRA Task	JIRA Link: PM-37 PM Experiment 2 - 3 DOFS
MATLAB Changes	The function now models a 3-mass, 4-spring system. <code>pm_mdof_free_vibration_modified.m</code> is a variant of Error! Reference source not found. such that it will return the amplitude slices for a participating mass. Four test cases created.
Overall Summary	Based on the test cases the MATLAB function is correctly computing the 3-DOF response, showing normal modes and superposition. Extra tests created for response based on mass and stiffness changes.

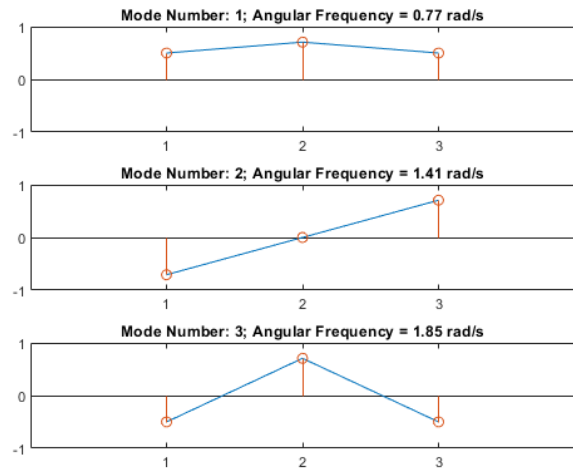
Experiment ID	PM_EX02_01
Test	Effects of varying spring stiffness values on the normal modes of a 3-DOF mass-spring system.
Input parameters	$M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ $K = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$ (multiplied for each iteration)
Other Parameters	$x_0 = [1, 0, 0]'$; No initial velocity.
MATLAB Changes	Create new MATLAB function: <code>pm_mdof_control_variance_exp02_test01.m</code> Change plots to show relationship between spring stiffness and frequency and the effects on amplitude plots for different stiffness. For complete definition see REPORT_SUBMISSION\Experiments\pm_exp02

RESULTS	
<div> </div>	
Conclusion	Increasing spring stiffness increases the frequencies of its normal modes.

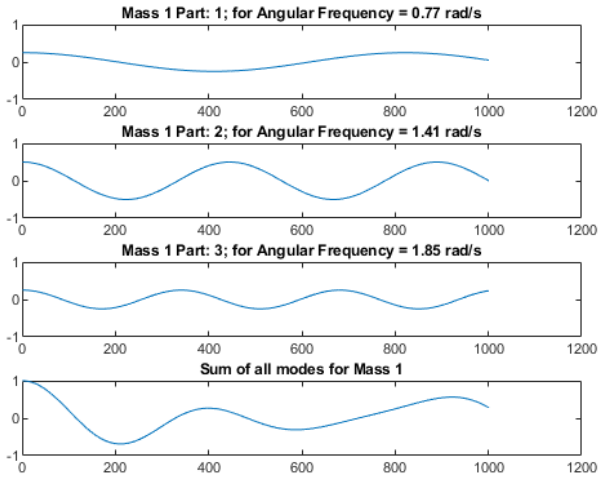
Experiment ID	PM_EX02_02
Test	Effects of varying mass values on the normal modes of a 3-DOF mass-spring system.
Input parameters	$M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ (multiplied for each iteration e.g. 1,2,3 etc.) $K = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$
Other Parameters	$x0 = [1, 0, 0]'$; No initial velocity.
MATLAB Changes	Create new MATLAB function: pm_mdof_control_variance_exp02_test02.m Change plots to show relationship between mass values and frequency and the effects on an amplitude plots for different stiffness.

	For complete definition see REPORT_SUBMISSION\Experiments\pm_exp02
RESULTS	
<div></div>	
Conclusion	Increasing mass values decreases the frequencies of its normal modes.

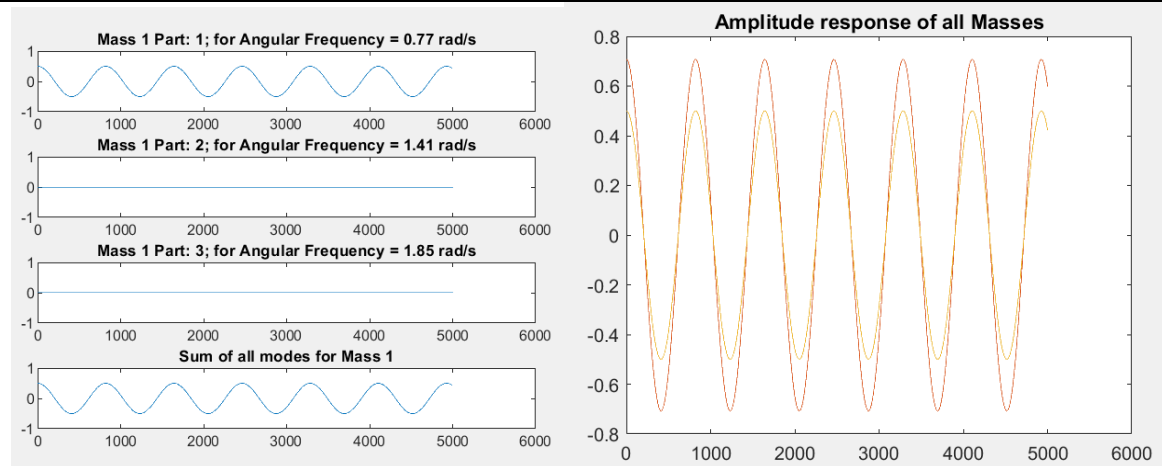
Experiment ID	PM_EX02_03
Test	Validate correct number of modes, with correct values for 3-DOF system.
Input parameters	$x_0 = [1, 0, 0]^T$;
Other Parameters	n/a
MATLAB Changes	Create new MATLAB function: pm_mdof_control_variance_exp02_test03.m Change plots to show normal modes for 3-DOF system, derived from the eigenvectors and eigenvalues of the system.

	For complete definition see REPORT_SUBMISSION\Experiments\pm_exp02
RESULTS	
 <p>Mode Number: 1; Angular Frequency = 0.77 rad/s</p> <p>Mode Number: 2; Angular Frequency = 1.41 rad/s</p> <p>Mode Number: 3; Angular Frequency = 1.85 rad/s</p>	
Conclusion	These results agree with the theory for normal modes, showing for a 3-DOF system there are three normal modes. It can be noted for this odd-DOF system there is a stationary point in the middle for mode 2.

Experiment ID	PM_EX02_04
Test	Validate amplitude response of mass 1 is a superposition of all normal modes (whose coefficients are dependent on the initial conditions of the system).
Input parameters	$x_0 = [1, 0, 0]^T$;
Other Parameters	n/a
MATLAB Changes	<p>Create new MATLAB functions: pm_mdof_control_variance_exp02_test04.m</p> <p>Change plots to show amplitude response of each normal mode for a 3-DOF system, and their resulting overall amplitude response for the corresponding mass</p> <p>For complete definition see REPORT_SUBMISSION\Experiments\pm_exp02</p>

RESULTS	
<div></div> <div><pre>pm_mdof_superposition_exp02_test04 Eigenvectors (as columns) 0.5000 -0.7071 -0.5000 0.7071 0.0000 0.7071 0.5000 0.7071 -0.5000</pre></div>	
Conclusion	<p>From visual inspection, the angular frequency for each part is equal to the angular frequency derived from the eigenvalues.</p> <p>For first part: Natural frequency = 0.1218 Hz 1/0.1218 = 8.21 seconds. This is equivalent to 821 samples for the calculated outputs, which is correct.</p>

