University of Southern Queensland

Faculty of Health, Engineering & Sciences

Condition Monitoring of Rotating Machinery - Vibration Analysis

A dissertation submitted by

Richard Amos Little

in fulfilment of the requirements of

ENG4112 Research Project

towards the degree of

Bachelor of Mechanical Engineering

Submitted: October, 2015

Abstract

The aim of condition monitoring is to detect defective machine components and allow timely repair before failure and secondary damage occurs. This project, in particular, covers the condition monitoring techniques applicable to rotating machinery and vibration analysis.

Vibration analysis is a specialised role and equipment owners often have a limited understanding of it. Practicing technicians also benefit from better understanding fundamental principles and ongoing training. Key to improving outcomes of CM programs is improving knowledge and understanding.

Utilising overall project specifications, a schedule, flow and Gantt charts, decision trees, listing activities with diagrams and milestones ensured the orderly development of the project. Researching condition monitoring practices and standards applicable to rotating machinery led to developing required outcomes, software specifications and user requirements for a virtual training package prior to writing the Matlab program. Development of the vibration training program in line with ISO18436-2 requirements and industry recognised practices for personnel conducting and utilising vibration condition monitoring resulted in a program that is easy to use, familiarising the user with fundamental concepts and readily detectable faults.

The key aspect of the project, the development of a training and awareness tool based on industry practices and expectations, is displayed in the training program. It is a tool that can help technicians and plant owners better understand and implement vibration condition monitoring and improve outcomes thus reducing costly unplanned plant outages.

University of Southern Queensland

Faculty of Health, Engineering & Sciences

ENG4111/2 *Research Project*

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Health, Engineering & Sciences, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Health, Engineering & Sciences or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled "Research Project" is to contribute to the overall education within the student's chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Dean

Faculty of Health, Engineering & Sciences

Certification of Dissertation

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.
I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Richard Amos Little
0050085773

Acknowledgments

I would like to thank my supervisor Mr Robert Fulcher for guidance and feedback on work throughout the project.

I would also like to thank work colleagues Mr Phil Carroll for ideas and interest in my project and Mr Alan Fogarty for support, throughout the course of the program, in helping manage my work responsibilities whilst absent from work studying.

Richard Amos Little

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Machines use rotating components to transfer energy and are found in all industrial endeavors. From use in paper mills, power plants, diesel engines in trucks or ships, chemical processing plant to small electric motors and pumps used in space. Moubray (1997) outlines how prior to the the Second World War there was a tendency to fix a machine when it failed. Those machines were typically heavy, over engineered machines. The effort in war time to increase factory output, and the increased demand for raw materials, lead to to machinery being leaned down and improvements in maintenance practice. Meeting the need to produce more machinery from a set amount of steel meant reducing component sizes especially in the urgency of war. This lead to increased component speeds, improved performance and output from machines which in turn required more maintenance. To maintain higher equipment output levels the maintenance strategy became one of preventing failure with work such as rebuilds to stop failures of harder working smaller components. Improved inspection regimes and measurement of product properties from assembly lines indicated wear of production line machinery and the need for repair could then be planned and prepared for. Advances in technologies lead to the development of more sophisticated condition monitoring (CM) techniques either designed for or best suited to specific machine types, machine components or failure modes. There are condition monitoring techniques that can be used while the machine is operating or being productive, techniques that require the machine to be running but out of production mode and inspection techniques for which machinery must be out of service or dismantled. There are so many techniques that choosing the best and cost efficient can be challenging. The goal of condition monitoring is to detect defective components and allow timely repair before failure and secondary damage occurs.

1.2 Project Aim and Methodology

This project covers condition monitoring techniques applicable to rotating machinery and in particular the technique of vibration analysis with the aim of developing a virtual software package for awareness and training purposes. Methodology for this project sets out a process to conduct the project in an objective manner. A primary task is the research of technologies used by the various condition monitoring techniques. From this research,

established standards and practices a grading of techniques as suitable to detect failure modes at different stages of fault development is desired. This process highlights what are common industry expectations of a vibration analyst and is supported by current standards regarding the competencies expected of trained technicians. With a reliable base of information the aim of the project is the development a virtual training package for vibration analysis and awareness. The project defines the requirement outcomes of a training program and incorporates the researched dynamic properties into the program for introduction to the user from the concept of simple harmonic motion to balancing techniques. The virtual package can be easily operated with the use of graphical user interfaces (GUI) allowing adjustment to machine physical properties and a display of the effect these changes have on a machines.

1.3 Project Rationale

Vibration analysis (VA) is a tool capable of detecting changes in a machines condition well before a fault develops to the stage of risk of failure being high. It is a specialized role and many sites are not large enough to employ permanent staff to fulfill this role. As a result most staff of companies owning machinery have had limited exposure to VA. Many companies use specialized external labor to conduct this work and often do not have in house personnel with a solid understanding of the service they buy. This can lead to misunderstandings surrounding the technique, the results of testing and limitations of the technique. In some cases these misunderstandings result in machine failure. Technicians new to the field may also find they receive only basic training before being sent into the field with a data collector/analyzer to diagnose complex and expensive machinery faults. This can cause a great deal of stress for analyst and plant owner and is a recipe for poor outcomes. The science of the technique is well documented and there are formal training options available for an ISO level accreditation.

Chapter 2

Literature Review

2.1 Failure Modes and Defects

Every machine is subject to a number of ways in which it may fail. This can range from wear and tear from heavy duty or deterioration with age. Lubrication failures can include the wrong lubricant, the wrong amount or type of lubrication, excess oil shear, oxidation, additive deterioration and contamination by water, dirt or chemical. Failure may be caused by operator error or equipment misuse, errors from maintenance tasks and fundamental problems with design. These are known as failure modes. Unless a machine has a component with an inherent weakness that is proven to fail at a given point then there is a good chance it can be left in service with minimal maintenance. In fact maintenance often has the effect of introducing human errors of some degree which can be detrimental to machine life. Moubray (1997 p. 12-13) outlines how there are broadly six patterns regarding the "conditional probability of failure against operating age for a variety of electrical and mechanical items, shown in figure 2.1 on page 6. Percentages for the six probability of failure patterns are A 4%, B 2%, C 5%, D 7%, E 14%, F 68%. This study also showed that the more complex the machine is the higher the percentage of patterns E and F are. This is some proof that doing maintenance is a risk to the machines reliability and that there is little or no relationship between how long a machine has been in service and how likely it is to fail. It also shows that once a machine is past the infant mortality phase then failure rate is likely to be relatively constant.

Moubray (1997) classifies failures into three groups

- When capability falls below desired performance.
- When desired performance is greater than initial capability.
- When the machine capability does not meet desired performance from the new.

Once a machine is in service and a fault has been detected three ways of managing the situation are to slow the rate of deterioration by reducing the load on machine, monitor it to detect the onset of failure then repair or remove it from service once the risk of it failing is too high or let it run to failure. Machine failure can lead to product contamination, upset clients or financial penalties for not meeting contract obligations, cause irreparable damage to expensive infrastructure, cessation of operations, kill and risk the safety of personnel, cause environmental and public health disasters such as large oil spills and

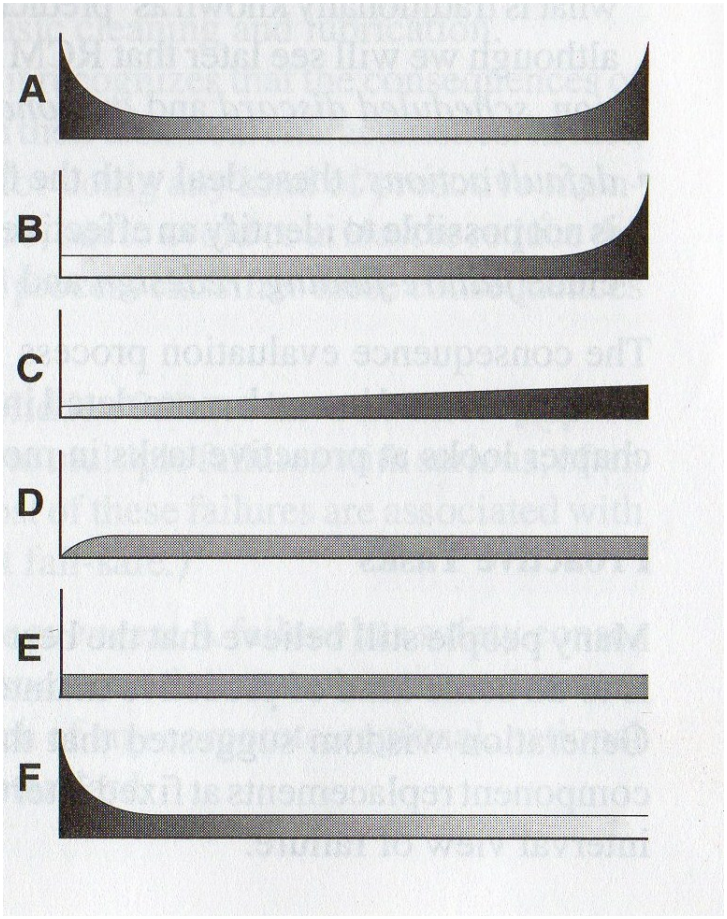


Figure 2.1: Failure patterns and percentages for probability of failure. Pattern A 4%, B 2%, C 5%, D 7%, E 14%, F 68%. (Moubray 1997 p. 12)

nuclear contamination.

Condition based maintenance makes use of the P-F curve to illustrate machine deterioration and the need to capture machine condition in the context of time (figure 2.2 on page 7). The point P is when a potential failure is detected. The point F on the curve is a level of failure. Failure can be determined as completely destructive or the machine may still work but not meet predetermined standard for further operation. The trick is to have a condition monitoring inspection program in place to detect a fault at the point P when the failure is developing with enough time to plan and prepare for the required maintenance, safely before the point F of functional failure is reached. Condition monitoring needs to be conducted at a time interval between inspections no shorter than the P-F period, ideally half of the P-F time.

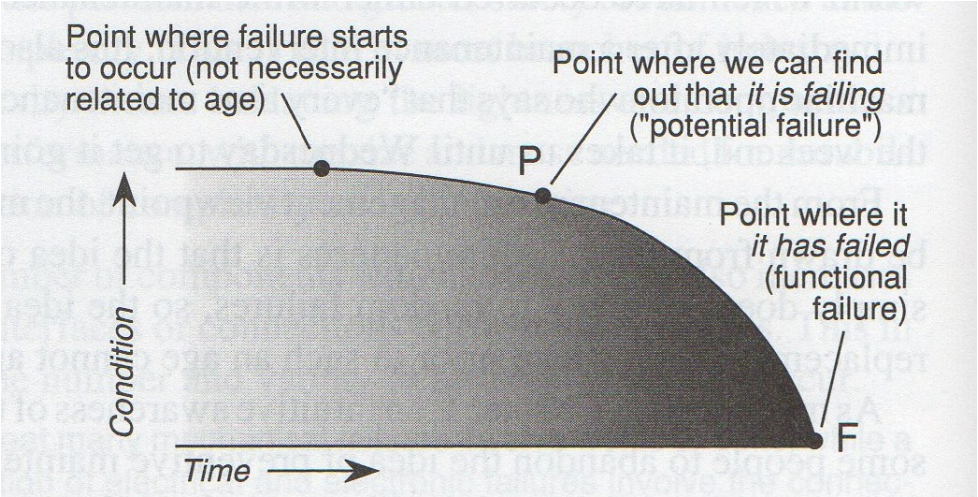


Figure 2.2: The P-F Curve (Moubray 1997 p. 144)

2.2 Condition Monitoring of Rotating Machinery

Understanding the failure modes of the machine to be monitored is essential to applying to best technique for detecting likely failure modes. Equipment to detect failures. Moubray (1997) lists approximately 100 condition monitoring techniques and classifies them broadly as

- Particle effects
- Chemical effects
- Temperature effects
- Physical effects
- Electrical effects
- Vibration effects

2.3 Particle Condition Monitoring

For rotating machinery this is typically the monitoring of lubrication for contamination by detecting foreign particles that inhibit or degrade the function of the lubrication and cause damage to machine components.

2.3.1 Ferrography

This is a technique that entails running a sample of fluid/oil over an inclined glass slide. This slide is prepared so that the particles will adhere to the slide as they pass over it. The particles are distributed along the slide from larger at the start to smaller at the lower end. There is also a varying magnetic field along the length of the slide that will cause the magnetic particles to align themselves with the field identifying them as ferrous particles. Used to analyze the particle count, ferrous content and non ferrous contamination debris. Methods of varying degrees of complexity and cost can be employed from microscopic examination with reflected and or transmitted light with red, green or polarized filters to aid visualization. An electron microscope can also be used to examine the wear particles for closer analysis on determination of which machine component they came from and the method in which they were dislodge from their origin.

2.3.2 Analytical Ferrography

Analytic Ferrography uses a machine to prepare the ferrogram and scans it to conduct an analysis. The machine automatically reports findings on number, size and type of particles, and wear mechanisms based on shape of particles

2.3.3 Direct Reading (DR) Ferrography

A direct reading ferrogram subjects a sample to a magnetic field to separate ferrous material. Light is projected through the area where the ferrous material is deposited and based on the strength of the light passing through the deposit in two areas, two trend-able readings are given based on particle sizes of greater than and less than 5 microns.

2.3.4 Mesh Obscuration (MO) Particle Counter

This test uses an instrument to measure the pressure differential across three mesh sizes, for example 5,15 and 25 microns. Based on the pressure difference across each mesh the quantity of particles larger than each screen size is determined. It is not able to distinguish whether particles are wear or contamination. This can be translated to an

ISO4406 cleanliness level depending on mesh sizes used.

2.3.5 Pore Blockage (PB) Particle Count (Flow Decay)

A fluid sample is pushed under pressure through fine precision screens, for example, 5, 10 and 15 microns. Over time as the particles block the screens the flow is reduced. The produces a flow vs time decay curve which a mathematical program converts to a particle size distribution. This reading can be converted to an ISO4406 cleanliness level

2.3.6 Light Extinction (LE) Particle Counter

An incandescent light is shone through a fluid as it passes through the particle counter at a specific volumetric flow rate. A photo-voltaic diode detects the quantity of light passing through the sample and as the particles cause interruption to the light the instrument measures the amount of light reflected and how much passes through. The output voltage of the diode is used to determine the quantity and size of the particles. This can be converted to an ISO4406 cleanliness level.

2.3.7 Light Scattering particle Counter

Uses a laser light, an object cell for the fluid to pass through and a photo-voltaic diode. Fluid passes through the cell at a specific volumetric flow rate. The laser light hitting the diode changes as particles pass through the cell and the reading of the diode output is used to determine particle size and quantity. This can be converted to an ISO cleanliness level.

2.3.8 Real Time Ferromagnetic Sensor

The sensor is an electromagnet and it attracts the magnetic particles as they pass and holds them. The frequency of the current is effected relative to the mass of the particles. The test is run over a set time to give readings that can be trended over time. This only tells the mass of ferrous material.

2.3.9 All Metal Debris Sensor

(Miller, Kitaljevich 2006) Three wound coils are positioned on a section of pipe which the oil flows through. The two outer coils are powered with high frequency current in opposite directions. The center coil is the sensor coil. As a particle passes the first and third coils it disturbs their magnetic fields which in turn generate an output on the center sense coil. The mass of a ferrous particle is proportional to the output signal and the surface area of a non ferrous conductive particle is proportional to the output signal. A ferrous conductive particle produce a signal with opposite phase to that of a non ferrous conductive particle. Needs to be in a return line before filtration/settling.

2.3.10 Graded Filtration

A sample of diluted oil is passed through a series of filtration discs. Discs are examined visually for the number and size of particles. The distribution of particles is graphed and the profile of graphs is what determines whether wear is normal or not.

2.3.11 Magnetic Chip Detection

This is the use of a magnetic plug that is exposed to the fluid. The magnetic particles are held on the plug by the magnet. The magnet plug is removed periodically for visual or microscopic inspection. The plug must not be down stream of filtration units.

2.3.12 Blot Testing

Drops of oil are placed on blotting paper. The larger particles are left in the center as the oil is drawn outward by the paper. The finer particles are carried outward by the oil as it spreads, the finer the particle the further it is from the center. A clearly visible ring is present around the blotted area if there is sludge in the sample. It can take up to 24 hours for a sample to finish but hours or less is often sufficient. The oil can be heated briefly to $240°C$ (Troyer 1999) and if the oil is near failing from thermal stress or oxidization it will show on the blot test.