Experiment ID	PM_EX02_05
Test	Test response on eigenvector as initial amplitude for previous test plot, verifying the output only consists of one mode frequency.
Input parameters	$x_0 = [0.5, \sqrt{2}/2, 0.5]'$; These values derived from the first eigenvector in test 04.
Other Parameters	n/a
MATLAB Changes	<p>Create new MATLAB functions: pm_mdof_control_variance_exp02_test05.m</p> <p>For complete definition see REPORT_SUBMISSION\Experiments\pm_exp02</p>

RESULTS

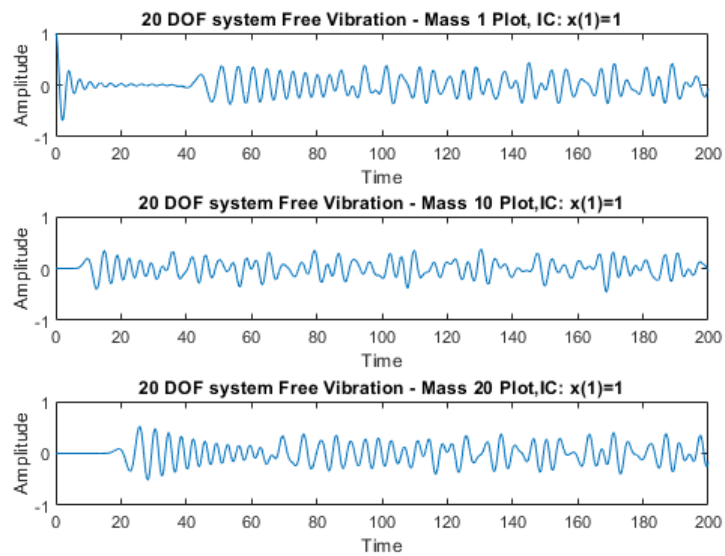
Conclusion	<p>Due to the initial displacements of each mass corresponding to the first eigenvector, the only frequency component is the first mode.</p> <p>The second plot shows the amplitude response of each mass. Note that mass 1 and 3 have the same values whereas mass 2 has higher amplitude as per the plot of mode 1 from experiment 3.</p>
-------------------	---

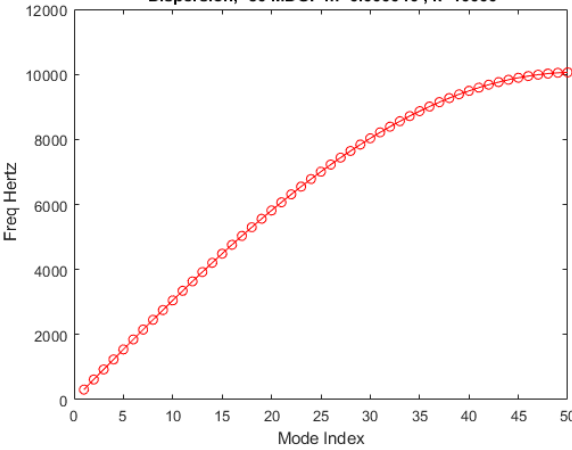
A.3 EXPERIMENT SET 3

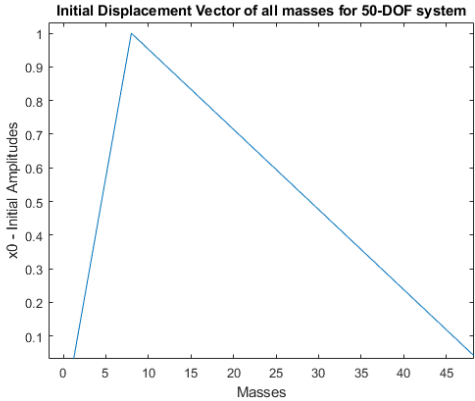
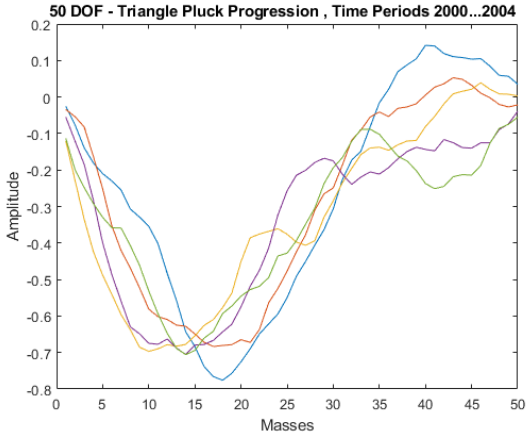
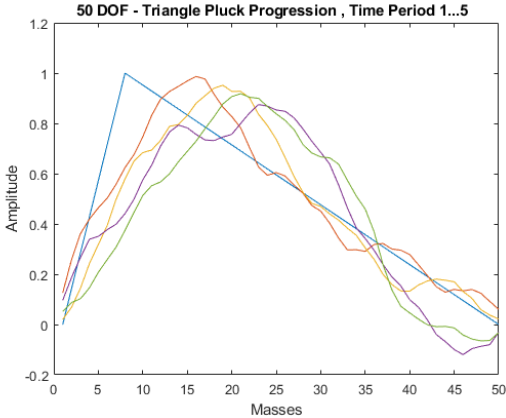
Experiment Set	PM_EX03
Aim	<p>Develop functions for N-DOF matrices definitions :</p> $[M] = \begin{bmatrix} m_1 & 0 & 0 & 0 & \dots & 0 \\ 0 & m_2 & 0 & 0 & \dots & 0 \\ 0 & 0 & m_3 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & \dots & 0 \\ \dots & \dots & \dots & \dots & m_{n-1} & 0 \\ 0 & 0 & 0 & 0 & 0 & m_n \end{bmatrix}$ $[K] = \begin{bmatrix} k_1 + k_2 & -k_2 & 0 & \dots & 0 & 0 \\ -k_2 & k_2 + k_3 & -k_3 & \dots & 0 & 0 \\ 0 & -k_3 & k_3 + k_4 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & 0 \\ 0 & 0 & \dots & \dots & k_{n-1} + k_n & -k_n \\ 0 & 0 & 0 & \dots & -k_n & k_n + k_{n+1} \end{bmatrix}$ <p>With the matrices created programmatically, the usage of the displacement function created in Experiment 1 is extended to work for any number of masses and springs.</p>
JIRA Task	JIRA Link : PM-38 Experiment 3 - MDOF Initial Tests
MATLAB Changes	<p>Create a new function</p> <p>pm_mdof_create_kmatrix_linear.m</p> <p>Extra function:</p>

	pm_mdof_triangle_pluck_condition.m 5 test cases added
Overall Summary	The K matrix is calculating the matrices as expected. Extra function developed in this experiment set for a custom initial displacement condition in the shape of triangle (simulating a string pluck). Both functions are validated.

Experiment ID	PM_EX03_01
Test	Verify construction of K-matrix (stiffness) for constant and different values for each spring using a 4-DOF system.
Input parameters	Test output of K-matrix for the following spring values: k_vector_01 = [1,1,1,1,1]; k_vector_02 = [1,2,3,4,5];
Other Parameters	None
MATLAB Changes	Create test function pm_mdof_matrices_exp03_test01.m For complete definition see REPORT_SUBMISSION\Experiments\pm_exp03
RESULTS	
K Matrix for input: 1 1 1 1 1 Result: 2 -1 0 0 -1 2 -1 0 0 -1 2 -1 0 0 -1 2 K Matrix for input: 1 2 3 4 5 Result: 3 -2 0 0 -2 5 -3 0 0 -3 7 -4	
Conclusion	The matrices calculated by the function are correct for the input spring stiffness values. Therefore, it can be assumed the function performs correctly for an N-DOF system of any stiffness values.