2.3.13 Patch Test

A volume of test fluid is drawn by vacuum through 0.8-5 micron patch or disc. The degree of discoloration is used to determine the level of contamination by comparing it to a chart, from which an ISO cleanliness rating can be given. The patch can be examined under a microscope to determine type and size of particles. Available as a portable kit.

2.4 Monitoring Water in Oil

Moubray (1997) notes that "water in oil can reduce bearing life by up to 100 times" and that "one drop of water in 5 liters of oil at $85°C$ totally destroys zinc anti-wear additives". Water in oil also increases oxidation, reacts with additives to form acids, salts, slime, sludge and promotes microbe growth. It also affects oil viscosity, corrodes components, increases wear, blocks valves and enables more air to be suspended in oil reducing its lubricating ability. Techniques for detecting water in oil include:

2.4.1 Calcium Hydride Water Test

Water reacts with calcium hydride and releases hydrogen gas. Prescribed quantities of sample oil and calcium hydride are place into a container that has a partition to separate the contents before it is closed and sealed. The container is then shaken to mix its contents and begin the reaction. The quantity of water in a sample can be determined by the quantity of gas released, by measuring pressure in the container and reading off a chart the pressure to water quantity equivalent.

2.4.2 Karl Fischer Titration Test

Can be either volumetric or coulometric titration to determine the quantity of water in oil.

- The Coulometric method relies on the oxidization of sulfur dioxide by iodine a reaction that uses the water in the oil. The current the reaction creates across an anode and cathode in the solution is measured. A sensing current is run through

the solution by another anode-cathode in the solution and the voltage drop across the sensing circuit signals the end of the reaction. The charge required to complete the reaction is a measure of quantity of water in the sample tested. Can detect water as low as $1ppm$. $1mg$ water equivalent to 10.72C charge.

- The volumetric method requires measured addition of iodine and reaction ends when the water in the sample is used. The end of the reaction is sensed by a drop in voltage across an anode/cathode in the solution at which point the addition of reactant is stopped. The amount of reactant used determines the quantity of water present in the sample. Volumetric titration is not suitable for less than $100ppm$. Used for lubricating,hydraulic and transformer oils.

2.4.3 Crackle Test (Human Sensed)

Used for lubricating,hydraulic and transformer oils. Several drops of oil are place onto a hot plate at 120-160°C. The water in the sample vaporises and makes an audible crackling sound. Bubbling of the water and spitting may also be visible. Water content below $300ppm$ not easily heard. (Moubray 1997)

2.4.4 Crackle Test (Audio detector)

Several drops of oil are place onto a hot plate at at 120-160°C. The moisture in the oil will vaporise and a microphone detects the noise. A computerized collector converts the signal threshold crossings into a ppm reading. Water content as low as $25ppm$ detectable. Used for lubricating,hydraulic and transformer oils. (Moubray 1997)

2.4.5 Moisture Monitor (Vapor Induced Scintillation)

Uses a hand held instrument.The unit has a probe with a small heated element that is submersed in a sample. A microphone detects the water vaporising and a computerized collector converts signal threshold crossings into a ppm reading. Water content as low as $25ppm$ detectable. Used for lubricating,hydraulic and transformer oils. (Moubray 1997)

2.4.6 Clear and Bright Test

A visual inspection to assess whether the oil is become hazy. Low level of suspended water in the oil may be difficult to detect visually. Generally a high concentration of water is present once noticeable as the water in oil may need to be at the saturation point to be distinguishable. Severe levels of water present when emulsified milky white.

2.5 Chemical Condition Monitoring

Chemical monitoring techniques used on oils and fuels of rotating machinery and can be used to detect indicators of machine wear. Chemical monitoring is also used to detect chemicals associated with degradation of a fluid and its additives as well as contamination.

2.5.1 Atomic Emission (AE) Spectroscopy

The three methods following are variation on the same theme. The objective of them is to elevate energy state of the particles in the sample and cause them to emit light energy. This light is diffracted into the different characteristic wavelengths according to each element. The intensity of each elements emission is proportional to the elements concentration in the sample. A detector (such as a vacuum photomultiplier, photo diode or silicon photomultiplier measures each element line intensity and a computer analyses and outputs results. They are accurate and fast. Instruments are available to measure individual element light lines by means of a movable grating and a single photo detector requiring a burn of sample for each element to be analyzed thus a longer time to sample a range of elements. May be unable to atomize particles over 8 microns.

- Electric Arc or a Flame AE - An electric arc or a flame is used to vaporise a sample portion. Provides within several parts per million (ppm) accuracy.
- Rotating Disc Electrode AE - A rotating graphite disc is immersed in a sample and picks it up as it rotates. The disc and sample are subject to a high temperature electric arc that completely atomizes the sample on the disc and in this process the elements in the sample emit light. Provides parts per million accuracy.

- Inductively Coupled Plasma AE - Argon gas is heated to between $8000K$ to $10000K$ by a strong radio frequency magnetic field producing plasma. The oil sample is diluted with a low viscosity solvent so it can be nebulized and carried by the argon gas into the plasma stream. The high temperature excites the particles in the sample which radiate their characteristic emissions. Very accurate and can provide parts per billion accuracy.

2.5.2 Atomic Absorption (AA) Spectroscopy

AA works on the principle that a material atom absorbs light of a specific wavelength. The sample is diluted and atomized by heat. Atomization can be by nebulizing the sample into a hot flame, for example an acetylene mixture or as it passes through an electrically heated graphite tube. The flame or tube outlet is irradiated by a specialized lamp with the characteristic wavelength of the metal tested for. The higher the concentration of that metal in the sample the higher the absorption of light. The absorption is measured and computed output given.

2.5.3 Thin Layer Activation

This involves irradiating the surfaces to be monitored with a beam of radioactively charged particles. This makes a thin layer of material radioactive. The radioactivity of the component can be used as a measurement as radioactive strength will decrease with wear. The radioactivity of wear particles can also be used as a measure of wear. It is used for monitoring wear on turbine blades, engine cylinders, shafts, bearings, electrical contacts and cooling systems. Wear of 1 micron can be detected but reactivation is required every four years. Can be used to monitor wear on inaccessible components while a machine is in service.

2.5.4 X-Ray Fluorescence Spectroscopy

High energy X-Ray is used to bombard the sample to raise the energy level of the contaminants. The contaminants in the sample emit a secondary X-Ray which is characteristic of the elements present, known as their characteristic fluorescence. An analyzer is used

to convert the characteristic emissions and their intensity to a result. Can detect any particle size, large or small. Good accuracy, precision and repeatability. Able to detect more elements than AA or AE. Accurate results require a cryogenically cooled detector. X-Ray health precautions are required. Reveals wear metals and contaminants such as silicon and corrosion.

2.6 Lubrication Condition Monitoring

The condition of lubrication is an important aspect of machine health. Oils and greases are complex and tailored to suit load, temperature, speed and environment. Many adverse lubricant conditions can affect machine life. The wrong type of lubricant, wrong viscosity, wrong additives, depleted, oxidized or worn out aspects of the oil that stop it operating as designed. Failure of lubricant will lead to premature failure of machine.

2.6.1 Viscosity Monitoring

Some techniques for monitoring viscosity are listed.

- Viscosity Monitor - Hand held units are available. A probe is placed in the oil for an instant readout.
- Falling Ball Comparator - Based on Stoke Law a ball with a small is dropped into a tube filled with sample, allowed to reach terminal velocity then timed over a distance. Available in a kit with a set range of balls and conversion chart for time to viscosity.
- Kinematic Viscosity test - Instruments are available to conduct this test. It works on measuring the time taken for a given quantity of oil at a certain pressure to flow through a calibrated tube and referencing this time to a charted viscosity.

2.6.2 Fourier Transform Infrared (FT-IR) Spectroscopy

FT-IR uses the light emissions of various elements to determine their concentration in a sample. Uses a special broadband infra red beam that is passed through a sample. As

it passes through it is altered by the characteristic absorbency of the contaminants. The altered beam enters a detector which is converted into an audible electronic signal and then converted into individual wavelength and amplitude data by a Fourier transform. The contaminated sample is compared to data of an unused sample.

2.6.3 Ultra Violet and Visible Absorption Spectroscopy

Works on the principle that elements will absorb light of a certain wavelength. The sample and contamination elements are exposed to a high energy ultra violet (UV) hydrogen or deuterium lamp, or a visible light such as a tungsten lamp. The amount of light absorbed by the energized particles is measured using a wavelength separator such as a prism or a diffraction grating. Concentrations can be assessed by scanning the whole spectrum or by focusing on the particular characteristic wavelength of an element. Used to measure changes in oil condition such as alkalinity, acidity and insolubles from combustion. For oils of engines, gas turbines, transmissions, gearboxes, compressors and hydraulic systems.

2.6.4 Color Indicator Titration

This is used to measure mineral oil acidity and alkalinity. The higher the reading the greater the deterioration of the oil. Sample is dissolved into a mixture of toluene, isopropyl alcohol and water and titrated with an alcoholic base or acid solution to the titration endpoint indicated by a color change. For oils of engines, gas turbines, transmissions, gearboxes, compressors and hydraulic systems.

2.6.5 Total Acid Number - Total Base Number (TAN/TBN)

Determines the deterioration of mineral oil by a measurement of acidity or alkalinity. Sample is dissolved in a Ph neutral solvent. Acidic oil is titrated with potassium hydroxide and a basic oil titrated with hydrochloric acid. The titration point can determined by measuring change in the solutions electrical conductivity as the titrate is added. The titration neutral point can also be determined by measuring light that passes through the sample during titration, sample temperature during titration and observing changing color. The TAN/TBN is the amount of titrate required to neutralize a sample of oil

in mg/g. For oils of engines, gas turbines, transmissions, gearboxes, compressors and hydraulic systems.

2.6.6 Exhaust Emission Analyzer (Four gas Analysis)

Used to analyze combustion efficiency by measuring oxygen, carbon monoxide, carbon dioxide and hydrocarbons in exhaust gas. The analyzer is inserted into the exhaust pipe for a reading. High CO means engine is running rich. High oxygen indicates it is a lean mixture or and exhaust leak. CO_2 is at a maximum at the optimum fuel/air ratio. and drops when it is too lean or rich. High hydrocarbon indicate a misfire or incomplete combustion.

2.7 Additional Condition Monitoring Techniques

There are numerous other condition monitoring techniques listed by Moubray (1997) including the following.

Temperature Effect Monitoring

- Focal Plane Arrays
- Fibre Loop Thermometry
- Temperature Indicating Paints

Physical Effect Monitoring

- Liquid Dye Penetrator
- Electrostatic Fluorescent penetrator
- Magnetic Particle Inspection
- Strippable Magnetic Film
- Eddy Current Testing

- X-ray Radiography
- X-ray Radiographic Flouroscopy
- Rigid Boroscopes
- Cold Light Rigid Probes
- Deep Probe Endoscope
- Pan-view Fiberscopes
- Electron Fractography
- Strain Gauge (Strain)

Electrical Effects Monitoring

- Line Polarization resistance (Corrator)
- Electrical Resistance (Corrometer)
- Potential Monitoring
- Power Factor Testing
- Electrical Surge Comparison
- Motor Current Signature Analysis
- Power Signature Analysis
- Partial Discharge
- High Potential testing (Hi-Pot)
- Magnetic Flux Analysis
- Battery Impedance Test

2.8 Vibration Monitoring

Vibration analysis as a means of monitoring the condition of rotating machinery has been around for decades. The modes of failure that can be detected dynamically as vibration

include damage from effects of bearing fatigue, damage, wear, geometric anomalies, effects of corrosion, brinelling and false brinelling, lubrication issues between rollers and cage - rollers and raceways - cage and raceway.

In a rolling element bearing that lasts its full life fatigue will be a normal failure mode. The cyclical stress on elements initiates sub surface cracking that will often develop at the interface between the harder surface material and the tougher sub surface structure. As the size of cracking increases with fatigue a small piece of the hardened surface will break away and bearing elements will impact on the surface defect.

Brinelling occurs from overloading a stationary bearing and results in plastic deformation, often seen as roller indentations on inner or outer raceways and flattened areas on rollers. False brinelling also occurs while the bearing is in a stationary state with mechanisms of corrosion and fretting from relative movement of components. These mechanisms can occur when the lubrication separating elements is pushed out over time and exposes metal to metal contact of components to fretting, exposure of metal to moisture and oxidization. Corrosion can also occur as a result of moisture reacting with the lubricant ant the creation of acids.

Damage from corrosion can occur when contaminants enter the system such as water, process materials of fluids. Water and other fluids can have the effect of reacting with oil additives to create acids detrimental to bearing life. This can occur with inappropriate vent breathers, leaking cooling systems, damaged seals, or the machine being subject to a wetter environment than it was designed for.

Bearing wear is caused mainly by dirt and materials entering the bearing causing varying degrees of element surface damage. This can occur because of human error with introduction of contaminated lubrication error, defective of inappropriate vent breathers or oil system filtration, damaged seals or loose machine components compromising sealed compartments. The effects of wear from a grinding action can include changes in the dimensions of bearing components (geometric errors) from a grinding action, changed contact areas within bearing, increased rolling friction, slipping and skidding in bearings, increased internal clearances in all machine components such as gearing and hydraulic systems. Larger particle contamination can initiate surface defects causing impacting and accelerated fatigue damage.

Inadequate or incorrect lubrication can cause bearing rollers to skid which causes friction. This causes a temperature increase and if friction is great enough and the heat generated cannot be dissipated then the temperature can rise until bearing elements have their metallurgical properties changed, the cage may melt or be deformed, the bearing elements may weld together and cause complete destructive failure. Temperature increases can cause inner raceways fitted to shafts to spin if it expands too much from being heated.

Skidding can also occur in bearings as a result of poor bearing choice or incorrect use of equipment. A bearing is designed to work with a minimum load and without this load and adequate internal friction, to make the bearing elements roll, they have a tendency to skid. Too heavy a lubrication for a given application can also cause bearing rollers to skid.

The temperature range a machine is exposed to will affect component clearance due to thermal expansion. Operating outside a designed temperature range can cause excess or inadequate load on bearings. It may cause gears to mesh incorrectly or a machine to run with misalignment, can render lubrication inadequate or inappropriate if a separation film cannot be sustained or it may cause inadequate load on bearing elements casing rollers to skid.

2.8.1 Simple Harmonic Motion

This term is used to describe the motion of a mass connected to a spring with ideal conditions of no friction or losses. In this case the mass continues oscillate back and forward. The number of oscillations that occur in one second is known as the frequency (f) and has the unit of Hertz (Hz). Hookes law defines the amount of force $F(N)$ required to deflect a spring of stiffness $k(N/m)$ by a displacement of $x(m)$

$$F = kx \tag{2.1}$$

Once the force is known Newtons second law 2.2 can be used in conjunction with Hookes law to determine the acceleration of the object.

$$F = m\ a \tag{2.2}$$

,

$$a = \frac{kx}{m} \tag{2.3}$$

Similarly a point on circle rotating at a constant speed, viewed in one plane, moves backward and forward with the same harmonic motion that a mass on a spring does. The displacement of a mass moving in simple harmonic motion is sinusoidal an can be defined as

$$x = A\sin(\omega t) \tag{2.4}$$

where A is the maximum displacement, ω is the angular velocity (rad/s) and t is time (s).

In both these cases, linear oscillation and rotating motion, at the extremity of displacement the object changes direction and at this instant the displacement is a maximum amplitude, the velocity is zero and acceleration is a maximum amplitude. The object continues to accelerate until it passes the central point where displacement is zero, velocity is at maximum amplitude and acceleration amplitude is zero. Displacement, velocity and acceleration are vector quantities meaning they have both an amplitude and a direction.

2.8.2 Time and Frequency Domains

Vibration as acceleration, velocity or displacement is represented in the time domain as a time waveform. This is graphed as time on the x axis and vibration amplitude on the y axis. A basic time waveform describing simple harmonic motion is made by a sine wave. The frequency of a sine wave in Hertz (Hz) is the number of cycles completed in one second. For a shaft rotating at 1 Hz it completes one rotation/cycle per second. The basic principal is that an imbalance vibration on this shaft will represent a sine wave in the time waveform with a frequency of 1 Hz. This time waveform can be graphed in the frequency domain and is known as a frequency spectrum. The x axis units are frequency and the y-axis remains as vibration amplitude. For the case of a 1 Hz shaft the frequency

spectrum will have a line at Hz on the frequency axis of the same amplitude as shown in the waveform.

2.8.3 Fourier Transform

Using a process called the Discrete Fourier Transform (DFT) it is possible to break a time domain signal into constituent sinusoids with real/amplitude and imaginary/phase components.

$$F(m) = \sum_{n=0}^{N-1} f(n) e^{-2\pi imn/N}, \quad m = 0, 1, 2, ..., N-1 \tag{2.5}$$

where n and m are the index of the input and output, N is the size of the sample.

The Fast Fourier Transform (FFT) is a method by which the number of calculations required to perform the DFT is reduced. Briefly it is done by sorting and dividing the signal in two as many times as possible, the reason why binary storage files are required for the most efficient FFT algorithms. The DFT calculations are performed on the smaller arrays in a structured fashion diagrammatically known as butterflies, reducing the sorting and merging process to give the compiled result.

As an example similar to that shown by Osgood (2007) the calculations required by the FFT process when the number of points in a waveform is $N = 2^{14} = 16384$ (giving $6400LOR$), is in the order of $O(N\log N)$, that gives $2^{14}\log_2 2^{14} \approx 229 \times 10^3$. By applying the DFT process directly to the initial array calculations are in the order of $O(N^2)$, that is $2^{2\times 14} \approx 268 \times 10^6$ calculations, that is near 1200 times greater.

2.8.4 Frequency and Period

One period T is the time it takes to complete one cycle or one revolution. The relationship between period and frequency is

$$T = 1/f \tag{2.6}$$

2.8.5 The Concept of Phase

Phase is a relative concept where for example the position in time of the peak of one sine wave is compared to position in time of another sine wave peak. The measurement of phase is normally given as an angle and describes the difference between the peaks of both the sine waves. A sine wave can also have its phase shifted by adding or subtracting the desired phase angle ϕ

$$x = A\sin(\omega t \pm \phi) \tag{2.7}$$

2.8.6 Frequency Units

In vibration analysis the rate of rotation of a shaft may be represented in a variety of units

- $Hertz(Hz)$ Cycles per second
- RPM/CPM Revolution/Cycles per minute
- $Orders$ The speed of a shaft can be described as $1order$. This is the frequency of interest as a ratio of shaft speed. If the frequency of interest is $100RPM$ and the shaft speed is $100RPM$ then the frequency of interest can be described as $1order$.

Converting between Hz and RPM is given by the relationship

$$f(Hz) = RPM/60 \tag{2.8}$$

2.8.7 Amplitudes and Units

Vibration amplitude is represented as units of

- Displacement - Imperial units of inches (in) or thousandths of an inch ($mils$) or metric units of millimetres (mm) or microns

- Velocity - Commonly as Imperial units of inches per second (in/s) or Metric units of millimetres per second (mm/s)
- Acceleration - Imperial units of inches per second per second (in/s^2), Metric units of metres per second per second (m/s^2) and commonly as G's, 1G being the acceleration due to gravity given as $9.81m/s^2$

Conversion between metric and imperial units is given by the relationship that 1 $inch = 25.4mm$.

For vibration analysis integration is the mathematical process converting acceleration to velocity and velocity to displacement. It can be done by a process of calculating the area under a curve for each time step. The following equation shows how breaking the area under a curve into n slices of width Δ t the area of each slice is calculated and summed. Taking more slices makes the calculation more accurate to the point where the integral over an interval from a to b is equivalent to taking infinite slices over that interval.

$$lim_{n\to\infty}\sum_{i=1}^{n} f(t_i)\Delta t = \int_b^a f(t)dt \tag{2.9}$$

For a known function such as a sine wave there are rules governing the integration of the curves function and the integral can be calculated directly and accurately.

Differentiation is the process of going from displacement to velocity and from velocity to acceleration. Differentiation is a process of calculating and plotting the slope of a curve (rise/run) at each time step. Again the smaller the steps are the more accurate the result to the point of making them infinitely small and the results can be calculated directly using predefined rules on specific functions.

$$f'(t) = lim_{\Delta t\to 0}\frac{f(t+\Delta t) - f(t)}{\Delta t} \tag{2.10}$$

The measurements of displacement, velocity and acceleration units can be presented in the following ways

- RMS This is the square root of the mean of the square of the waveform and is

representative of the level of constant energy or constant vibration level that is present. The RMS level for a sine wave is

$$x_{RMS} = A_{max}/\sqrt{2} \tag{2.11}$$

- $Peak(P)$ This measure describes the amplitude of vibration from 0 to the maximum peak value.
- $Peak\ to\ Peak(P-P)$ This is a measure of amplitude from the minimum peak level in the time waveform to the maximum peak level.

2.8.8 Data Sample Properties

Measuring vibration usually involves measuring and recording a sensor voltage. The instruments used to record a data sample often requires some input on parameters regarding the sample it is to record. These input values affect the output qualities of data and determine what defects will and will not be captured in the recording.

Some instruments, for example, will record 50000 samples per second. The user in this case has the choice of selecting the amount of time to sample. Nyquist theorem states that for a maximum frequency of $fmax$ to be captured in a waveform the frequency of data sampling must be at least $2 \times fmax$. Theoretically up to a 25000Hz spectrum could be extracted from this waveform. The $fmin$ is determined by the length of the time recorded where at least one cycle must be recorded.

Common hand held data collectors use binary storage of fixed file sizes ranging from $2^{10} - 2^{15}$ bits, which give the number of points in a recorded time waveform. This conventional file sizing also makes computing the Fast Fourier Transform (FFT) faster thought improved modern algorithms have reduced the time to process the FFT on data files of an unconventional size. Hardware introduces a degree of error in the process of turning an analogue signal into a digital signal. Hardware digitizing the signal guarantees a quality signal to 78.125% of the Nyquist sample rate and for this reason the higher frequency portion is not used in the frequency domain. This further removes aliasing that may occur in the upper frequency range. For these units 2.56 is the factor that gives 78.125% of the Nyquist frequency. Therefore the sampling rate determines the

frequency lines of resolution (LOR) that will be in the given the frequency domain. Lines of resolution are also known as frequency bins of FFT bins. The following shows how the number of points in a time waveform equates to lines of resolution for a frequency spectrum.

- $2^{10}/2.56 = 1024/2.56 = 400LOR$
- $2^{11}/2.56 = 2048/2.56 = 800LOR$
- $2^{12}/2.56 = 4096/2.56 = 1600LOR$
- $2^{13}/2.56 = 8192/2.56 = 3200LOR$
- $2^{14}/2.56 = 16384/2.56 = 6400LOR$
- $2^{15}/2.56 = 32768/2.56 = 12800LOR$

With these units one also has to choose a maximum frequency $fmax$ as a collection parameter. This will also be the $fmax$ in the resultant spectrum. For example if a $fmax = 2000Hz$ and $6400LOR$ are chosen, the time duration of the waveform will be $6400/2000Hz = 3.2s$ and contain 16384 points. The spectrum for this point will have an $fmax = 2000Hz$ and with $6400LOR$ will display $2000Hz/6400 = 0.3125Hz$ between plotted points.

2.8.9 Aliasing

Aliasing occurs when the sampling rate is not high enough to accurately represent the changes that occur in the analog signal and what happens is a lower frequency than is present can be recorded in the data. An example of this is shown in figure 2.3 on page 27 where a 5Hz signal is sampled at 5.26Hz and a resulting low frequency signal is created that is not present.

2.8.10 Natural Frequency

For the simple harmonic motion of a mass attached to a spring there are two factors that effect the natural frequency of system, the magnitude of the mass and the stiffness of

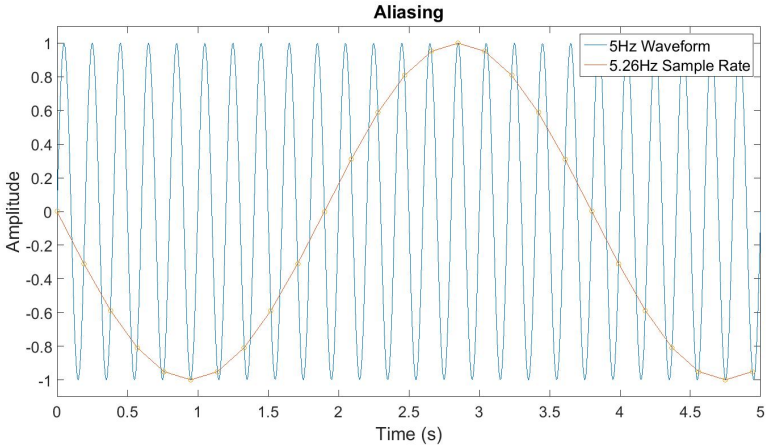


Figure 2.3: This shows aliasing of a 5Hz signal sampled at 5.26Hz and the lower frequency that appears to be present.

the spring. The natural frequency is the frequency that the mass will oscillate back and forward at once released and it given by

$$\omega_n = \sqrt{\frac{k}{m}} \tag{2.12}$$

In the general sense this applies to machinery, structures and vibrating systems where increasing the vibrating mass will reduce the natural frequency and increasing the stiffness will increase the natural frequency.

2.8.11 Damping

Damping is a system property that removes energy from from a vibrating system in a way that reduce the maximum amplitude of a freely oscillating system. Viscous damping c is a linear property where the resulting force is proportional and in the opposite direction to the velocity of the system mass.

$$F = cv \tag{2.13}$$

Where c the system damping coefficient has units of $N \cdot s/m$ and v is the velocity m/s of the system mass. Structural damping is also a linear property, suitable for a system with a sinusoidal forcing function, with equivalent damping defined as

$$c_{eq} = \frac{2k\beta}{\omega} \tag{2.14}$$

where β and empirically obtained system constant

The critical damping c_c of an oscillating system is defined by

$$c_c = 2m\omega_n = 2\sqrt{km} \tag{2.15}$$

Where c_c the system critical damping coefficient, units of $N \cdot s/m$.

The damping ratio ζ for a system is the ratio of system damping to critical damping coefficients and is has no units.

$$\zeta = c/c_c \tag{2.16}$$

A system with $\zeta = 1$ is critically damped. A system with ζ <1 is under damped and will overshoot the equilibrium point on oscillation and an over damped system with ζ >1 is over damped taking longer to reach the equilibrium point than a critically or under damped system.

Damping also has an effects on the natural frequency. The damped natural frequency ω_d is given by

$$\omega_d = \omega_n\sqrt{1-\zeta^2} \tag{2.17}$$

2.8.12 Logarithmic Decay

When an under damped single degree of freedom system is subject to an impact and left to settle, the rate of decay is defined by Balachandran (2009, p154) as “the logarithmic decrement δ is defined as the natural logarithm of the ratio of any two successive amplitudes of the response that occur at a period T_d apart”.

$$\delta = \frac{1}{p} ln\left(\frac{x(t)}{x(t+pT_d)}\right) \quad p = 1, 2, 3, ... \tag{2.18}$$

Once δ has been determined the following equation can be used to calculate the damping ratio

$$\zeta = \frac{1}{\sqrt{1 + (2\pi/\delta)^2}} \tag{2.19}$$

A bump test commonly mentioned as a test for natural frequency involves recording the response of a machine component when impacted. Analysis of the waveform and these equations can be used to determine the damping ratio.

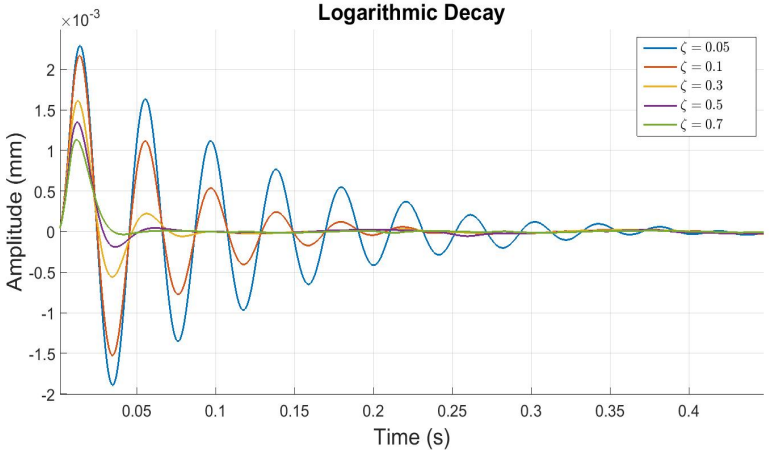


Figure 2.4: The following shows the logarithmic decay of vibration for under damped system with various degrees of damping.

2.8.13 Resonance

If the natural frequency of the structure a machine mounted on is near the forcing frequency then the magnitude of vibration will be increased by a factor dependent an system damping ratio ζ and frequency ratio Ω. This magnification of amplitude is a condition known as resonance.

The frequency ratio Ω is the ratio of the forcing frequency to the natural frequency

$$\Omega = \frac{\omega}{\omega_n} \tag{2.20}$$

The amplitude response factor is also known as transmissibility. There is also an affect on the phase of the response dependent on the system properties. The amplitude response $H(\Omega)$ and phase response θ for a system with a rotating unbalance mass is given by the following

$$H(\Omega)_{ub} = \frac{\Omega^2}{\sqrt{(1-\Omega)^2)^2 + (2\zeta\Omega)^2}} \tag{2.21}$$

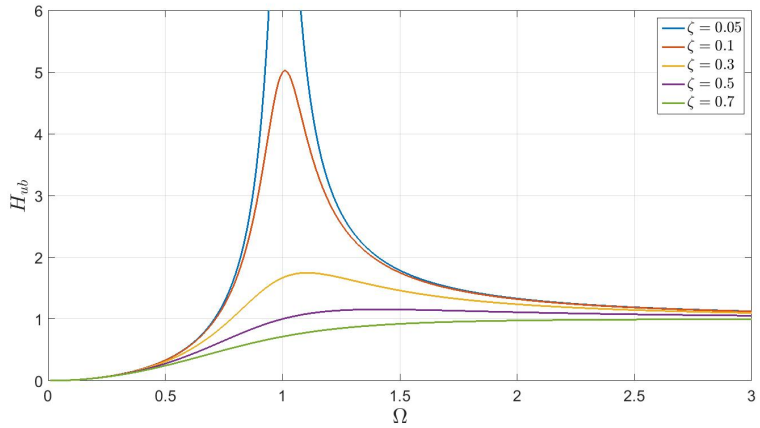


Figure 2.5: This shows the amplitude response vs the frequency ratio for a single degree of freedom system with a rotating unbalance acting directly on the system mass for several values of damping ratio.

$$\theta_{ub} = tan^{-1}\frac{2\zeta\Omega}{1-\Omega^2} \tag{2.22}$$

The amplitude response of a system that has base excitation is defined as

$$H(\Omega)_{mb} = \frac{\sqrt{1+(2\zeta\Omega)^2}}{\sqrt{(1-\Omega^2)^2 + (2\zeta\Omega)^2}} \tag{2.23}$$

$$\theta_{mb} = tan^{-1}\frac{2\zeta\Omega^3}{1+\Omega^2(4\zeta^2-1)} \tag{2.24}$$

These transmissibility equations show that for the case where a forcing frequency coincides with a natural frequency, if damping is not sufficient, magnification of vibration (resonance) will occur and this will be destructive to a machine and lead to premature failure.

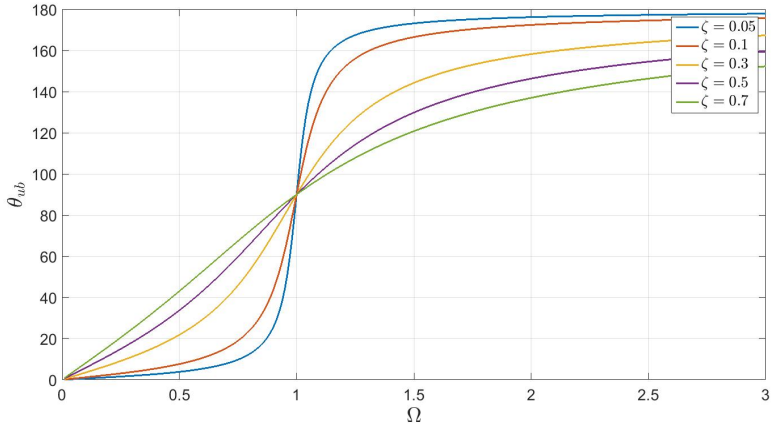


Figure 2.6: This shows the phase response vs the frequency ratio for a single degree of freedom system with a rotating unbalance acting directly on the system mass for several values of damping ratio.