Experiment ID	PM_EX03_02
Test	Examine the displacement/time response of masses ($m1$, $m10$, $m20$) along a 20-DOF system.
Input parameters	Initial displacement of 1 for only first mass $x0(1)=1$
Other Parameters	$M=1$, $K=1$
MATLAB Changes	Create test function <code>pm_mdof_matrices_exp03_test02.m</code> For complete definition see <code>REPORT_SUBMISSION\Experiments\pm_exp03</code>
RESULTS	
	
Conclusion	<p>The displacement of mass 1 takes 20 seconds to propagate through the system to mass 20. There appears to be a period of sine-like motion in the masses measured such as seen in mass 20 from $t=20$ until $t=60$. The low speed of propagation is due to the low natural frequencies of this system.</p> <p>Additionally, as per experiment 2, non-modal initial conditions result in an unstable system. This is to be expected from a mass-spring system where an eigenvector is not used as the initial displacement (or velocity) vector.</p> <p>Finally, the initial displacement of only 1 mass in the system, appears to emphasise higher harmonics, as opposed to the fundamental and low harmonics.</p>
Notes	Examine the effects of different initial displacement conditions on the frequency response of a mass in Experiment 4.

Experiment ID	PM_EX03_03																								
Test	Verify occurrence of dispersion in a 50-DOF system.																								
Input parameters	50-DOF, m = 0.00001, k = 10000																								
Other Parameters	None.																								
MATLAB Changes	Create test function pm_mdof_dispersion_exp03_test03.m For complete definition see REPORT_SUBMISSION\Experiments\pm_exp03																								
RESULTS																									
<div><p>Dispersion, 50 MDOF m=0.000010 , k=10000</p><table border="1"><caption>Approximate data points from the dispersion plot</caption><thead><tr><th>Mode Index</th><th>Freq Hertz</th></tr></thead><tbody><tr><td>0</td><td>0</td></tr><tr><td>5</td><td>1500</td></tr><tr><td>10</td><td>3000</td></tr><tr><td>15</td><td>4500</td></tr><tr><td>20</td><td>6000</td></tr><tr><td>25</td><td>7500</td></tr><tr><td>30</td><td>8500</td></tr><tr><td>35</td><td>9200</td></tr><tr><td>40</td><td>9700</td></tr><tr><td>45</td><td>9900</td></tr><tr><td>50</td><td>10000</td></tr></tbody></table></div>		Mode Index	Freq Hertz	0	0	5	1500	10	3000	15	4500	20	6000	25	7500	30	8500	35	9200	40	9700	45	9900	50	10000
Mode Index	Freq Hertz																								
0	0																								
5	1500																								
10	3000																								
15	4500																								
20	6000																								
25	7500																								
30	8500																								
35	9200																								
40	9700																								
45	9900																								
50	10000																								
Conclusion	This plot shows the frequency of each eigenvalue, corresponding to each mode for this system. Initially, there is a linear harmonic series, but this convergences towards approximately 10 kHz for higher modes, indicating dispersion.																								
Notes																									

Experiment ID	PM_EX03_04
Test	Create “pluck” function to approximate more realistic initial conditions.
Input parameters	<div><pre>pluckPos = 8; pluckAmp = 1; x0 = pm_mdof_triangle_pluck_condition(dof,pluckPos,pluckAmp);</pre></div> <div></div>
Other Parameters	<pre>dof = 50; massSize = 0.00001; kSize = 10000; v0 = zeros(1,dof)';</pre>
MATLAB Changes	Create functions <code>pm_mdof_matrices_exp03_test04.m</code> <code>pm_mdof_triangle_pluck_condition.m</code>
RESULTS	
<div></div>	
Conclusion	For the first five time-steps, the waves propagate through the system based on its initial displacements. This continues after two thousand time-steps, showing a sinusoid-like shape typical of a plucked string. In this case, the pluck is approximated as a triangle-shaped displacement vector. It is important to note in a real string, creating an infinitely small point would add additional harmonics. A more realistic pluck may have a more curved shape around the highest displacement point.

Notes	While this system creates a longitudinal wave, the general behaviour of the system is still valid for transverse wave behaviour for equivalent initial conditions.
-------	--

Experiment ID	PM_EX03_05
Test	Analyse effect of pluck position on frequency response.
Input parameters	pluckPos = 8, 16 ,32; pluckAmp = 1; x0 = pm_mdof_triangle_pluck_condition(dof,pluckPos,pluckAmp);
Other Parameters	dof = 50; massSize = 1; kSize = 1; Mass to plot: 25
MATLAB Changes	Create function pm_mdof_matrices_exp03_test05.m For complete definition see REPORT_SUBMISSION\Experiments\pm_exp03

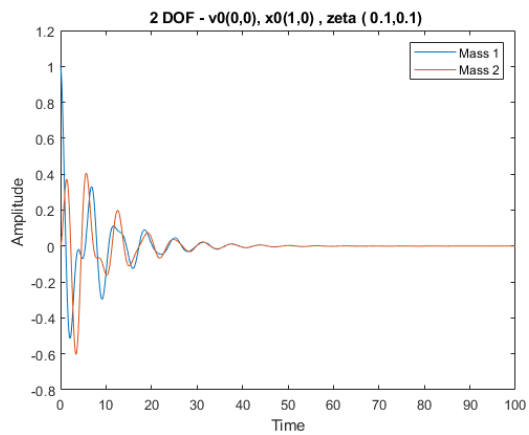
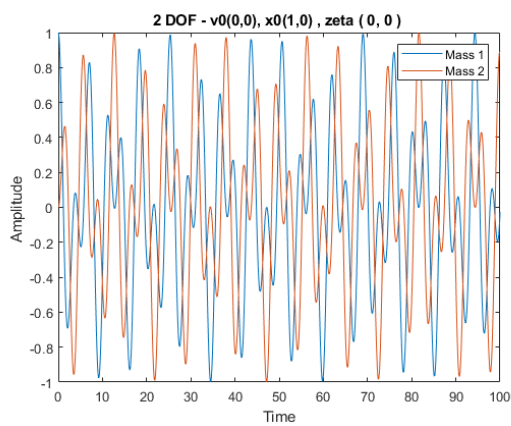
RESULTS	
<div><div><p>50 DOF - Triangle pluck position=8 with FFT</p><p>This plot shows the magnitude spectrum for a pluck position of 8. The x-axis is Frequency (Hz) from 0 to 0.5, and the y-axis is Power (dB) from 0 to 100. The spectrum shows a series of peaks, with the highest peak at approximately 0.05 Hz.</p></div><div><p>50 DOF - Triangle pluck position=16 with FFT</p><p>This plot shows the magnitude spectrum for a pluck position of 16. The x-axis is Frequency (Hz) from 0 to 0.5, and the y-axis is Power (dB) from 0 to 100. The spectrum shows a series of peaks, with the highest peak at approximately 0.1 Hz.</p></div><div><p>50 DOF - Triangle pluck position=32 with FFT</p><p>This plot shows the magnitude spectrum for a pluck position of 32. The x-axis is Frequency (Hz) from 0 to 0.5, and the y-axis is Power (dB) from 0 to 100. The spectrum shows a series of peaks, with the highest peak at approximately 0.2 Hz.</p></div></div>	
Conclusion	For a pluck position closer to the centre of the system, there is less higher harmonics compared to when plucked closer to one end of the system. If you consider the pluck a superposition of modes, the

	pluck closer to one end will require greater contribution from higher harmonics compared to a middle pluck.
Notes	