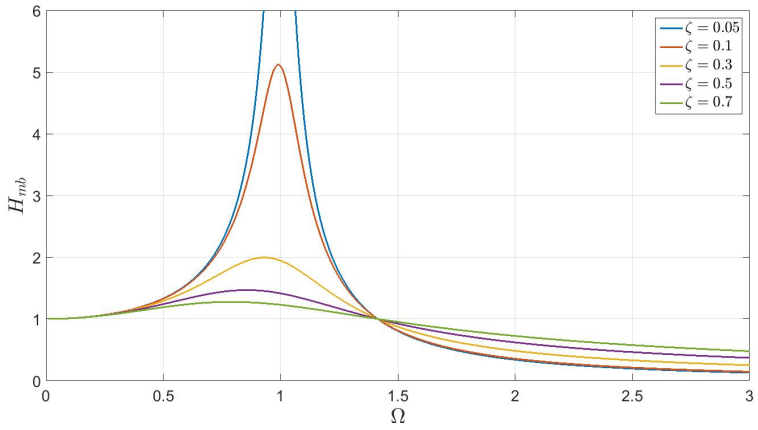


Figure 2.7: This shows the amplitude response vs the frequency ratio for several values of damping ratio. This is for a single degree of freedom system where the excitation/input force is external to the system mass.

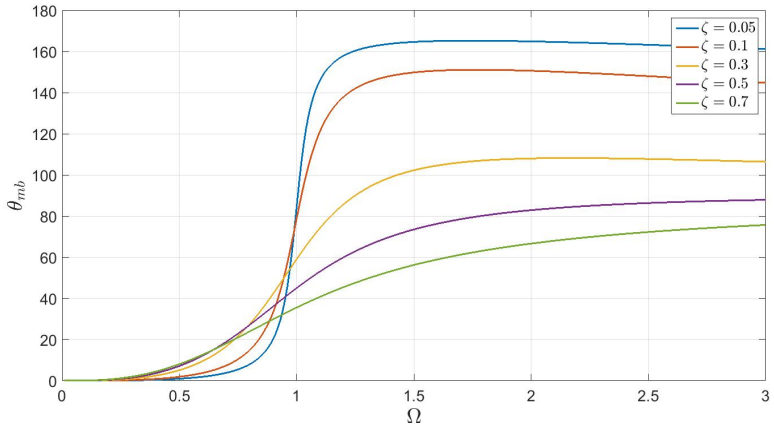


Figure 2.8: This shows the phase response vs the frequency ratio for a single degree of freedom system with a rotating unbalance acting directly on the system mass for several values of damping ratio.

2.8.14 Time Windows

The FFT process assumes that the time domain signal begins at zero amplitude, is purely periodic and end again at zero. This can be the case for a sine wave and it is also the case for an impulse starting at zero before the pulse and decayed to zero at the other end of the waveform. Other than such ideal cases without time windows the FFT process will cause what is called leakage in the frequency domain from the bin where it should be into adjacent bins. This is known as Gibbs phenomenon. Time windows are easiest applied prior to the fft process converting from the time domain to the frequency domain though they can be applied to the frequency domain. Because of the effect windows have on a waveform amplitude across the time domain the results of the FFT must be increased by a factor to compensate. For the Hanning window the correction factor is 2. A rectangular window is essentially having no window at all on a waveform. Time windows also have an effect on frequency and amplitude in the spectrum and there is often a compromise between having accuracy with amplitude or frequency definition.

Rectangular Window figure 2.9 on page 33

Hamming Window figure 2.10 on page 34

Hanning Window figure 2.11 on page 34

Blackman-Harris time window, figure 2.12 on page 34

Table 2.1: This table shows attributes of common windows (LDS Group 2003)

Window	Best for these Signal Types	Frequency Resolution	Spectral Leakage	Amplitude Accuracy
Barlett	Random	Good	Fair	Fair
Blackman	Random or Mixed	Poor	Best	Good
Flat Top	Sinusoids	Poor	Good	Best
Hanning	Random	Good	Good	Fair
Hamming	Random	Good	Fair	Fair
Kaiser-Bessel	Random	Fair	Good	Good
None / Rectangular	Transient and Synchronous Sampling	Best	Poor	Poor
Tukey	Random	Good	Poor	Poor
Welch	Random	Good	Good	Fair

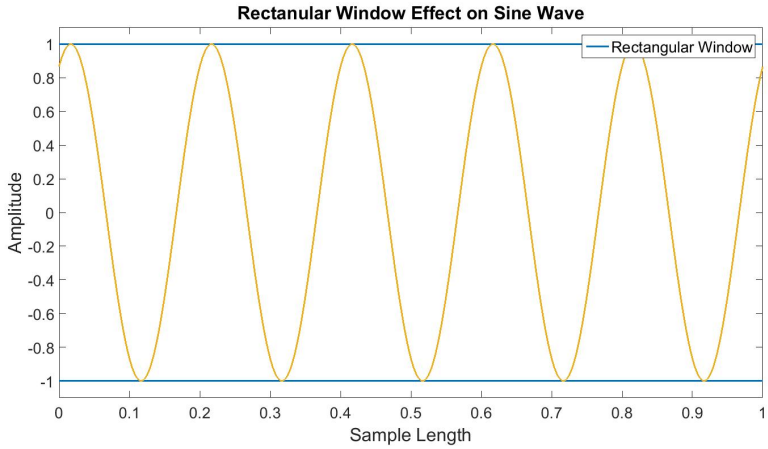


Figure 2.9: This shows the rectangular window having no effect on the sinusoid

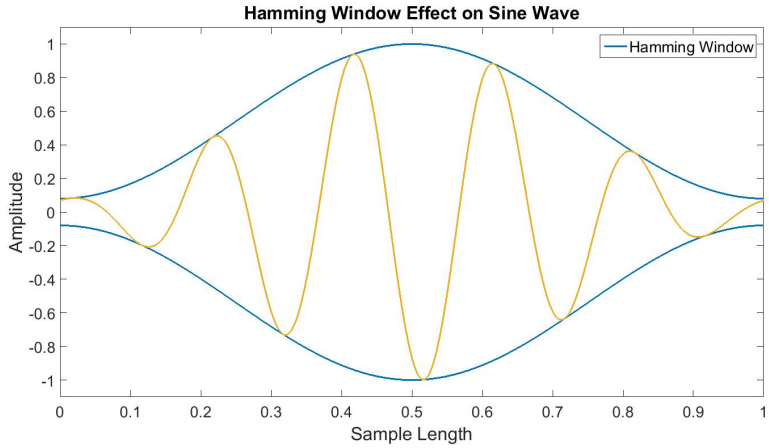


Figure 2.10: This shows the effect of the Hamming window on a sine wave

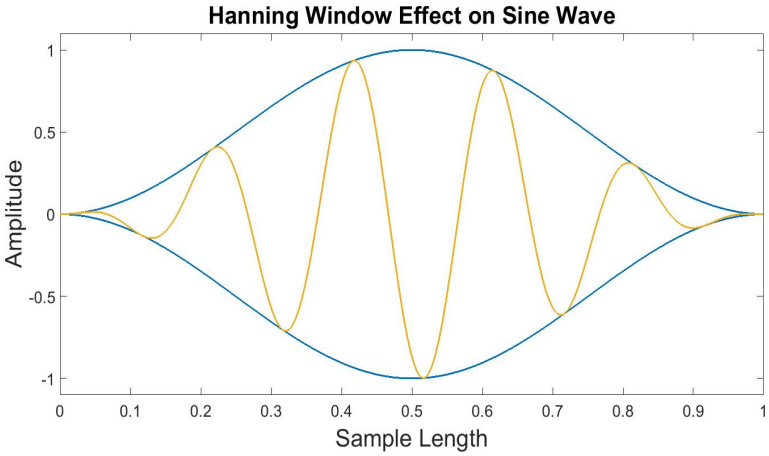


Figure 2.11: This shows the effect of the Hanning window on a sine wave

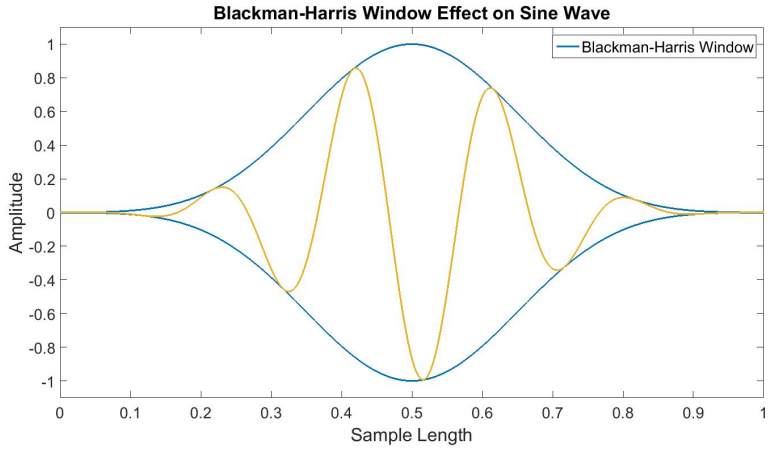


Figure 2.12: This shows the effect on the Blackman Harris window on a sinusoid

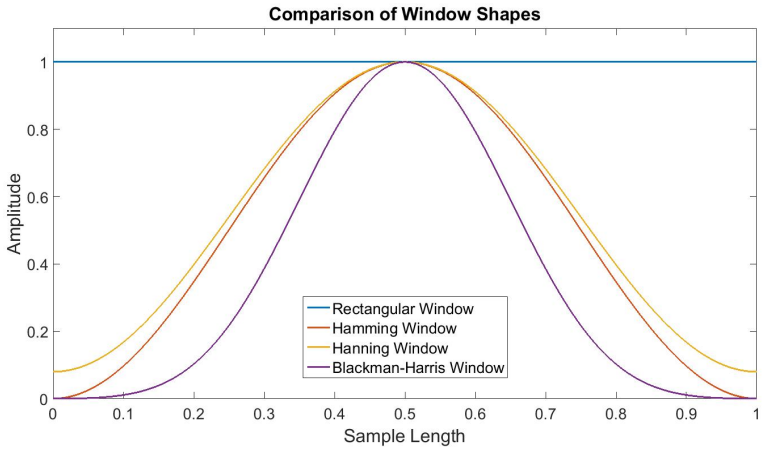


Figure 2.13: This shows a comparison of the window shapes graphed together

Comparison of window shapes figure 2.13 on page 35

2.8.15 Vibration Measuring Devices and Transducers

There are numerous factors to consider in deciding how vibration condition monitoring might be implemented with rotating machinery. Criticality of equipment, in service access, temperatures, safety considerations, may determine the measuring device used. These factors as well as machine design and expected failure modes will determine the sensor that will be used.

Transducers sensing vibration convert mechanical energy to an electrical signal for for digitizing and evaluation. These transducers include proximity or eddy current probes for measuring displacement, velocimeters for measuring velocity, accelerometers measuring acceleration, lasers, acoustic emissions transducers and microphones. The output of these transducers is usually transferred via cable or wireless connection to a hand held recording device or to a permanently installed system. These devices can display a live readout of the reading sensed, record trends and or store the data in a predefined format eg. waveforms, spectra, RMS or position readings from proximity sensors. Permanently installed systems can be set up to trigger alarms, control processes or shut down machinery based on predefined or custom configured settings. Shut down of equipment can be based on one reading alone or based on multiple input conditions such as vibration, speed, temperature and pressure. There are many vibration measuring devices available from easy to use hand held units that give a readout of machine condition, hand held units for complex diagnosis

with waveform and spectral analysis capability to permanently installed systems set up to constantly monitor machine vibration. The following are some examples of available measuring devices and transducers.

Hand Held Vibration Meters

This SKF hand held meter gives an overall velocity vibration level readout and it also takes a higher frequency filtered reading aimed at bearing health. The unit gives a readout based on ISO 10816-3. It also takes an infra-red temperature reading. It operates at a velocity range from 0.7 to 65 mm/s RMS. It weighs 125 grams and has a 10hr battery life.

Figure 2.14: This shows the SKF hand held CMAS 100-SL vibration meter. (SKF 2014)

Portable Vibration Analyzers

The portable GE Commtest vb5, vb6, vb7 and vb8 vibration analyzer units offers from 1-4 channels for simultaneous data collection. The units have a 95db dynamic range. They offer demodulation for a filtered high frequency reading. They offer up to 6400 or 12800 lines of resolution depending on the unit purchased. FMax for a standard unit is 40 kHz with the option for upgrade to 80kHz. Windowing options offered are Hanning and rectangular (none). Battery life is quoted as 10hrs. The unit weighs 1.2kg.

Figure 2.15: This shows the portable GE Commtest vb8 Vibration Analyzer. (GE Commtest 2015)

Emmerson Process Managements CSI 2140 is a portable vibration analyzer offering up to 4 channel simultaneous data collection. It has a 120db dynamic range. Analogue to digital converter is 24 bit. They offer demodulation and Peakvue as options for obtaining filtered high frequency data. The unit give up to 12800 lines of resolution. The True Zoom option provides an effective 300,000 lines of resolution. FMax is 80kHz maximum. Windowing options are Hanning or rectangular (none). Battery life is quoted as 10hrs. The unit weighs 1.79 kg. Hardware and software available for balancing, laser alignment.

Figure 2.16: This shows the portable Emmerson CSI 2140 Machinery Health Analyzer. (Emmerson 2015)

Permanently Installed Systems

There is also a variety of permanently installed continuous monitoring systems available and the Allen Bradley XM-120 and XM-121 modules are just one example of such systems. They come in a two channel module that can be run independently or fitted to in a bank feeding data back to either other XM modules, programmable logic controllers, a distributed control system or a condition monitoring system. The modules will accept accept any voltage output measurement device for vibration, pressure, temperature or tachometer. Suitable for order analysis or variable speed monitoring. The units operate with a 24 bit analogue to digital converter and a dynamic range of 90db. Offers 100, 200, 400 and 800 lines of resolution. Up to 16 alarms can be set per module on any measured parameter with logic operators of greater than, less than, inside range and outside range. These units may not give as much resolution as portable analyzers but they are a very powerful tool able to be set up to monitor equipment around the clock and ensure it operates within a tolerance that be accurately defined in terms of frequency bands and levels.

Figure 2.17: This shows an Allen Bradley Rockwell XM-121 module. (Rockwell 2015)

Proximity Sensors

Inductive proximity sensors used for vibration readings also known as eddy current sensors come as a unit with the sensor and an oscillator/demodulator to drive and read the sensor. A radio frequency signal is transmitted by cable to the probe and radiates from the probe tip. This generates eddy currents in the target/shaft. These eddy currents in the target affect the output of the oscillator demodulator and this is output is compared

to a reference value. The AC component of the signal is proportional to vibration and the rectified DC signal is proportional to the air gap. These probes are sensitive to the resistivity and magnetic permeability of the the target material. The probe and systems must be suitable type for the target material even down to the type and quantity of alloys in a steel shaft. They are sensitive to shaft run-out and need a smooth ground or polished area three times wider than the probe to read off. They are often used on babbitt/journal bearings and thrust arrangements and can be mounted on brackets external to the bearings or internally through holes in the bearings themselves. On a shaft for radial measurements there is often two mounted at 90 degrees at a bearing. They are also sensitive to residual magnetic fields and shafts need to be degaussed.

Velocity Transducers

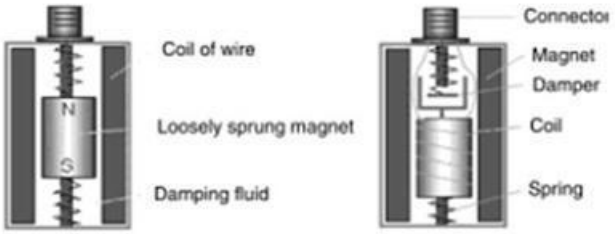


Figure 2.18: This shows two types of velocity transducers.(Scheffer 2004, fig. 3.1)

Velocity meter transducers work on the principle of having a magnet move relative to a wound coil. The moving component, whether magnet or coil is lightly sprung and has a damper to stop free oscillation. It is manufactured to give an output voltage proportional to the speed of the movement of the machine it is mounted on. They typically have a linear response over a frequency range of $10Hz - 1kHz$ though models are available for low frequency applications down to $1Hz$. Their sensitivity is often given as $mV/mm/s$. A general purpose sensor is of a sensitivity between $20 - 30mV/mm/s$ (Scheffer 2004) with sensors ranging from 4-50 mV/mm/s. They are sensitive to influence from external magnetic fields which can affect sensor output signal. A shield can be used to protect the sense coil from an external magnetic field. Ideally mounted by threaded stud or clamped in position. Can be held in position. They don not require external power. They have moving parts and can fatigue and require routine calibration annually to inspect function over their measurable frequency range. Two different types are shown in shown in figure

2.18 on page 39.

Accelerometers

Machines with rolling element bearings and gears "should be monitored with accelerometers" (Scheffer 2004 p.9). This of course can depend on the speed of a shaft. As the name accelerometer implies they measure acceleration by having a mass attached to a piezoelectric crystal. The mass and crystal are matched to give a calibrated charge from the crystal when the mass exerts a force on it that is proportional to the acceleration of the mass/unit. This is demonstrated by newtons second law, equation 2.2 page 20.