A.4 EXPERIMENT SET 4

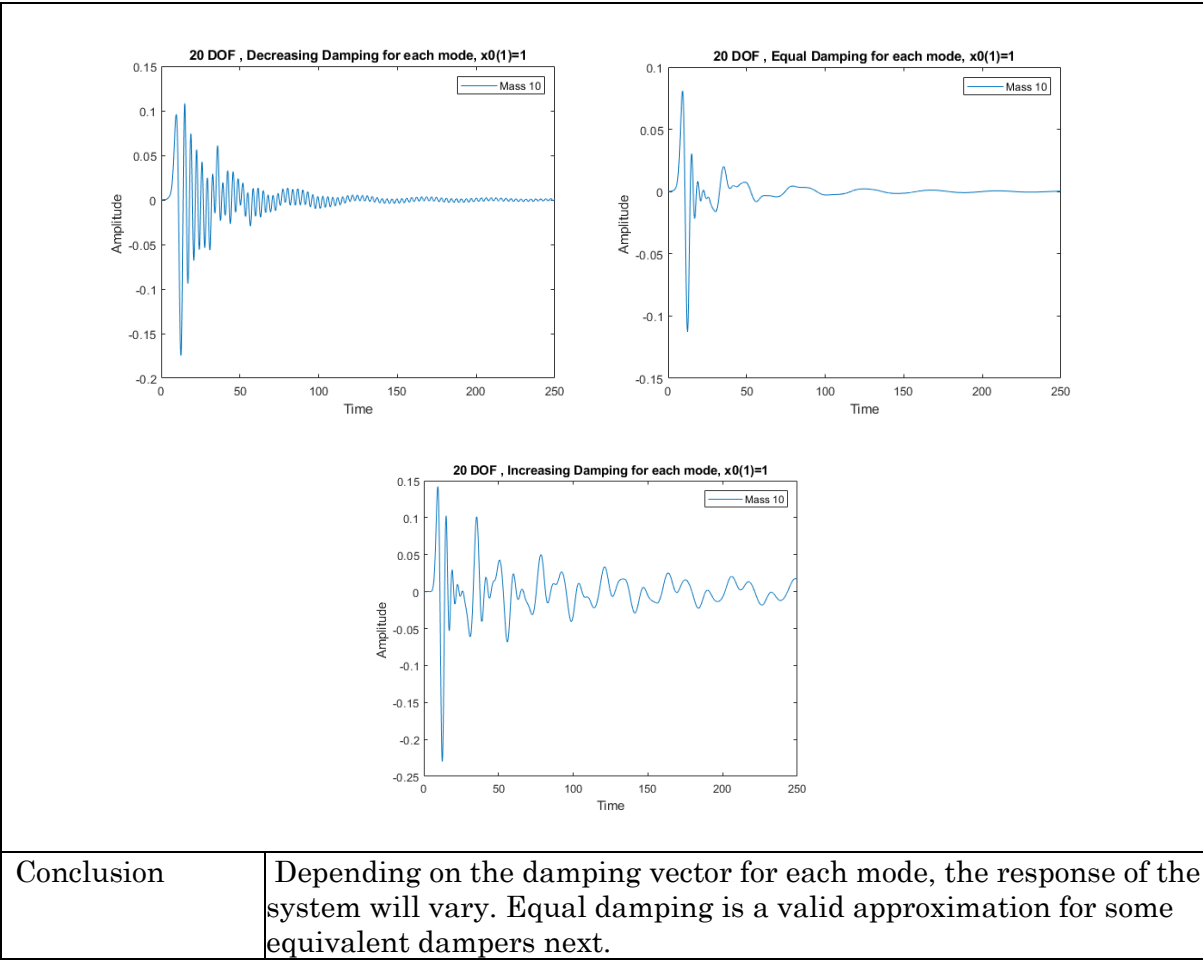
Experiment Set	PM_EX04
Aim	<p>Develop a MATLAB function for the following equation of motion. <i>Free Vibration Proportionally Damped Modal Equation</i></p> $x(t) = \sum_{i=0}^n e^{-\zeta \omega_i t} \{\psi\}_i \left(\frac{\{\psi\}_i^T [M] x(0) \cos \omega_i t}{\sqrt{1 - \zeta^2}} + \frac{\{\psi\}_i^T [M] \dot{x}(0) \sin \omega_i t}{\omega_i \sqrt{1 - \zeta^2}} \right)$
JIRA Task	JIRA Link: PM-41 Damped MDOF System
MATLAB Changes	<p>Create a new function <code>pm_mdof_free_damped_vibration.m</code></p> <p>Additional function is support of apply the modal damping coefficients as a vector <code>pm_mdof_create_zeta_vector.m</code> Zeta vector will describe the damping coefficient for each mode. N.B. any zeta value must be below 1 as system is only based upon underdamping equations. 4 Test cases added.</p>
Overall Summary	<p>Modal damping function successfully created and validated. Supporting function also added as initial conditions for zetas.</p>

Experiment ID	PM_EX04_01
Test	Decay response in a 2-DOF system.
Input parameters	$x0 = [1, 0]'$; $v0 = [0, 0]'$;
Other Parameters	
MATLAB Changes	<p>Create function <code>pm_mdof_free_vibration_ex04_test01.m</code> For complete definition see REPORT_SUBMISSION\Experiments\pm_exp04</p>

RESULTS

Conclusion	System behaves correctly with no damping, and when damping is added for all modes, there is decay.
Notes	Implementation of decay is arbitrary, so does not directly correspond to model with dampers added for every spring connection.

Experiment ID	PM_EX04_02
Test	Decay response in a 20-DOF system with novel damping characteristics.
Input parameters	$x0 = [1, 0]'$; $v0 = [0, 0]'$;
Other Parameters	For 'Normal' case, damping ratio is same for all modes. For 'Increasing' case, damping ratio increases for higher modes. For 'Decreasing' case, damping ratio decreases for higher modes.
MATLAB Changes	Create functions <code>pm_mdof_free_vibration_ex04_test02.m</code> <code>pm_mdof_create_zeta_vector.m</code> For complete definition see REPORT_SUBMISSION\Experiments\pm_exp04
RESULTS	



Appendix B: MATLAB FUNCTIONS

B.1 PM_MDOF_FREE_VIBRATION.M

```
% M - Mass Matrix , K - Stiffness Matrix
% x0 (displacement) and v0 ( velocity ) as column vectors for the masses
% time span 't:interval:extent' over which to the calculate displacements
% Based on the initial conditions
% This are NOT confined to a fixed-fixed system
% The Katrix could be representative of a fixed-fixed or fixed-open etc
% boundary condition

% amplitudes returned as N rows * length(t) - 1 row for each mass
function [amplitudes, omegas, naturals, eigenVectors, eigenValues] =
pm_mdof_free_vibration(M,K,x0,v0,t)
[dofs,cols]=size(M);
amplitudes= zeros(dofs,length(t));
[eigenVectors,eigVals]=eig(K,M);
% Each eigenVector returned is a column
% the eigVals are returned as a diagonal, change to a vector
eigenValues = diag(eigVals);

omegas=sqrt(eigenValues);
naturals = omegas/(2*pi); % frequency in Hertz

% normalize the eigen vectors to the mass matrix
for dof=1:dofs
    norm=sqrt(eigenVectors(:,dof)'*M*eigenVectors(:,dof));
    eigenVectors(:,dof)=eigenVectors(:,dof)/norm;
end
for dof=1:dofs
    eigenVector = eigenVectors(:,dof);
    eigenArray = eigenVector';
    omega = omegas(dof);
    % from the modal wave equation
    % xt = Sum i = 1..n : EV(i)* ( EVT(i) * Mx0 * Cos w(i)t
    % + ( EVT(i) * Mv0 * Sin w(i)t / w(i) ) )
    ampT= eigenVector*(eigenArray * M* x0*cos(omega.*t)+ eigenArray*M*v0/omega
    *sin(omega.*t));
    % add the parts to form the total ( superposition of modes )
    amplitudes=amplitudes+amp
end
```