The charge output from the crystal measured in pico Coulombs per g of acceleration (pC/g) where $1g$ is $9.8m/s^2$ acceleration. These sensors are generally a sealed unit and cannot be calibrated. A charge amplifier converts the charge to a proportional voltage output for computation as a vibration level. The amplifiers are typically internal and known as integrated electronic piezoelectric (IEPE) accelerometers. They have no moving parts. Accelerometer sensitivities are given as mV/g. Common systems require a constant current source or a regulated voltage source. It uses a two wire system, one for power/signal and the other a common. The circuitry of internal amplifiers is susceptible to the effects of heat. Cable lengths beyond 150m may begin to reduce the frequency response range. Mounting accelerometers is best done by screwing them to the machine. This gives the best high frequency response of all mounting methods. As the stiffness in the mounting is reduces so is the mounting resonant frequency, affecting the upper frequency response of the accelerometer to the vibration present. The sensors can also be glued to the machine for permanent installation. Portable measurements with magnet mounted sensors is a common method as the initial cost of permanent installations across large sites is high. Holding the accelerometer on the machine with the use of a stinger can also be used, but care must be taken to hold it secure and still to ensure the best high frequency response and that the low frequency aspect of the signal is stable. Stingers can be shaped specifically to reach between the air cooling fins on an electric motor in an effort to access the best transmission path for higher frequency defect vibration. There are even wooden shaft riders that can be attached to accelerometers to measure the vibration of the shaft directly. The general purpose accelerometer of $100mV/g$ sensitivity has a reliable frequency range of $1-10kHz$ beyond which the response is outside an acceptably

linear range. Accelerometers are also available with wireless transmitters.

The Commtest Ranger system combines the an accelerometer and wireless transmitter for harder to access areas and battery life is advertised as 18 months when used as recommended.

SKF also has the 'Micro-Vibe', a wireless transmitting 20mV/g accelerometer for use with android devices designed as a portable device with a batter life of 5hrs with two 'AAA' batteries. Data allows for up to 12800 LOR with an acceleration FMax of 30kHz. The software caters for Hanning, rectangular and flat top windows. Data can be uploaded then to a base PC loaded with software for detailed analysis.

Figure 2.19: This image shows the 20mV/g, 'AAA' powered SKF wireless transmitting accelerometer the Micro-Vibe. (SKF 2014)

Laser Transducers

Often uses a neon-helium laser source with a frequency of $4.74 \times 10^{14} Hz$. Inside the instrument the laser is split into two beams. One of these beams is a reference beam and the other is pointed at the object to be measured. The laser pointed at the object is reflected back to the instrument and is changed in frequency and phase by the objects vibration, the Doppler shift effect. The instrument compares the reflected laser light to the reference beam for a vibration readout. Good for high temperature application where it is too hot for other sensors. Also good for use where the mass vibrating is small. Can be set up to measure one, two or three dimensional movements and also to measure torsional vibration. Used on structures, bridges, wind mill gearboxes, gas turbines, delicates membrane vibrations. Frequency response ranges up to $100kHz$.

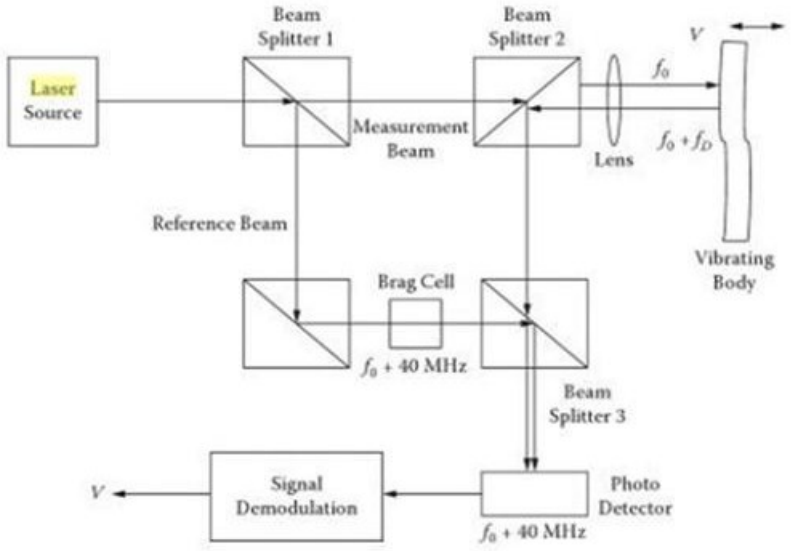


Figure 2.20: This shows the layout of a laser vibration meter (Mohanty 2014)

Acoustic Transducers

Common frequency range is from as low as $20kHz$ up to $2MHz$. Measured with a parabolic microphone , they are designed to detect a very high frequency stress wave generated when a cracks grow under load and when short time impacts such as defects are encountered. Because the frequencies measured are so high there is a lot of machine noise that will not be present as that is predominantly in the lower frequency ranges below $20kHz$. An example of use is setting up the equipment beside a rail line, with the microphone pointed at the train axles as they pass the instrument detects the noise bearing defects generate.

Ultrasonic Analysis

Generally covers a sound range from $20kHz$ to $100kHz$. The ultrasound instrument detects sound frequencies above the human hearing range. The instruments is commonly placed in contact with the machine close to the bearing. A transducer can also be used to turn airborne ultrasonic signal into an electrical signal for measurement. The impacts excite the machines higher natural frequencies and are detected by the instrument. An overall level can be read or the signal can be demodulated and turned into a spectrum

for frequency analysis. Has many applications from steam leaks, valve leaks, pressure vessel/pipe leakage, vacuum leaks. Can monitor corona generated in electrical switch gear and static discharge events. Basic instruments used underground for inspection of conveyor rollers where background noise is high and there are many rollers present. Also used in measuring lower frequencies where an ultrasonic sender is mounted to a machine and a receiver measures and translates how the signal is distorted by Doppler shift.

2.8.16 Vibration Analysis Techniques

A rotating machine produces varying degrees of overall vibration and that may be spread over a wide range of frequencies. The art in vibration monitoring is to know where to find the indications of problems, that is to target the specific type of indications that will alert one to an impending problem or fault condition that will reduce the life of the machine. The following techniques have been developed with specific aspects of machine defects and all are applicable to rotating machinery and a range of defects.

Broad Band Vibration Analysis

A meter records the overall signal from the sensor and gives an RMS reading, the root mean square. Frequencies monitored can vary depending on the transducers response range and meter settings.

Octave Band Analysis

The vibration spectrum is divided into octaves or fractional octave bands. An RMS level is reported for each frequency band. A machines condition can be interpreted by comparison to preset levels, by trending with historical date or by an experienced technician. This is an example of constant percentage bandwidth.

Constant Band Width Analysis

The signal from a sensor is subjected to a constant bandwidth filter over the frequency range being measured, that is the same sample rate is used at the low end of the frequency

spectrum as is used at the higher end.

Constant Percentage Bandwidth

As the name implies the bandwidth filter is a constant percentage of the frequency range being targeted.

Real Time Analysis

A signal is recorded on computer and played back through a real time analyzer. The signal is transformed into the frequency domain. A constant bandwidth spectrum is produce at a resolution dependent on the sample rate of the recorded time waveform. Varying resolutions and frequency bands can be selected, listened to or analyzed as waveform or spectrum. Suited to short duration, transient vibration for analysis though may record a long waveform limited by memory. Multi channel units recording data simultaneously can aid in pinpointing spurious sources and in phase readings.

Time waveform Analysis

Used to be done with an oscilloscope. Good representation of transients like gear teeth impacting, slow beat frequencies, amplitude modulation, imbalance, misalignment and looseness. Use of a data collector with screen will show these vibrations in real time and data can be stored for further analysis. Filters can be applied during collection or in analysis software.

Time Synchronous Averaging

This technique requires a tachometer to tell the analyzer unit when to begin the data sample. Samples of vibration from each successive rotations of the shaft are added together and an average waveform is produced, This has the effect of removing non-synchronous vibration, whether from bearing, transients, random noise, vibration from other shafts or machines. Can be used to analyze specific gears and condition of each tooth. Can allow for precise location of defects for inspection on gears. Can be transformed into spectrum

or circular plots for visual representation.

Frequency Analysis

Typically uses an accelerometer and a hand held vibration collector. The data is collected with preset parameters and transferred to a computer where a software package is used to interpret the data. Often the waveform and spectrum are stored in the data collector. Filters can be applied by hardware before storing data or by post processing with software. Software packages allow changing units, trending, waterfall plots. A baseline reading is of a machine in good condition for future reference is beneficial. A change of one standard deviation over the baseline level for any forcing frequency can indicate a machine problem.

Cross Correlation

This is a comparison of two time waveforms and gives a measure of the degree of repeatability with 0 being no repeatability and 1 being the highest degree of repeatability

Autocorrelation

Similar to cross correlation this is a comparison of a time waveform with itself to determine what is repetitive in the waveform.

Cepstrum

A Cepstrum is the FFT of a logarithmic spectrum. It is used to separate the harmonics and sidebands that are present in a spectrum, allowing them to individually be trended. It lifts up the lower level indications making them easier to distinguish. Can also be used to filter out an echo or reflected vibration.

Amplitude Demodulation

First the signal goes through a high pass filter to remove low frequency vibrations. The signal is then demodulated by removing the higher frequency carrier component of the

wave. A low pass filter is applied to remove other unwanted high frequency noise. A resultant waveform can be converted to a spectrum. Moubray (1997) mentions it is difficult to use for detecting slow speed bearing faults because when the signal is subject to the low pass filter a portion of the short duration stress wave from the impact is filtered out and decreases likelihood of detection. Signals can be filtered by the instrument circuitry before being stored or a raw signal can be post processed by software.

Peak Value (PeakVue) Analysis

Separates the stress wave vibration from the signal by use of a high pass filter. The signal amplitude and high frequency shock pulse width is then enhanced before it is subject to the low pass filter. This means more of the original stress wave signal remains making it more sensitive than amplitude demodulation.

Spike Energy

The signal passes through a high pass filter. The remaining signal passes through a peak to peak detector. A time delay factor is used to envelope the signal decay after the impact and the peak to peak value is reported in 'gSE' units (acceleration waveform peak to peak value as $g's$). This can be subject to the FFT process to give the Spike Energy Spectrum

Proximity Analysis

Uses a displacement sensor to record position in time. Can be used to measure displacement and change in displacement vs time. Often used in applications with white metal bearings in an $x - y$ configuration and on critical thrust arrangements. Ideal for measurement of misalignment, oil whirl, imbalance, bent shafts. System often set up to trip machines on excess displacements and velocity levels.

Shock Pulse Monitoring

Detects the mechanical shock wave from impacting using a special layered piezoelectric accelerometer tuned to a mechanical and electrical resonance of $32kHz$. The shock pulse

or pressure wave excites the resonance in the transducer. For a given defect in a machine the amplitude of the sensor signal is directly proportional the impact velocity. Not affected significantly by background noise as it targets such a high frequency range. Was developed for monitoring rolling element bearings and is successful at detecting inadequate lubrication. Will detect other metal to metal impacting where a stress wave is generated.

Kurtosis

Looks at shock pulse related vibration in the higher frequency ranges by filtering out lower frequencies. A frequency band for analysis can be set and recorded by the instrument or a raw signal can be recorded and processed at a later time. It is a statistical analysis of the time domain signal. As the impact from a defect increases the amplitude difference from the waveform mean level increases and statistically there is greater variance and a higher peak in the distribution curve. A normal distribution has a Kurtosis (K) of 3 and as defect become evident K increases. Is very sensitive to noise from other impact sources. As a defect becomes larger and the time signal noisier the K value will decrease, so it can be a good early fault indicator.

2.8.17 Basic Faults

Vibration analysis is used to detect faults in machines and it is the analysts job to recognize when there is a fault present and determine what it is. Faults broadly fall into two categories;

- Synchronous faults covers unbalance, misalignment, looseness, gear damage, pump vane pass vibration and any vibration or fault that is a whole integer multiple of shaft speed, eg 1,2,3,4,5....$orders$
- Non-Synchronous faults covers the bearing defect vibrations, v-belt pass vibration and faults whose vibrations occur not at whole number multiples of run speed.

Unbalance

This vibration is always present at some level. It is recognizable in the time waveform as a sinusoid at the rotational rate of the shaft, and in the spectrum as a peak at the frequency of shaft rotation. It can be modeled a mass that rotates around the center of the shaft causing a periodic centrifugal force given by

$$f(t) = mr\omega^2 sin(\omega t) \tag{2.25}$$

where m is the out of balance mass, r is the radius of unbalance mass, ω is the forcing frequency in rad/s.

Misalignment

Radially this can be seen more in the velocity or displacement waveforms and gives a M-W pattern to the wave. In the spectrum this gives a strong peak at twice the frequency of the shaft speed, ie at $2orders$. This is strongest with solid couplings and is due to radial loading of the shafts each side of the coupling. Because the shafts are loaded 180° out of phase when they are coupled out of alignment, when they rotate the misalignment force is present, once from each shaft per revolution, twice per revolution. Axial misalignment shows as a run speed vibration in readings taken axially. It will bee present as a sinusoid in the waveform and as a peak at run speed frequency or $1order$ in the spectrum.

Looseness

When as shaft has excess radial or axial clearance it is not constrained to run in position as designed. The excess clearance allows it to move or slap around and this action normally occurs at the rotational rate of the shaft. This often causes an impact to some degree and this impact in the ideal case will be seen as a spikes in the time waveform spaced at the rotational period. In the frequency spectrum this will be present as multiple harmonics of the run speed frequency.

Bearing Damage

A bearings can display the following fault frequencies;

- Outer Race Defect Frequency ($ORDF$)

$$ORDF = f\frac{n}{2}\left(1 + \frac{\oslash_b}{\oslash_p} cos\Theta\right) \tag{2.26}$$

- Inner Race Defect Frequency ($IRDF$)

$$IRDF = f\frac{n}{2}\left(1 - \frac{\oslash_b}{\oslash_p} cos\Theta\right) \tag{2.27}$$

- Ball Spin Defect Frequency ($BSDF$)

$$BSDF = f\frac{\oslash_p}{2\oslash_b}\left[1 - \left(\frac{\oslash_b}{\oslash_p} cos\Theta\right)^2\right] \tag{2.28}$$

- Fundamental Train Defect Frequency ($FTDF$)

$$FTDF = \frac{f}{2}\left(1 - \frac{\oslash_b}{\oslash_p} cos\Theta\right) \tag{2.29}$$

where f is the shaft speed in Hz, n is the number of balls/rollers in the bearing, $\oslash_b$ is the ball/roller diameter, $\oslash_p$ is the ball/roller pitch diameter and Θ is the contact angle.

Bearing damage such as spalling occurs in the time domain as an impact and due to the geometry of a bearing these defects are non synchronous vibrations, eg an $ORDF$ may occur at $5.6 orders$, that means that there will be 5.6 impacts per revolution of the shaft from rollers hitting the defect in the outer race.

2.8.18 Corrective Actions

It is the nature of a machine to have natural frequencies. A machine should be designed not to have natural frequencies near inherent forcing frequencies. The problem arises when natural frequencies coincide with forcing frequencies. For an in service machine with this condition the mass or the stiffness can be changed to shift the natural frequency away from the forcing frequency. A different operational speed can be used to move the forcing frequency away from the natural frequency. The natural frequency of a structure can also change if it is weakened by damage, cracking or loose components so inspection

of a structure may be warranted. Modification of a machines operational speed, load, mass or structure may also cause undesirable resonance.

Correcting a case of unbalance requires a process of balancing. This is often not a straight forward process and a component can become out of balance for numerous reasons. In the case of fans dirt will often build up which may happen unevenly, fan blades may crack or erosion can occur causing unbalance. Some indications of looseness or an inadequately stiff structure can also give the same indication of unbalance. To confirm the cause of run speed vibration bump tests should be conducted to eliminate natural frequencies as a cause of elevated vibration. Testing phase between components is also recommended to eliminate looseness as a cause of elevated run speed vibration and to confirm that the phase shift is consistent with that common to unbalance. The simplest form of balancing is static balancing. This is a process of having a shaft on low friction rollers and allowing the heavy area or the shaft to rotate to the bottom under the effect of gravity. A trial weight is added to the shaft and the heavy spot allowed to rotate to the bottom again. Once the right amount of weight determined it can it can be fixed to the shaft at the correct radius or the same amount removed from a point at 180° on the same radius. Dynamic balancing is a more complex process where the shaft is rotating and measurements of the vibration are taken to determine the location of the unbalance mass on the shaft. Unbalance can further be described as coplanar with the rotating unbalance in one plane or non-coplanar where unbalance masses along a shaft are not rotating in the same plane. If a shaft is dynamically balanced it will also be statically balanced, but a statically balanced shaft will not necessarily be dynamically balanced.

In the case of misalignment shafts need to be aligned. Rigid bolted couplings need to be unbolted and separated allowing them to move independently as the shafts are rotated during the alignment process. Measurements of misalignment can be taken with a sturdily mounted dial indicator. Zero the dial indicator at the top and rotate the two shaft together recording 4 dial readings at 90° intervals checking it come back to the zero reference at the top. It can also be done by turning the shaft with the dial on it and recording the 4 dial positions at 90° intervals, then turn the shaft without the dial 180°, zero the dial then rotate the shaft with the dial on it and recording the 4 dial positions at 90° intervals again. From these readings the horizontal and vertical difference in alignment can be calculated and adjustment made to correct alignment. It is also easy to use todays user friendly laser alignment equipment. Machine dimensions as required by the hand

held computer as required such as distance between mounting feet on a motor and the pump, distance between the mounting feet and the coupling. A laser is connected to one side of the coupling or shaft and a pick up connected to the other. The shafts can be turned together or incrementally allowing the pick up the pass the laser, vise versa, so it can read the change in the position of the laser on the pick up. The computer uses the machines dimensional information and the angles measured by the laser to calculate the change required in packing under machine mounting feet to correct the alignment. The calculated changes in packing are made and the measurements retaken to ensure correction.

A defective bearing requires replacement before deterioration develops threatening a high or unjustifiable chance of failure.

2.8.19 Case Studies

Wind Turbine

Wiggelinkhuizen et al. (2008) outline a condition monitoring program used in the CONMOW wind turbine project using on-line and off-line monitoring systems. The project classed five CM categories and assigned each varying scores based on three desirable aspects, early fault detection, availability of tools/instruments and the potential of the CM technique to reduce maintenance costs. Oil analysis and temperature, acoustic emissions and vibration analysis were used on drive train components. Accelerometers are used in higher speed shafts and displacement sensors on low speed shafts. Time and frequency analysis of electrical power was used to monitor the generator. They used fiber optic and mechanical strain gauge analysis to monitor blade and structure condition. Trending the resonant responses of key components and visual inspections were also used. On this project they report that there was high vibration detected in one turbine due to generator misalignment. Bearing temperatures on the turbines increased with generator output and the bearing temperature is graphed as a function of power output to give them a more stable and accurate trend. Another trend they use is acceleration vibration as a function of wind speed squared and it gives them a straight line trend.

Gear Faults

Using acoustic emissions to monitor gearbox conditions Loutas et al 2008 RMS found RMS levels to vary as much as 33% over time between cold and normal operating temperature due to oil film thickness. AE levels were found to increase with increased gear teeth pitting. One limitation is the shaft riders used in some AE applications, eg for gears fault detection.

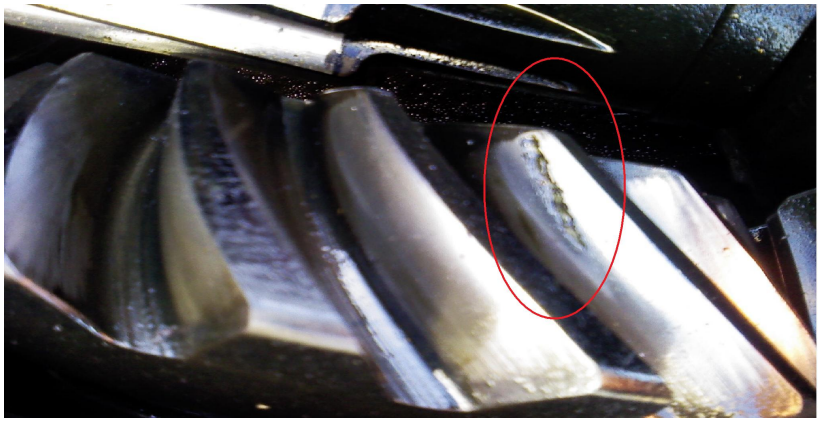


Figure 2.21: This image shows input pinion gear spalling from fatigue. (Bureau Veritas 2008-2015)

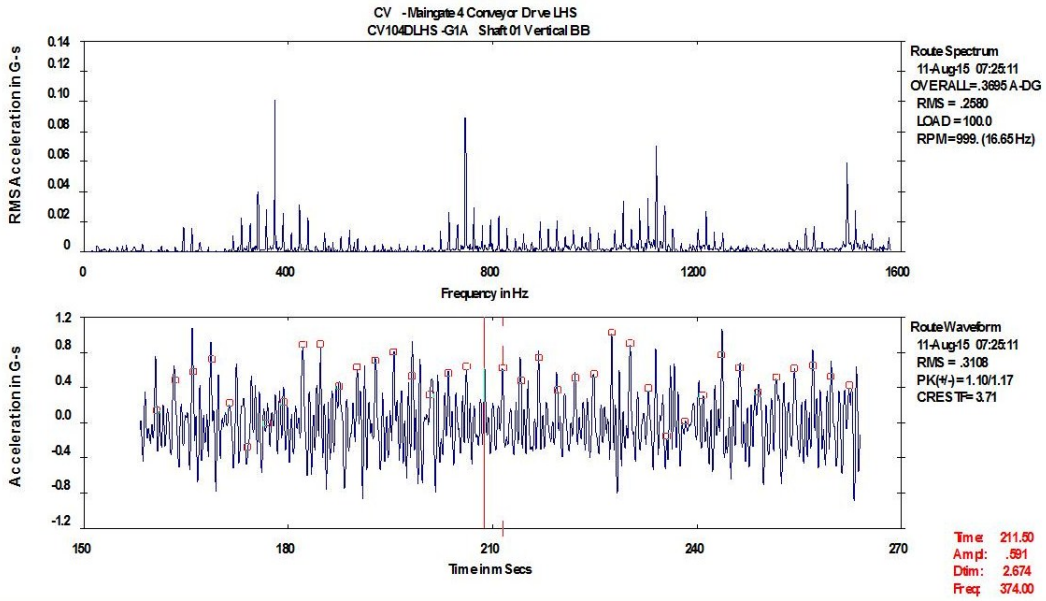


Figure 2.22: This spectrum and waveform show vibration from input pinion damage. The input shaft speed is approximately 16.2Hz, the pinion has 23 teeth giving the defect frequency near 374Hz. (Bureau Veritas 2008-2015)

Diesel Engine

The Cooperative Research Center for Infrastructure and Engineering Asset Management (CRCIEAM) in a study with a diesel engine, Lin et al. (2011), lists approximately 29 modes of failure for a diesel engine that cause symptoms of misfire, low or insufficient power, overheating, excessive mechanical noise, excessive black smoke, excessive white/gray smoke and engine will not start. The modes of failure cover faults with the fuel injection system, the engine valve and timing system, cylinder and piston units, the cooling and electrical systems. Though vibration monitoring has been the preferred technique In the study they show that AE was successful in detecting an induced injector faults in the engine at a full load condition. Data for the induced fault condition shows a higher synchronous average AE RMS level compared to the no fault condition. In separate projects the group has shown AE can be successfully used to fault find a in a slow speed gearbox and in a marine diesel engine with higher signal to noise ratio compared to the more commonly used technique of vibration monitoring making it better suited to applications with high background noise and vibration.

Screw Compressors

The CRCIEAM note that screw compressors are not covered by ISO standard though. They report that a major cause of failure in screw compressors is failure of lubrication, and makes mention how changes in thrust clearance can be critical. This makes AE, vibration and oil monitoring suitable for these machines. Sudden changes of flow and or pressure at inlet and outlet of these machines can also cause or indicate a failure.

Conveyor Pulley Bearing Defect

The following is vibration data for a conveyor pulley defect that has developed with strong impacting that can be felt at the pulley shaft.

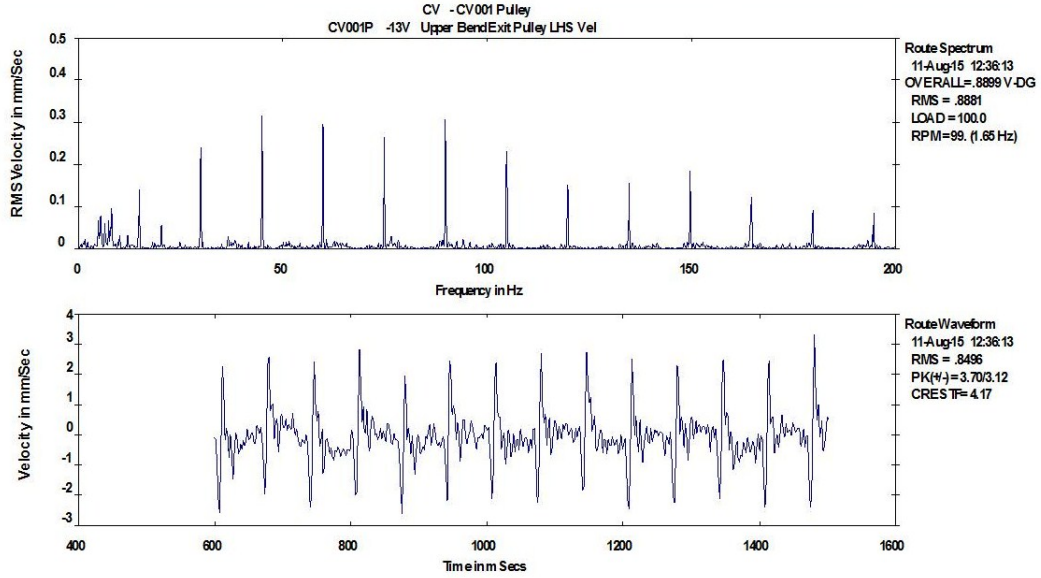


Figure 2.23: This image shows the conveyor bend pulley outer race defect with the defect impact rate of 15Hz. The spectrum and waveform are in velocity units of mm/s. (Bureau Veritas 2008-2015)

Gearbox Bearing Defect

The following image of a damaged bearing inner raceway. The vibration data for this advanced damage, is presented in the time wavefrom with data from a machine in good condition for comparison. See figure 2.25 on page 56

Figure 2.24: This image shows advanced inner race damage on a dragline hoist gearbox input shaft. (Bureau Veritas 2008-2015)

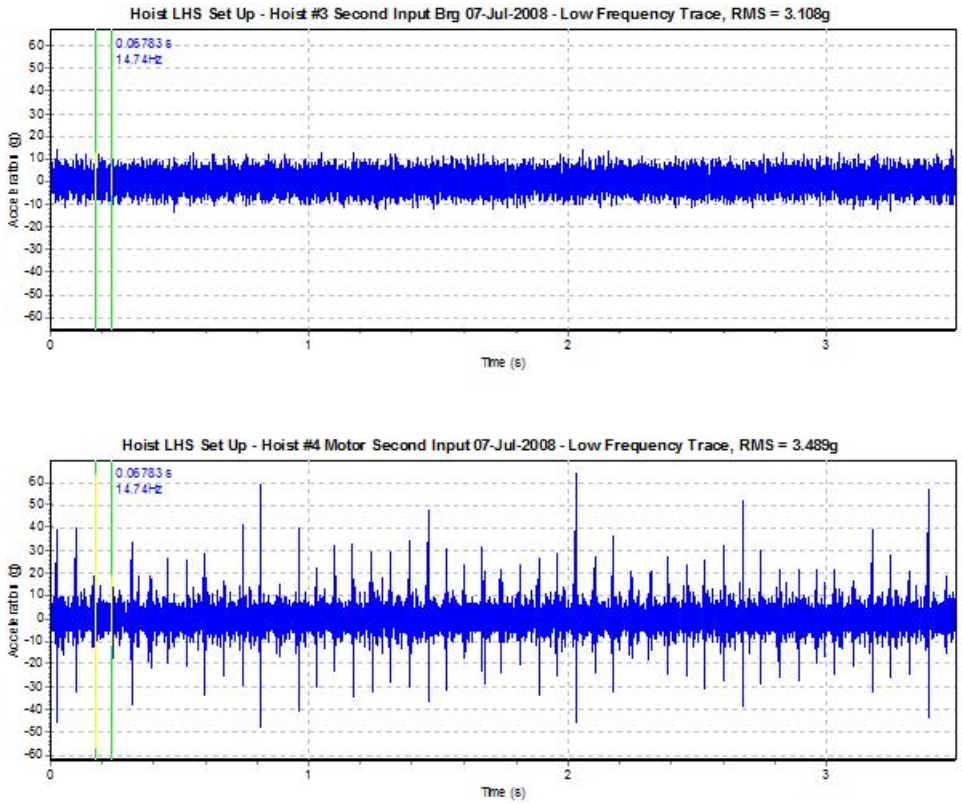


Figure 2.25: This image shows vibration from a healthy gearbox at top compared to the impacting from the advanced inner race damage vibration at the bottom The waveform acceleration units are $G's$. (Bureau Veritas 2008-2015)

2.9 Applications for Condition Monitoring Techniques

There are a wide range of rotating machines of varying complexity for which one or a number of condition monitoring techniques will be suited to detecting defects at their different stages. These rotating machine will use different lubricants, run at different temperatures, have different operating tolerances be of differing sizes and operate with a range of power requirements. Further a single machine may operate with different loads in different phases of production. These will be powered by differing sources, electric motors, steam turbines, gas turbines, internal combustion engines, hydraulic motors, or be an ancillary drive in a train of machines. All of these machines will have failure modes specific to their design and the systems they depend upon to maintain the desired function. Some condition monitoring techniques are more suited to early fault detection while others are suited to detecting symptoms at end of life. Depending on the machine some techniques will detect both early stage faults and end of life conditions, for example increased temperature in a high speed shaft may indicate too much or insufficient lubrication which can lead to premature failure and a temperature rise can also indicate end of life failure. Eddy current sensor for example will not detect an impact shock wave in a machine as it is suited to lower scale frequencies. This variety has lead to techniques being categorized in international standards to machine classes, with reference to size, mounting structure, load and speed. The table 2.2 on page number 58 is similar to table A1 pg 11 of AS ISO 17359:2014 in that it lists the groups of condition effects that are suitable in predicting machine condition.

ISO Standard 7919 covers evaluation of measurement of mechanical vibration on rotating shafts grades steam turbines and generators in excess of 50MW with normal operating speeds of 1500rpm, 1800rpm, 3000rpm, 3600rpm under normal operating conditions, under run up and coast down phases and covers changes that can occur during operation. Levels in this standard cover a broad frequency band and are presented as peak to peak waveform displacement of the shaft and are categorized into four levels:

- A - good,
- B - acceptable
- C - satisfactory for short term operation
- D - being widely accepted as being damaging.

Table 2.2: This table matches the measurable condition effects that are applicable to the tabled machines

Parameter	Electric Motor/Generator	Gas Turbine	Steam Turbine	Turbine Blades	Internal Combustion Engines	Engine Cylinders	Engine Valves	Gearbox	Pump	Roller Bearing	Babbitt Bearing	Hydraulic Systems	Compressor	Shafts	Fans	Belt Drives	Solid Couplings	Fluid Couplings
Dynamic Effects	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
Particle Effects		•	•		•			•	•	•	•	•	•					•
Water Contamination		•	•		•			•	•	•	•	•	•					•
Chemical Effects		•	•		•			•	•	•	•	•	•	•				•
Lubrication Condition		•	•		•			•	•	•	•	•	•					•
Temperature	•	•	•	•	•			•	•	•	•	•	•	•	•	•		•
Physical Effects	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
Electrical Effects	•																	

ISO 10816 part 1 covers general guideline to evaluation of measurement of mechanical vibration by measurement on non rotating part using the same condition assesments A, B , C and D as used as in ISO7919. In addition machines are categorized in four classes:

- Class 1 Machine components eg up to 15kW electric motors
- Class 2 Medium size machines without special foundations, rigidly mounted machines up to 300kW on special foundations
- Class 3 Large prime movers and large machines with rigid, heavy and relatively stiff foundations
- Class 4 Large prime movers and large machines on relatively soft foundations, eg steam turbines with output greater than 10MW

Table 2.3: This table grades the severity of vibration on non rotating parts for classes of machines based on broadband velocity RMS vibration levels (ISO10816-1.