B.2 PM_MDOF_FREE_VIBRATION_MODIFIED.M

```
% M - Mass Matrix , K - Stiffness Matrix
% x0 (displacement) and v0 ( velocity ) as column vectors for the masses
% time span 't:interval:extent' over which to the calculate displacements
% Based on the initial conditions
% This are NOT confined to a fixed-fixed system
% The Katrix could be representative of a fixed-fixed or fixed-open etc
```

```

% boundary condition

% amplitudes returned as N rows * length(t) - 1 row for each mass

% THIS is a modified function for to support experiments
function [amplitudes, omegas, naturals, eigenVectors, eigenValues, massNParts] =
pm_mdof_free_vibration_modified(M,K,x0,v0,t,n)
[dofs,cols]=size(M);
massNParts = zeros(dofs,length(t));
amplitudes= zeros(dofs,length(t));
[eigenVectors,eigVals]=eig(K,M);
% Each eigenVector returned is a column
% the eigVals are returned as a diagonal, change to a vector
eigenValues = diag(eigVals);

omegas=sqrt(eigenValues);
naturals = omegas/(2*pi); % frequency in Hertz

% normalize the eigen vectors to the mass matrix
for dof=1:dofs
    norm=sqrt(eigenVectors(:,dof)'*M*eigenVectors(:,dof));
    eigenVectors(:,dof)=eigenVectors(:,dof)/norm;
end
for dof=1:dofs
    eigenVector = eigenVectors(:,dof);
    eigenArray = eigenVector';
    omega = omegas(dof);
    % xt = Sum i = 1..n : EV(i)* ( EVT(i) * Mx0 * Cos w(i)t
    % + ( EVT(i) * Mv0 * Sin w(i)t / w(i) ) )
    ampT= eigenVector*(eigenArray * M* x0*cos(omega.*t)+ eigenArray*M*v0/omega
* sin(omega.*t));
    massNParts(dof,:) = ampT(1,:);
    % add the parts to form the total ( superposition of modes )
    amplitudes=amplitudes+ampT;
end

```

B.3 PM_MDOF_CREATE_KMATRIX_LINEAR.M

```

% From the general form :
% ( [M] * v0 ) + ( [K] * v0 ) = 0 ( free vibration - no damping )
% derive the stiffness KMatrix for a linear M-Dof system FIXED at both ends
% FIX:k1-m1-k2-m2-k3-m3-k4:FIX
% Input is an array (row ) of stiffness K's
% the array size has dof + 1 elements
% Example 3 -dof
% [ k1+k2    -k2      0
%    -k2     k2+k3   -k3
%      0     -k3     k3 +4 ]
% pm_mdof_create_kmatrix_linear( [1,1,1,1] );
%Returns
% [ 2    -1     0
%   -1     2    -1
%    0    -1     2 ]

```

```

function [KMatrix] = pm_mdof_create_kmatrix_linear( kVector)

    dof = length( kVector) -1 ;
    % create a dof length vector of values k1+k2, k2+k3, k3+k4
    addedKs= zeros(dof,1);
    for ( i = 1:dof )
        addedKs(i) = kVector (i) + kVector(i+1);
    end
    % make a diagonal out of the k1+k2, k2+k3, k3+k4
    KMatrix = diag(addedKs);
    % make vector of negative k's starting at position 2 ; -k2, -k3,-k4 etc
    negativeKs = zeros(dof-1,1);
    for ( i = 2:dof)
        negativeKs(i-1) = -kVector (i);
    end
    % now apply these diagonally either size of the k matrix
    for ( i = 1:dof -1 )
        KMatrix( i, i+1 ) = negativeKs(i);
        KMatrix( i+1, i ) = negativeKs(i);
    end
    if ( issymmetric( KMatrix ) == 0 )
        error('**** Error. \nCalculated [K] Matrix is incorrect' )
    end
end
end

```

B.4 M_MDof_FREE_DAMPED_VIBRATION.M

```

% M - Mass Matrix , K - Stiffness Matrix
% x0 (displacement) and v0 ( velocity ) as column vectors for the masses
% time span 't:interval:extent' over which to the calculate displacements
% Based on the initial conditions
% This are NOT confined to a fixed-fixed system
% The Katrix could be representative of a fixed-fixed or fixed-open etc
% boundary condition
% zetas is a vector of modal damping ratios( ie damping applied to each mode
% zetas must be < 1 , as equation is based on Underdamping
%% amplitudes returned as N rows * length(t) - 1 row for each mass
function [amplitudes, omegas, naturals, eigenVectors, eigenValues] =
pm_mdof_free_damped_vibration(M,K,zetas,x0,v0,t)
if ( max(max(zetas)) >= 1 || min(min(zetas)) < 0 )
    error ('Values for zeta damping must be 0 <= zeta < 1 for an underdamped system
' );
end

[dofs,cols]=size(M);
xSz = length( x0);
vSz = length( v0);
zSz = length( zetas);
[kdofs,kcols]=size(K);

if ( xSz ~= dofs || vSz ~= dofs || zSz ~= dofs || kdofs ~= dofs || kcols ~= dofs ||
cols ~= dofs)
    error ('Values of the input paramters are not for the same DOF dimensions' );
end

```

```

amplitudes= zeros(dofs,length(t));
[eigenVectors,eigVals]=eig(K,M);
% Each eigenVector returned is a column
% the eigVals are returned as a diagonal, change to a vector
eigenValues = diag(eigVals);

omegas=sqrt(eigenValues);
naturals = omegas/(2*pi); % frequency in Hertz

% normalize the eigen vectors to the mass matrix
for dof=1:dofs
    norm=sqrt(eigenVectors(:,dof)'*M*eigenVectors(:,dof));
    eigenVectors(:,dof)=eigenVectors(:,dof)/norm;
end

dampedOmegas = omegas.*sqrt(1-zetas.^2);

for dof=1:dofs
    zeta = zetas(dof);
    eigenVector = eigenVectors(:,dof);
    eigenArray = eigenVector';
    omega = omegas(dof);
    dampedOmega = dampedOmegas(dof);
    ampT= eigenVector*(eigenArray * M * x0*cos(omega.*t)/sqrt(1-zeta^2) +
eigenArray*M*v0/omega *sin(omega.*t)/dampedOmega);
    ampT = ampT.*exp(-zeta*omega.*t);
    % add the parts to form the total ( superposition of modes )
    amplitudes=amplitudes+ampT;
end

```

B.5 PM_MDOF_CREATE_ZETA_VECTOR.M

```

function [zetaVector] = pm_mdof_create_zeta_vector(zetas,type,dofs)
    switch type
        case 'normal' %damping coefficient is same for all modes
            zetaVector = zetas*ones(dofs,1)
        case 'increase' % damping coefficient increases for each mode
            zetaVector = zeros(dofs,1)
            for i=1:dofs
                zetaVector(i,1)=i*zetas/dofs
            end
        case 'decrease' % damping coefficient decreases for each mode
            zetaVector = zeros(dofs,1)
            for i=1:dofs
                zetaVector(dofs-i+1,1)=i*zetas/dofs
            end
    end
end