<table>
<tr><th>R.M.S. Vibration Velocity mm/s</th><th>Class 1</th><th>Class 2</th><th>Class 3</th><th>Class 4</th></tr>
<tr><td>0.28</td><td rowspan="3">A</td><td rowspan="3">A</td><td rowspan="5">A</td><td rowspan="6">A</td></tr>
<tr><td>0.45</td></tr>
<tr><td>0.71</td></tr>
<tr><td>1.12</td><td rowspan="2">B</td></tr>
<tr><td>1.8</td><td rowspan="2">B</td></tr>
<tr><td>2.8</td><td rowspan="2">C</td><td rowspan="2">B</td></tr>
<tr><td>4.5</td><td rowspan="2">C</td><td rowspan="2">B</td></tr>
<tr><td>7.1</td><td rowspan="5">D</td><td rowspan="2">C</td></tr>
<tr><td>11.2</td><td rowspan="4">D</td><td rowspan="2">C</td></tr>
<tr><td>18</td><td rowspan="3">D</td></tr>
<tr><td>28</td><td rowspan="2">D</td></tr>
<tr><td>45</td></tr>
</table>

Table 2.4: This table matches dynamic effects monitoring techniques that are suitable for monitoring the condition of the tabled machines based on defects and failure modes they are likely to develop.

Dynamic Techniques	Electric Motor/Generator	Gas Turbine	Steam Turbine	Turbine Blades	Internal Combustion Engines	Engine Cylinders	Engine Valves	Gearbox	Pump	Roller Bearing	Babbitt Bearing	Hydraulic Systems	Compressor	Shafts	Fans	Belt Drives	Solid Couplings	Fluid Couplings
Broadband	•	•	•		•			•	•	•	•	•	•	•	•		•	•
Octave Band	•	•	•		•			•	•	•	•	•	•	•	•	•	•	•
Constant Band Width	•	•	•		•			•	•	•	•	•	•	•	•	•	•	•
Constant Percentage Bandwidth	•	•	•		•			•	•	•	•	•	•	•	•	•	•	•
Real Time Analysis	•	•	•		•			•	•	•	•	•	•	•	•	•	•	•
Time waveform	•	•	•		•			•	•	•	•	•	•	•	•	•	•	•
Time Synchronous Averaging	•	•	•		•			•	•		•	•		•	•		•	•
Frequency Analysis	•	•	•		•			•	•	•	•	•	•	•	•	•	•	•
Cross Correlation	•	•	•		•			•	•	•	•	•	•	•	•	•	•	•
Cepstrum	•				•			•	•	•		•	•		•	•		•
Amplitude Demodulation	•	•	•		•			•	•	•	•	•	•	•		•	•	•
Peak Value	•				•		•	•		•		•	•					•
Spike Energy	•						•	•	•		•	•						•
Proximity Analysis		•	•											•	•		•	
Shock Pulse							•			•								
Kurtosis										•								
Acoustic														•				
Ultrasonic						•	•		•			•	•					

Table 2.5: This table matches the various dynamic effects monitoring techniques and their suitability for detecting the listed defects and failure modes that are common to rotating machines.

Dynamic Techniques	Fatigue	Wear	Cracks	Broken Gear Teeth	Shock	Stress Waves	Imbalance	Bent Shaft	Misalignment	Looseness	Eccentricity	Transients	Cavitation	Inadequate Lubrication	Fluid Flow	Oil Whirl	Resonance	Damaged Drive Belts
Broadband	•	•					•	•	•				•	•	•			
Octave Band	•	•				•	•	•				•	•	•				
Constant Band Width	•	•					•	•	•	•			•	•	•			
Constant Percentage Bandwidth	•	•				•	•	•	•	•			•	•	•			
Real Time Analysis					•													
Time waveform			•	•			•	•	•	•	•		•					
Time Synchronous Averaging	•	•				•												
Frequency Analysis	•	•					•		•	•			•					•
Cepstrum	•	•		•														•
Amplitude Demodulation	•	•	•	•						•								
PeakVue	•	•	•	•			•			•								
Spike Energy	•	•		•			•			•			•	•	•			
Proximity Analysis								•	•	•		•				•	•	
Shock Pulse	•	•				•								•				
Kurtosis	•			•		•												
Acoustic	•		•		•													
Ultrasonic	•		•			•								•				

Table 2.6: This table matches lubrication contamination monitoring techniques to rotating machines based on faults and failure modes to be detected and the lubrication systems these machines typically utilize.

	Electric Motor	Gas Turbine	Steam Turbine	Turbine Blades	Internal Combustion Engines	Gearbox	Pump	Roller Bearing	Babbitt Bearing	Hydraulic Systems	Compressor	Shafts	Fans	Belt Drives	Solid Couplings	Fluid Couplings
Particle Contamination Monitoring																
Ferrography	•	•	•		•	•	•	•	•	•	•	•	•	•	•	•
Analyt Ferrography	•	•	•		•	•	•	•	•	•	•	•	•	•	•	•
DR Ferrography	•	•	•		•	•	•	•	•	•	•	•	•	•	•	•
MO Particle Count	•	•	•		•	•				•	•					•
PB Particle Count	•	•	•		•	•	•	•	•	•	•	•	•	•	•	•
LE Particle Count	•	•	•		•	•	•	•	•	•	•	•	•	•	•	•
LS Particle Count	•	•	•		•	•				•	•					•
Real Time Ferromagnetic Sensor	•	•	•		•	•	•	•	•	•	•	•	•	•	•	•
All Metal Debris Sensor		•														
Graded Filtration	•	•	•		•	•	•	•	•	•	•	•	•	•	•	•
Mag Chip Detection	•	•	•		•	•	•	•	•	•	•	•	•	•	•	•
Blot Testing	•	•	•		•	•	•	•	•	•	•	•	•	•	•	•
Patch Test	•	•	•		•	•	•	•	•	•	•	•	•	•	•	•
Water Contamination Monitoring																
Calc Hydride Test	•	•	•		•	•				•	•					•
K Fischer Titration	•	•	•		•	•				•	•					•
Crackle Test	•	•	•		•	•				•	•					•
Vapor Induced Scintillation	•	•	•		•	•				•	•					•
Clear and brightt	•	•	•		•	•				•	•					•

Table 2.7: This table matches the chemical monitoring techniques used with lubrication of listed rotating machines based on the lubrication systems these machines typically utilize and faults and failure modes to be detected.

Chemical/Lubrication Monitoring	Electric Motor	Gas Turbine	Steam Turbine	Turbine Blades	Internal Combustion Engines	Engine Cylinders	Gearbox	Roller Bearing	Babbitt Bearing	Hydraulic Systems	Compressor	Shafts	Fluid Couplings
Atomic Emission Spectroscopy	•	•	•		•		•			•	•		•
Atomic Absorption Spectroscopy	•	•	•		•		•			•	•		•
X-Ray Fluorescence Spectroscopy	•	•	•		•		•			•	•		•
Fourier Transform IR Spectroscopy	•	•	•		•		•			•	•		•
UV and Visible Absorption Spectroscopy	•	•	•		•		•			•	•		•
Thin Lay Activation				•		•		•	•			•	
Color Indicator Titration	•	•	•		•		•			•	•		•
Total Acid/Base Number	•	•	•		•		•			•	•		•
Exhaust Emission Analysis		•			•								
Viscosity Monitoring	•	•	•		•		•			•	•		•

Chapter 3

Training Program Design

3.1 Design Requirements of Training Unit

The aim is produce a basic program for training and awareness and it will primarily be aimed at skills required for a level 1 technician incorporating some aspects required by level 2 technicians and level 3 technicians. Animation would be beneficial to give an understanding some basics such as simple harmonic motion and superposition of waveform components. Capability to select a machine, its speed, a fault, generate a time waveform, view and manipulate frequency spectra would be desirable in a training unit.

3.1.1 User Requirements

To meet ISO 18436-2 requirements a level 1 technician should have a level of training allowing them to conduct pre-defined, simple single channel vibration condition monitoring activities in accordance with established procedures and under the direction of at least a level 2 technician. Personnel classified to category I shall at least:

- Know of the basic principles of vibration and recognize the different units of measurement
- Be able to collect reliable data ensuring appropriate standards of repeatability
- Be able to identify errors in collected data
- Be able to retrieve pre-defined measurement settings for use with VA equipment and transfer data from an analysis system to a computer-based system
- Be able to compare overall or single-value vibration measurements against pre-established alert settings
- Be able to identify deviations from the norm for single-value vibration values and trends
- Report on visual observations of equipment condition

They shall not be responsible for:

- The choice of sensor, test method or technique or for any analysis or diagnosis to be conducted.

- The assessment of test results, other than identifying conditions against pre-established criteria, such as acceptance, alert, alarm, shutdown, etc.

Further, training course topics to be covered for a level 1 technician are outline in Table A.1 in the standard are Principles of vibration, Data acquisition, Signal processing, Condition monitoring, Fault analysis, Corrective action, Equipment knowledge and Acceptance testing. Under these subject areas Table A.2 in ISO18346-2 outlines further the topics to be covered in training. For a level 1 technician this includes being introduced to simple harmonic motion, the relationship between period and frequency, basic understanding of phase, manage frequency in different units, amplitudes in r.m.s., peak and peak to peak values, units and units conversions, an awareness of the time and frequency domains, an awareness of natural frequencies, resonance, critical speeds and their effect on vibrations. They should also be aware of available transducers powered and not powered, mounted resonance of sensors and general effects of various mounting techniques, upload and download data, be made aware of basic terminology lines of resolution, number of lines FMax time to sample, recognize basic preset faults such as looseness, unbalance, misalignment, bearing damage, be aware of basic maintenance action, alignment, lubrication. Some knowledge of equipment as far as recognizing motors, generators, fans, pumps and a basic understanding of the faults that can be present. Be able to recognize different types of rolling mills, compressors and basic gearbox that can be encountered. Level 1 technicians need to be aware of safety implications related to conducting these tasks. It would beneficial for the user to have good HSC math, trigonometry, simple harmonic motion and superposition and an understanding of some HSC physics such as Newtons laws.

It is desirable that the virtual training unit will meet some ISO18436-2 Level 2 training objectives also. In addition to level 1 knowledge a level 2 trained technicians should be able to conduct machinery vibration measurements and basic vibration analysis of single-channel measurements, with or without phase trigger signals, in accordance with established and recognized procedures. In addition to level 1 knowledge, experience and skills a level 2 technician shall at least:

- Be able to define the measurement activities to be undertaken by a category I individual in the course of routine data collection

- Be aware of and capable of using the basic principles of signal analysis and, as such, can define acquisition and analysis settings to collect data appropriate to the machine(s) monitored
- Be able to perform basic (single-channel) impact tests to determine natural frequencies
- Be able to interpret and evaluate test results from routine analysis and acceptance tests in accordance with specifications and standards
- Be able to diagnose common fault indications and recommend basic corrective actions commensurate with their area of machinery experience including carrying out single-plane balancing of rigid rotors with or without phase
- Be able to provide technical guidance to and instruct category I personnel

A level 2 technician is to have a greater understanding all of the topics listed for level 1 training. In addition to these skills the following are some of the recommended topics for training. An understanding of integration, enveloping, band pass filters, demodulation, crest factor, phase reference, recognize factors of frequency, stiffness, mass, damping and isolation and their interdependence, perform dual channel functions including phase, familiarity with transducer types and mounting, run-out compensation, calibration, triboelectric effects, have a basic understanding of clipping, truncation and leakage, be aware of the Hanning window and it effect on amplitude and frequency, anti aliasing filter, bandwidths, resolutions, filtering and averaging to reduce noise, diagnose fault conditions of unbalance, looseness, misalignment, bearing damage, gear mesh faults, rotor bar and stator faults, drive belt faults and resonances, single plane balancing of rigid rotors and mass estimation, analyze and reporting of routine testing.

Further to these some level 3 concepts may be included in the virtual package if time permits. This level of the virtual package if developed could introduce the user to concepts of:

- The use of various hardware
- Non routine machine investigation
- Complex faults and analysis techniques

- Advanced analysis techniques
- Two plane rigid rotor balancing
- The effect of recommending changes to machinery and its operation

3.1.2 Functional Requirements

The following features are desirable in the virtual package concept:

- There will be a graphical user interface to operate the system.
- The user will choose from levels 1,2 or 3.
- Choosing level 1 to follow a guided step by step approach
- Choosing level 2 to follow a guided step by step approach
- Choosing level 3 allowing use of all GUI options
- Machine components will be selectable, such as a driver, intermediate components such as a lay shafts or belt drives and driven components such as a gearbox, a pump or fan.
- Capability to select a machine, its speed and a fault
- Generate a time waveform
- View and manipulate frequency spectra would be desirable in a training unit.
- Animation would be beneficial to give an understanding of some basics such as simple harmonic motion and superposition of waveform components.

3.1.3 Level 1 Functional Requirements

Principles of Vibration

The user will be given explanations and a waveform and spectral plot, an output from a basic machine with no defects, to represent the following

- Basic Simple Harmonic Motion
- The relationship between period and frequency
- Provide a basic understanding of phase
- Represent frequency in different units
- Amplitudes in r.m.s., peak and peak to peak values
- Show units and units conversions

Data acquisition

Use a simulated data collector on a simulated machine, upload and download data, be made aware of basic terminology, eg. lines of resolution, number of lines, FMax and time to sample,

Signal processing

Provide an awareness of the time and frequency domains.

Dynamic Properties

Provide an awareness of natural frequencies, resonance, critical speeds and their effect on vibrations.

Fault analysis

Provide function to help recognize basic preset faults such as looseness, unbalance, misalignment, bearing damage and be aware of basic corrective actions such as alignment and improving lubrication.

Equipment Knowledge

Improve knowledge of equipment as far as recognizing motors, generators, fans, pumps and a basic understanding of the faults that can be present in these. Facilitate recognition of different types of rolling mills, compressors and basic gearboxes that can be encountered. Level 1 technicians need to be aware of safety implications related to conducting condition monitoring tasks.

Acceptance testing

Make user aware of acceptance testing and provide for assessment of test results against pre-established criteria, such as acceptance, alert, alarm, shutdown etc.

3.1.4 Level 2 Functional Requirements

Functional requirements of the virtual training unit for the level 2 user may include the following;

Signal Processing

Introduce the user to the concepts of integration, the fft, enveloping, band pass filters, demodulation, crest factor, phase reference, anti aliasing filter, bandwidths, resolutions and averaging to reduce noise,

Data Acquisition

Provide the user with the ability to use the basic principles of signal analysis to define appropriate data acquisition settings.

Dynamic Properties

Provide introduction to system properties of stiffness, mass, damping and isolation and their interdependence.

Time Windows

Make the user aware of the Hanning window and it effect on amplitude and frequency, provide a basic understanding of clipping, truncation and leakage.

Fault Analysis

Teach the user to diagnose fault conditions of unbalance, looseness, misalignment, bearing damage, gear mesh faults, rotor bar and stator faults, drive belt faults and resonances, and introduce dual channel functions like relative phase.

Balancing

Provide for learning single plane balancing of rigid rotors. Provide a method to calculate unbalance mass and its angular position.

3.1.5 Level 3 Functional Requirements

Further to these some level 3 concepts may be included in the virtual package if time permits. This level of the virtual package if developed could introduce the user to the following concepts.

Available Hardware

Introduce the ability to virtually fit an accelerometer, velocimeter or proximity probe to a machine.

Complex Fault Analysis

Introduce a mystery fault and provide a toolbox of techniques that can be used to diagnose such as filtering, averaging, coherence, cepstrum, phase, synchronous averaging.

Two Plane Rigid Rotor Balancing

Simulate a rigid shaft supported by two bearings with a randomly positioned mass of a random size and provide a method to determine the axial and angular position of the unbalance mass and allow for its correction.

Machine Modification

Allow user to diagnose a fault and be given options to modify lubrication properties, machine stiffness in two planes, damping properties and component mass. Change the speed of a machine or the load on it to achieve desirable effects. Multiple degree of freedom systems.

3.1.6 Virtual Machine Component Specification

Mounting Structure

Desirable feature of three different structures of varying physical properties allowing for different characteristics and different transfer of vibration between driver, intermediary and driven. Steel reinforced concrete base. Steel plate structure. Separated structures.

Driving Component

Electric motor selected from list of manufacturer and model. Natural frequencies determined by basic 3D modeling or by bump test. Imbalance as a feature and bearings as listed by manufacturer to simulate faults. Desirable electrical related vibrations such as pole pass frequency and rotor bar looseness.

Coupling selection

Desirable to be able to select from solid couplings to one with a degree of flexibility. This will allow for modeling a variance in misalignment effect on vibration.

Gearbox

Desirable component for vibration simulation. Would be a single reduction. Bearings included for fault generation. Vibration associated with gear eccentricity desirable. Variance in levels of gear deterioration desirable. Imbalance desirable.

Pump

Include selectable pumps for generation of bearing, imbalance, looseness and vane pass faults. Displaying cavitation as a fault is a desirable. Natural frequencies of pumps may be determined by basic 3D modeling or by bump test on real pump.