```

B.6 PM_MDOF_TRIANGLE_PLUCK_CONDITION.M

```
% create a triangular pluck displacement x0 vector
% at position with amplitude x
% returns an column vector dof x1 elements

function [ XVector] = pm_mdof_triangle_pluck_condition( dof, position, x)

try
    if ( position <1 || position > dof )
        error('Apex Pluck position is out of bounds %d greater than dof %d',position, dof);
    end
    %if position is not an integer
    if ( mod(position,1) ~= 0)
        error('Apex Pluck position is not a valid mass position');
    end
    % positions start at 1 for mass 1
    if(position == 1)
        left_gradient = x;
    else
        left_gradient = x/ (position -1);
    end
    right_gradient = x/ ( position -dof );

    for (i=0:dof-1)
        if (i < position)
            if (position == 1)
                x0(i+1) = left_gradient;
            else
                x0(i+1) = i * left_gradient;
            end
        else
            x0(i+1)= -1 * ( (dof-i- 1) * right_gradient );
        end
    end
    XVector =x0';
catch e
    error('pm_mdof_triangle_pluck_condition: %s',e.message);
end

end
```

B.7 PM_CREATE_INITIAL_CONDITION_VECTOR.M

```
% function to return initial conditions vectors
% all or some them can be applied to displacements, velocities
% masses, stiffness and damping modal coefficients
% example, N,0,0,0,N would not be applicable to masses or stiffness
% arg4 is for the pluck position
function [ vector] = pm_create_initial_condition_vector( dof, modeStr, arg3, pluckPosition)

    % remove any white spaces '1,2 ,3, 4,5 '
    % becomes '1,2,3,4,5'
    % 'Triangle Pluck' becomes 'TrianglePluck'
```

```

mode = modeStr(~isspace(modeStr));
% INITIAL CONDITIONS
% the ' will transpose 1 X N matrix to N x 1 ( Vector )
vector = zeros(1,dof)';

switch mode
case 'LinearIncrease' % increases for each mode up to arg3
    for i=1:dof
        vector(i,1)=i*arg3/dof;
    end
    return;
case 'LinearDecrease' % decreases for each mode starting at arg3
    for i=1:dof
        vector(dof-i+1,1)=i*arg3/dof;
    end
    return;
case 'Linear'
    vector = ones(1,dof)';
case 'None'
    vector = zeros(1,dof)';
case 'Triangle'
    vector = pm_mdof_triangle_pluck_condition( dof, pluckPosition, arg3);
    % already scaled so just return
    return;
case 'SoftTriangle'
    vector = pm_mdof_triangle_pluck_condition( dof,pluckPosition, arg3);
    vector = lowpass(vector,0.2);
    % already scaled so just return
    return;
case 'Bell'
    temp = zeros(1,dof);
    scale = 1000;
    step = 1/ scale;
    x = [-3:step:3];
    y = normpdf(x,0,1);
    r = length(y)/dof;
    for ( i= 1:dof )
        pos = round ( i * r );
        temp( i) = y (pos);
    end
    vector = temp';

case 'Step'
    direction = 1;
    cnt=1;
    for ( i = 1 : dof )
        vector(i) = direction ;
        if ( mod(cnt,5) == 0 )
            direction = direction * -1;
        end
        cnt = cnt +1;
    end

case 'N_2N_4N_8N'
    marray = 1 * 2.^(0:dof-1);
    vector = marray';

```

```

case 'N_2N_3N_4N'
    vector = (1:dof)';

case 'N_2N_N_2N'
    temp = ones(1,dof);
    for ( i=1:dof)
        if (mod(i,2) == 0 )
            temp(i) = 2;
        end
    end
    vector = temp';
case 'N_N_N_2N_2N_2N_N_N_N'
    temp = ones(1,dof);
    for ( i=1:dof)
        a = ceil( i/3);
        if (mod(a,2) == 0 )
            temp(i) = 2;
        end
    end
    vector = temp';

% These are more applicable for MASSES ie mass values
% should not be ZERO or NEGATIVE
case 'N,0,0,0,-N'
    vector(1) = -1;
    vector(dof) = +1;
case 'N,0,0,0,N'
    vector(1) = +1;
    vector(dof) = +1;
case '0,0,N,0,0'
    % 0, 0, 0, 1 , 0, 0 , 0
    if ( mod ( dof, 2 ) == 0 )
        error('**** Error. \nThere is no middle mass of a System with Dof: %d.',
dof)
    end
    vector( ceil( dof/2 ) ) = 1;
case '0,0,-N,N,0,0'
    if ( mod ( dof, 2 ) == 1 )
        error('**** Error. \nThere is no middle mass pair of a System with Dof:
%d.', dof)
    end
    vector( dof/2 ) = -1;
    vector( (dof/2) + 1 ) = 1;
case '0,N,-N,N,0'
    if ( mod ( dof, 2 ) == 0 )
        error('**** Error. \nThere is no middle mass set of a System with Dof:
%d.', dof)
    end
    vector( ceil( dof/2 ) -1 ) = +1;
    vector( ceil( dof/2 ) ) = -1;
    vector( ceil( dof/2 ) +1 ) = +1;

case 'N,-N,N,-N,N'
    direction = 1;
    for ( i = 1 : dof )
        vector( i ) = direction ;
        direction = direction * -1;
    end

```

```

        otherwise
            disp(['Applying CSV for input : ' mode]);
            % assume comma seperated list if not already processed
            vector = pm_mdof_csv_vector (dof, modeStr );

    end
    % scale the vector
    vector = arg3 * vector;

end

```

B.8 PM_MDOF_FORMAT_FFT.M

```

% FFT and calculations for Power Spectrum
function [fftplot,hertz] = pm_mdof_format_fft(soundAmp,t,fs)

try
    N=length(t);
    X_mags = abs(fft( soundAmp ));
    bin_vals = 0 : N-1;
    fax_Hz = bin_vals*fs/N;
    N_2 = ceil(N/2);
    fftplot = 20*log10(X_mags(1:N_2));
    hertz = fax_Hz(1:N_2);
catch e
    error('pm_mdof_format_fft: %s',e.message);
end
end

```

B.9 PM_MDOF_TIME_VARIANT_PICKUP.M

```

%TIME_VARIANT_PICKUP Introduces Time-Variant Pickup for Outputs
% The pickup can sweep to and forth along along mass chain
% If sweep is 1, it will pickup up proportionally from M1,M2,..Mn and Stop.
% If sweep is 2 , it will pickup up proportionally from M1,M2,..Mn and then
% back to ..M2,M1
% For a 4 DOF System - 1,2,3,4
% sweep 1 = 1,2,3,4      = 4 slices
% sweep 2 = 1,2,3,4,3,2,1 = 7 slices
% sweep 3 = 1,2,3,4,3,2,1,2,3,4 = 10 slices
% slices = ( sweeps * ( dof -1 )  + 1
% when the slices count has been determined, this results in a slice-size
% for the time element of the amplitude matrix
function [soundMoves, pickupArray] = pm_mdof_time_variant_pickup(amplitudes, sweeps,
mode )
try
    if ( strcmp(mode,'Amplitude Input')==0 && strcmp(mode,'Moving Linear') ==0 &&
strcmp(mode,'Moving Linear Reset') == 0 )
        error ('Invalid mode %s', mode );
    end
    [dof,timeSlots]=size(amplitudes);
    soundMoves = zeros(1,timeSlots);

    if strcmp(mode,'Amplitude Input')

```



```

    if ( sweeps <1 || sweeps > dof )
        error('mass %d for Amplitude Input is out of bounds ', sweeps);
    end
    pickupArray = amplitudes(sweeps,:);
    pickupArray = pm_mdof_normalise_audio(pickupArray);
    pickupArray=pickupArray+1;
    pickupArray=pickupArray*((dof-1)/2);
    pickupArray=pickupArray+1;
    pickupArray=round(pickupArray);
    for i=1:timeSlots
        soundMoves(i) = amplitudes(pickupArray(i),i);
    end
    return
end

slices = ( sweeps * ( dof -1 ) ) + 1;
sliceVal = timeSlots/slices;
slice = round(sliceVal);
maxSweeps = floor( ( timeSlots -1 )/ ( dof -1 ) );
if ( sliceVal < 1 )
    error('Maximum Sweep count is %d', maxSweeps);
end
direction = 1;
pos = 0;
for i=1:slice:timeSlots
    pos = pos + direction;
    m = mod ( pos, dof);
    if ( m == 0 )
        if ( strcmp(mode,'Moving Linear') )
            direction = direction * -1;
            if ( pos == 0 && direction == 1)
                pos = 2;
            end
        end
    end
    endExtract = (i + slice - 1);
    if ( endExtract > timeSlots )
        endExtract = timeSlots;
    end
    extract = amplitudes(pos,i:endExtract);
    for ( x= 1:length(extract) )
        pickupArray(i+x-1) = pos;
        soundMoves(i+x-1) = extract(x);
    end
    if ( strcmp(mode,'Moving Linear Reset') && m == 0 )
        pos = 0;
    end
end
catch e
    str = sprintf('pm_mdof_time_variant_pickup : %s',e.message);
    %disp(str);
    error ( str);
end

end

```

B.10PM_ANALYSE_AUDIO.M

```
% Physical Model , analyse audio utility function
% Generates FFT and Spectrogram for any audio file for report

[location,path] = uigetfile('*.wav');
audpth = strcat( path,location);
info = audioinfo(audpth);
[audio] = audioread(audpth);
audio = audio';
t=1:length(audio);
fs=info.SampleRate;
subplot(2,1,1)
[fft, hz]= pm_mdof_format_fft(audio(1,:),t,fs);
semilogx( hz, fft);
title('FFT');
xlabel('Frequency (Hz)');
ylabel('Amplitude (db)');
subplot(2,1,2) % Second Plot is Spectrogram of centre mass
%spectral analysis
spectrogram( audio(1,:)','128,120,128, fs); %spectral analysis
title('Spectrogram');
axis tight
```

Appendix C: **FORCED** EXCITATION

```

pm_mdof_forced_vibration_test01.m
dof = 10;
kVector=ones( 1, dof + 1 ) * 1000;
zVector=zeros( 1, dof + 1 );
mVector=ones( 1, dof );

[MMatrix] = pm_mdof_create_Mmatrix_linear( mVector);
[KMatrix] = pm_mdof_create_Kmatrix_linear( kVector);
% use the KMatrix function for the damping vector
% as same shape and formats apply as per stiffness
[ZMatrix] = pm_mdof_create_Kmatrix_linear( zVector);
% no initial velocity of displacement conditions
x0Vector = zeros(1, dof );
v0Vector = zeros(1, dof );

sampleRate = 1000; % 44100;
interval = 1/sampleRate;
extent = 2; % 2 secs
tv = 0:interval:extent;
zero_force_array = 0* tv;

% the forces for all dof's are zero
for d= 1:dof
    all_forces(d,:)=zero_force_array;
end

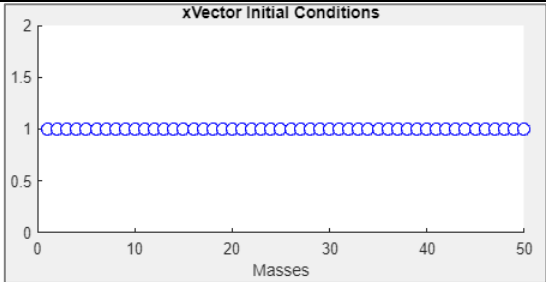
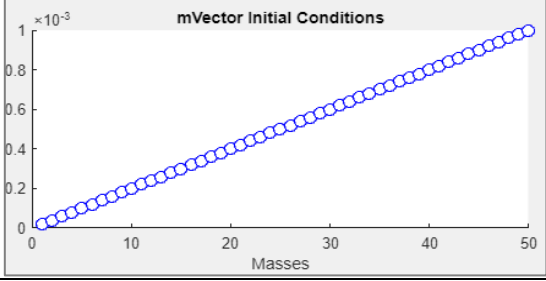
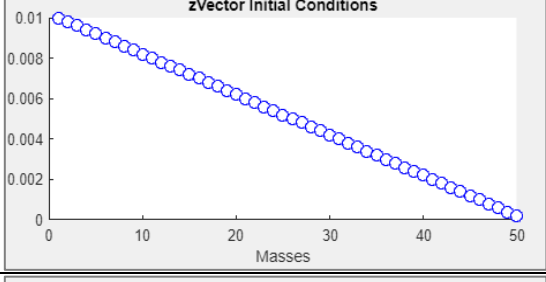
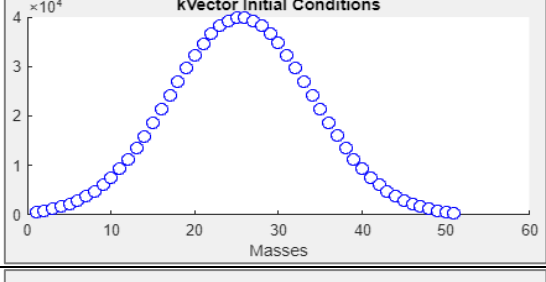
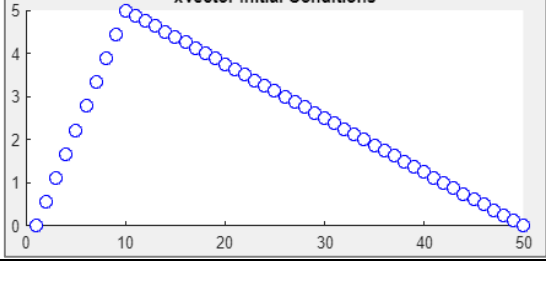
forceDuration = 50;
forceAmplitude = 1;
forceFreq = 100 ; % angular freq
forcedAmpl = 4;
applyforce = zero_force_array;
% create a cosine excitation over a short period
% this could be any forced profile ( ie hammer blow )
for i = 1:forceDuration
    applyforce(i)= forceAmplitude*cos(forceFreq* i);
end
% apply force to the 5th mass -
all_forces(5,:) = applyforce;
tic
[amplitudes]=pm_mdof_forced_vibration(MMatrix, KMatrix, ZMatrix,
all_forces,x0Vector,v0Vector,tv);
elapsed = toc;
% plot the effect on mass 1
plot(tv,amplitudes(1,:) );
title( 'Mass 1 Response, from Forced Cos(2 *[1 to 50] )burst at Mass 5 ' );
% plot all amplitudes
%plot(tv,amplitudes );
s = sprintf('dof = %d, elapsed %.3f', dof, elapsed );
disp(s);

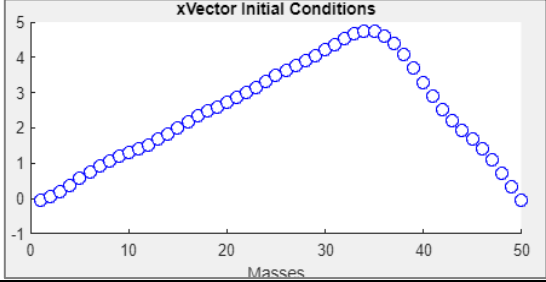
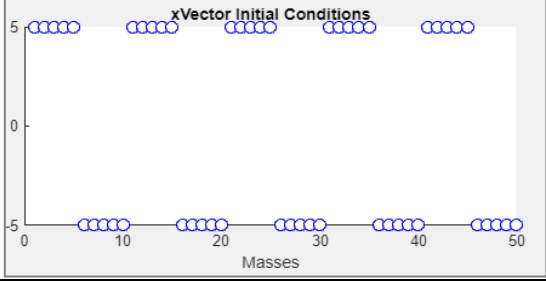
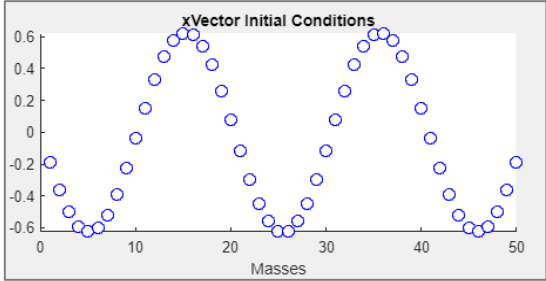
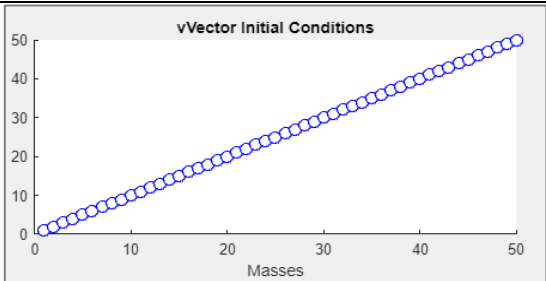
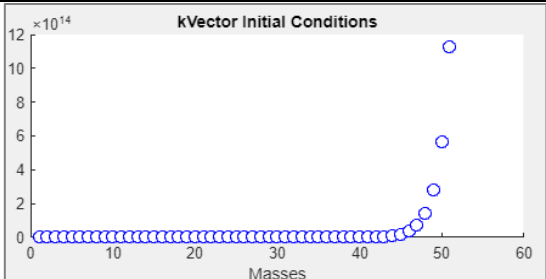
```

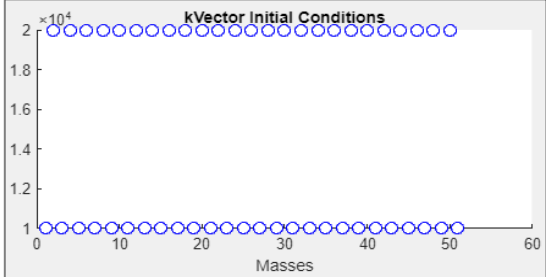
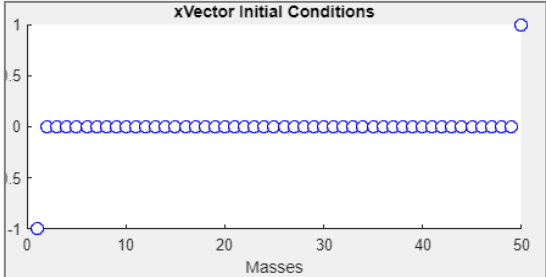
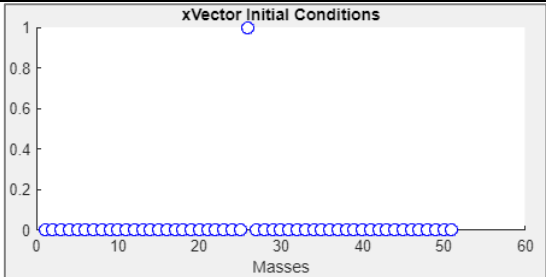
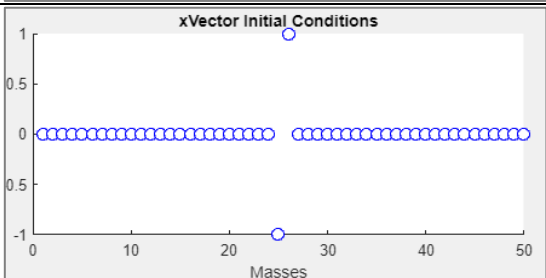
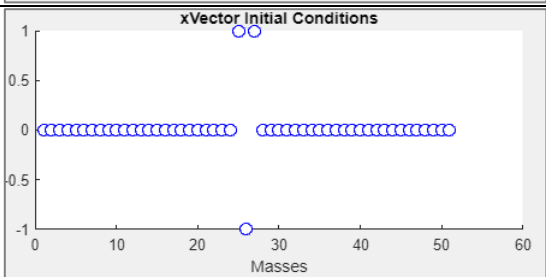
Appendix D: TABLE OF VECTOR PROFILES

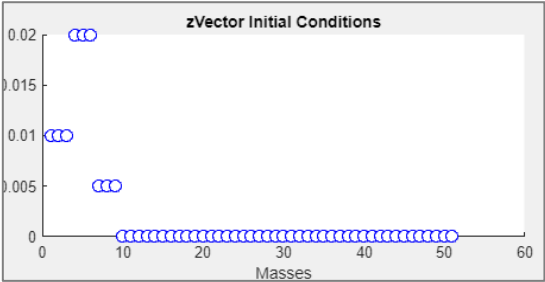
Graphic examples for `pm_mdof_create_initial_condition_vector.m`.

See Appendix B.6 for full MATLAB code.

Profile	Description	Graph	Used By
None	No Profile is applied.		X, V, Z
Linear	Same values applied across elements. Example, applied to X-Vector		X, V, Z, K, M
Linear Increase	Increase proportionally across elements from value/DOF to the value. Example, applied to M-Vector were $m = 0.001$		X, V, Z, K, M
Linear Decrease	Decreases proportionally across elements from the value to value/DOF. Example, applied to Z-Vector were $z = 0.01$		X, V, Z, K, M
Bell	Apply a normal probability density function around the centre mass. Example, applied to spring Stiffness K-Vector		X, V, Z, K, M
Triangle	Apply a triangular shaped distribution in conjunction with apex position. Example, applied to X-Vector, with Apex set to position 5		X, V, Z

Soft Triangle	Apply a triangular shaped distribution with a 'rounded' apex. Example, applied to X-Vector, with Apex set to position 35		X, V
Step	Applies a positive and negative distribution, changing for every 5 th mass. Example, applied to X-Vector		X, V
Xth Eigen	Uses the x-value to apply the Xth eigenvalue as a displacement, Resulting in the natural frequency of Xth normal mode for a mass. Example : The 5 th Modal Shape (X =5) applied as the Initial Displacements X-Vector		X
N_2N_3N_4N	Increments the distribution by factor of 1 for each element. Example, Applied to V-Vector		X, V, K, M
N_2N_4N_8N	Increments the distribution by factor of 2 for each element. Example, applied to K-Vector		X, V, K, M

$N_2N_N_2N$	Increments the distribution by factor of 2, resets and repeats Example, applied to K-Vector		X, V, K, M
$N,0,0,0,-N$	Applies only to the first and last masses in the system, with opposite effects. Example applied to X-Vector:		X, V
$0,0,N,0,0$	Applies only to middle mass of the system. Example, applied to X-Vector with a DOF of 51 (odd number)		X, V
$0,0,-N,N,0,0$	Applies only to middle two mass of the system with opposite effects. Example, applied to X-Vector with a DOF of 50 (even number)		X, V
$0,N,-N,N,0$	Applies only to middle three mass of the system with V-shape effect. Example, applied to X-Vector with a DOF of 51 (odd number)		X, V

<i>Free Edit</i>	<p>Comma separated values can be input for the masses for custom effects. Example, For a 10-DOF system, with $x=0.5$, and X-Vector Profile (Free Edit) = 0, 0.5, 0,-2</p> <p>Will result in the X-Vector [0, 0.25, 0, -1, 0,0,0,0,0,0]</p> <p>Example 1,1,1,2,2,2,0.5,0.5,0.5 applied to Z-Vector $z = 0.01$</p>		X, V, Z
------------------	---	--	---------