Belt Drive

Allow for modeling of pulley eccentricity and belt pass frequency.

3.1.7 Generated Faults

Bearings

Cage related defects, Ball Spin Defects, Outer Race Defect, Inner Race Defect, Looseness

Balance

Most rotating shafts will have the feature of an off center rotating unbalance mass. This may be in the plane of a bearing or coplanar

Looseness

Will generate a run speed impact. It is desirable this would be by an impact in the waveform. May be by the addition of run speed harmonics.

Misalignment

Will generate an appropriate signal of twice run speed frequency if misalignment is given as an option and chosen.

Random Vibration

May be added to represent random vibrations such as cavitation in a pump, a normal variation in amplitudes, a slight variation in bearing defect frequencies, lubrication problems.

3.1.8 System Output

Visual

- Time waveform
- Frequency Spectrum
- Cursors for identifying peaks in waveform and spectrum
- Images of machines
- Display of worded explanations
- Basic animation to represent simple harmonic motion is desirable

Audible

The user should be able to listen to generated waveforms.

Signal Output

It is desirable that a real data collector could be connected via USB to record a signal.

Tactile

The system could include a method by which the user can feel the generated vibration signal.

System Help

A hard copy manual be available describing the function, operation and specification of the system. This will also be available within the software.

3.2 Programming Software Review

There are many software packages available that could be used to create a virtual vibration training program. These softwares available have embedded functions, an example is Matlab performing a fast Fourier transform by using the command fft(...) and Excel using sin(...) to calculate sine of a value. A graphical user interface will be necessary to run the virtual training program.

Cooperative Research Centres for Infrastructure and Engineering Asset Management (CIEAM)s developed COLOMOS software, a collaborative work by Queensland University of Technology (QUT), Curtin University of Technology (CUT) and the University of Western Australia (UWA). COMOLOS was developed for condition monitoring of low speed machinery that is set up to collect data by on-line remote control, on-line data transfer, on-line live monitoring, off-line analysis and to record trends. It is made with the National Instruments made software LabVIEW. The screen shots of this package with graphical user interface and analysis screens shows some appealing visuals suitable for a training program. The National Instruments LabVIEW web site shows that signals can be made with virtual instruments within the software. LabVIEW could be an option but it is costly with the full version at $1100 and add ons are expensive. A student version is available for a 45 day free trial. The professional version is approximately $7850. This makes it a specialized tool and would limit availability of the training package to either those having the software or training with the version it was created on.

Matlab would give good visual representation of a machine signal and spectra. A machine

could be built in Matlab and/or Simulink to give a degree of realistic data with an output for forced and natural responses. Creating a completely stand alone executable file from Matlab is not possible with the student versions as the "deploytool" comes as an add on for a full licensed version. A 3D animation package is an add on that is available to students though and could give some scope for visualizing machine dynamics. This option could give some scope to future development of a the virtual training training unit as a salable product or adaption into an analysis program.

Microsoft Excel is an option that might be a little more difficult to build a virtual vibration training unit with but it has the benefit that it could be used by anyone with Microsoft. There are some great examples of animation available on the internet enabled by use of Visual Basic Macros in Excel that can be an aid to writing programming. It does not have some of the embedded functions of Matlab/Simulink and would require some checks and balances to ensure that code written in lieu of functions is correct. A trial GUI for the vibration program was partially developed in Excel and extra code had to be written to perform functions such as for the fft, or creating a square pulse wave. Excel would allow anyone to use it, and I have seen some good examples for other applications with graphical user interface and animations that seem very user friendly.

ANSYS16.0 capabilities pdf viewed on-line shows table with topics of vibration as modal, spectrum, harmonic, random vibration, rotor dynamics, super elements and component mode synthesis and mis-tuning all covered by ANSYS Multiphysics, Mechanical and Structural workbenchs. ANSYS will display spectra representing natural modes of a structure, the spectral response of a structure to an impulse and the spectral response of a structure to an input table of forcing frequencies/amplitudes. There appeared to be no waveform representation of vibration in the Mechanical workbench. Waveforms are a feature in the acoustic workbench and for this application they are airborne vibration.

The preferred option was to use the student Matlab package that has been used throughout BE Mechanical course and the Matlab Signal Processing Toolbox and its pre written functions. Matlab student version R2015a has been used to build the program. The program has been run on Windows 10 with Matlab R2015a 64bit and on Windows Vista with Matlab R2015a 32bit software.

3.3 Program Design and Development

The following covers concepts written into and used by the program when the user operates GUI controls.

3.3.1 Program Design

The concepts to be covered as GUI's are defined by the user requirements and have been combined in a flow chart showing connection of the program elements has been developed to map a structured approach to software development.

The opening GIU, 'Level Selection', allows the user to choose either Level 1, 2 or 3.

The GUI concepts and sequence for the Level 1 section of the program is the following.

Level 1

- Display and explanation of basic simple harmonic motion.
- Display and explain time and frequency domains.
- Display and explain the relationship between frequency and period
- Display and explain the concept of phase.
- Display and explain the concept of frequency in units of cycles per minute and orders of shaft speed.
- Display and explain measures of amplitude in RMS, peak, peak to peak.
- Display and explain units and conversions.
- Display and explain the concepts of lines of resolution, number of lines, frequency maximum and time to sample.
- Display and explain natural frequencies, resonance, critical speed and their effects on vibration.
- Display and explain basic faults.
- Explain possible corrective actions.

- Equipment knowledge.
- Give acceptance tests.
- Give a test bench for experimentation with these concepts.

Levels 2 and 3 have been built in a similar fashion from the system requirements and the following flow chart for the program was created.

Numbering of GUIs for Level 1 as , Level_1_1, Level_1_2 etc, and Level 2 in the same fashion.

3.3.2 Generating Data

Basic Concepts

For initial introduction to basic concepts some program GUI's are written to generate a purely sinusoidal wave or combinations of them that are graphed as sinusoid or superimposed for graphing.

Single Degree of Freedom System

The model used most in this program to simulate vibration from a machine is that of a single degree of freedom system with linear stiffness and viscous damping whose motion is defined by the following differential equation. For simulation initial stiffness and damping values were take from the Moazenahmadi (2015) and changed slightly to give a desired response to try and replicate real machine data, some of which is displayed in Appendix B as supporting information.

$$m\ddot{y} + c\dot{y} + ky = f(t) \tag{3.1}$$

where m is the total mass of the system and $f(t)$ is the input force.

Putting this into Cauchy form with $x_1 = y$, $x_2 = \dot{y}$ and this gives two equations

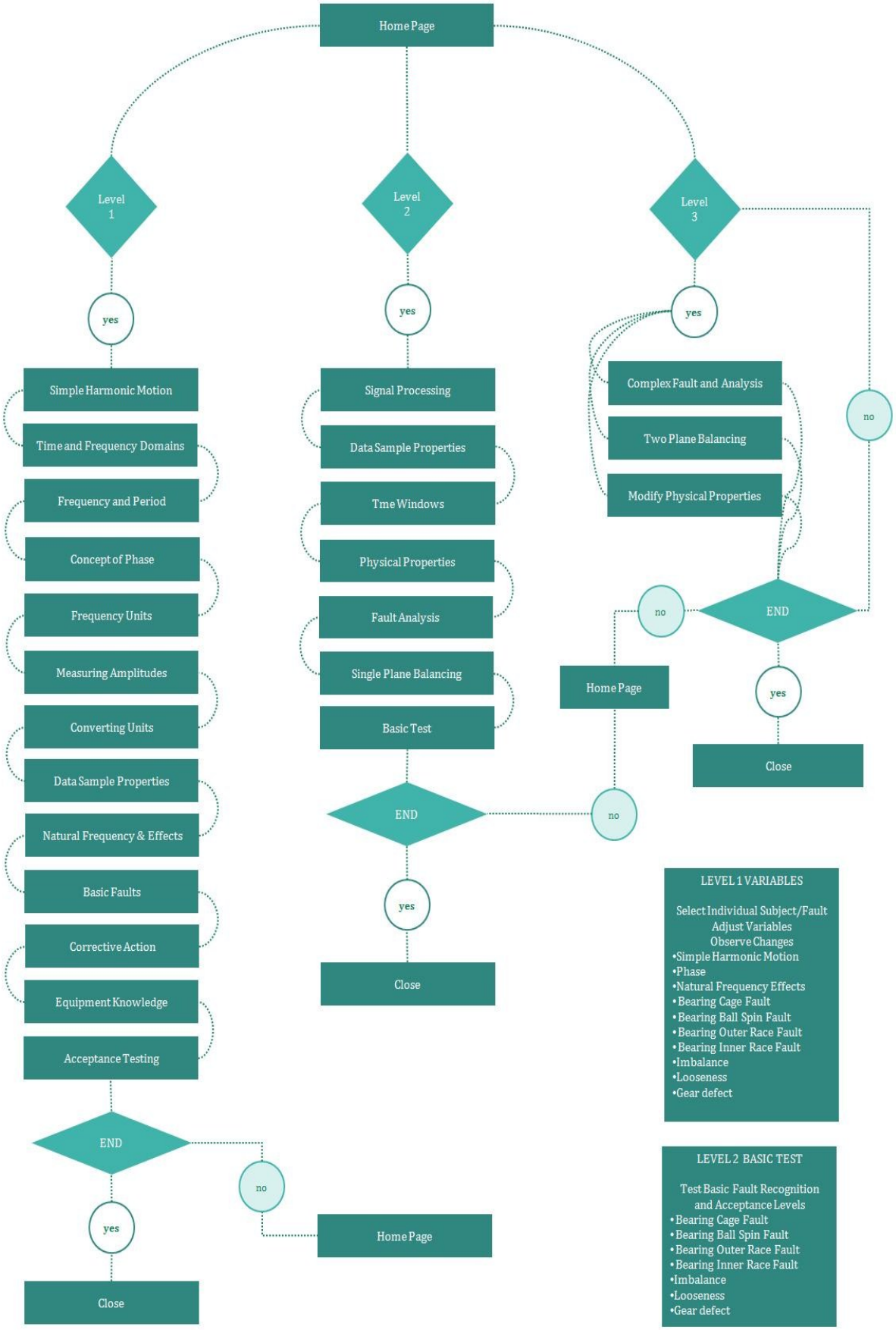


Figure 3.1: This figure shows the software design layout in a flow chart.

$$\dot{x}_1 = x_2 \tag{3.2}$$

$$\dot{x}_2 = \frac{1}{m} f(t) - \frac{k}{m} x_1 - \frac{c}{m} x_2 \tag{3.3}$$

This can be written as the matrix equation

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} f(t) \tag{3.4}$$

This is in the form of

$$\dot{\mathrm{x}} = \mathrm{A}\mathrm{x} + \mathrm{B} f(t) \tag{3.5}$$

where

$$\mathrm{A} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \tag{3.6}$$

$$\mathrm{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{3.7}$$

and

$$\mathrm{B} = \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} \tag{3.8}$$

Two Degree of Freedom System

The model used in the Level_2_4 section of the program is a two degree of freedom system with linear stiffness and damping. The motion of the system is defined by the following equations

$$m_1 \ddot{y}_1 + c_1(\dot{y}_1 - \dot{y}_2) + k_1(y_1 - y_2) = f(t) \tag{3.9}$$

$$m_2 \ddot{y}_2 + (c_1 + c_2)\dot{y}_2 - c_1 \dot{y}_1 + (k_1 + k_2) y_2 - k_1 y_1 = 0 \tag{3.10}$$

This can be written as the matrix form as

$$[M]\begin{bmatrix}\ddot{y}_1\\ \ddot{y}_2\end{bmatrix}+[C]\begin{bmatrix}\dot{y}_1\\ \dot{y}_2\end{bmatrix}+[K]\begin{bmatrix}y_1\\ y_2\end{bmatrix}=[F] \tag{3.11}$$

Where M is the inertia matrix, C is the damping matrix, K is the stiffness matrix and F is the input force matrix.

$$[M]=\begin{bmatrix}m_1 & 0\\ 0 & m_2\end{bmatrix} \tag{3.12}$$

$$[C]=\begin{bmatrix}c_1 & -c_1\\ -c_1 & c_1+c_2\end{bmatrix} \tag{3.13}$$

$$[K]=\begin{bmatrix}k_1 & -k_1\\ -k_1 & k_1+k_2\end{bmatrix} \tag{3.14}$$

$$[K]=\begin{bmatrix}k_1 & -k_1\\ -k_1 & k_1+k_2\end{bmatrix} \tag{3.15}$$

$$[F]=\begin{bmatrix}f(t)\\ 0\end{bmatrix} \tag{3.16}$$

Balachandran (2009) lays this out in the state space formulation for a multiple degree of freedom system

$$\left[\dot{\mathrm{X}}\right]=[\mathrm{A}]\,[\mathrm{X}]+[\mathrm{B}]\,[\mathrm{F}] \tag{3.17}$$

where

$$[\mathrm{X}]=\begin{bmatrix}X_1\\ X_2\end{bmatrix}=\begin{bmatrix}x_1\\ x_2\\ \dot{x}_1\\ \dot{x}_2\end{bmatrix} \tag{3.18}$$

and

$$\left[\dot{\mathrm{X}}\right] = \begin{bmatrix} \dot{X}_1 \\ \dot{X}_2 \end{bmatrix} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \ddot{x}_1 \\ \ddot{x}_2 \end{bmatrix} \tag{3.19}$$

$$[\mathrm{A}] = \begin{bmatrix} [0] & [I] \\ -[M]^{-1}[K] & -[M]^{-1}[C] \end{bmatrix} \tag{3.20}$$

$$[\mathrm{B}] = \begin{bmatrix} [0] \\ [M]^{-1} \end{bmatrix} \tag{3.21}$$

$[A]$ is a $2N \times 2N$ state matrix. $[B]$ is a $2N \times N$ matrix. $[I]$ is an $N \times N$ identity matrix and [0] is and $N \times N$ zero matrix.

3.3.3 Generating Defects

Moazenahmadi (2015) developed a complex multiple degree of freedom model for simulation of a bearing defect. The simulation with the model is an accurate representation when compare to the real system modeled. For a rectangular shaped defect the force calculated in the Moazenahmadi (2015) simulated model differs slightly from a square impulse that will be used in this project model.

Periodic pulse train with variable period and pulse duration is given by the Fourier series. Appendix B Table B eq (h) Magrab 2009 and Example 5.17

$$f(t) = F_o\alpha\left[1 + 2\sum_{i=1}^{\infty}\frac{sin(i\pi\alpha)}{(i\pi\alpha)}cos(2i\pi t/T)\right] \tag{3.22}$$

where F_o is the amplitude of the impulse, $\alpha = t_d/T, t_d$ is the duration of the pulse, T is the period of the impulse.

For this model I am using the Matlab signal toolbox function *rectpuls*. This is suitable for a defect with a constant amplitude such as looseness, an outer race defect, a cage

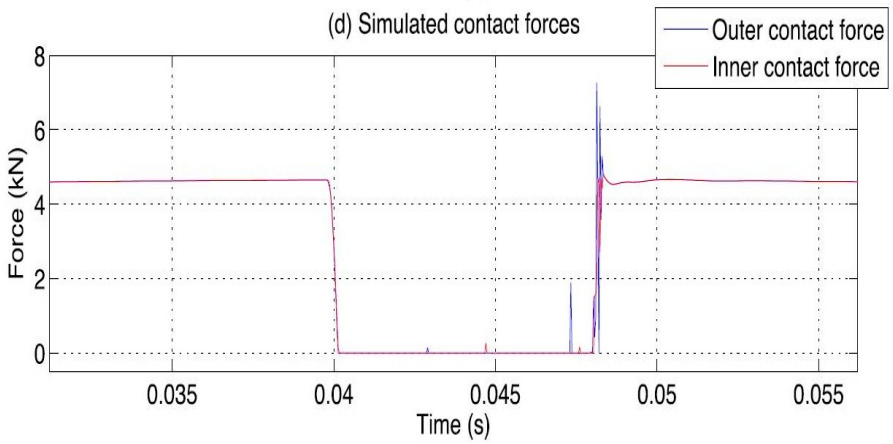


Figure 3.2: This figure shows the force calculated by Moazenahmadi (2015) in the modeling of defective bearings. The duration of the near rectangular pulse is approximately $0.008s$

defect and does not account for vibration with amplitude modulation.

Bearing defects can display amplitude modulation due to a varying amplitude of impulse force F_o either at the rotational rate of the shaft or at the rotational rate of the cage. NSK (2015) covers bearing load distribution and for a roller bearing with radial load only the loaded zone for rollers is less than 180°. The angle of the loaded zone is also dependent on shaft loads and bearing internal clearance. For this program defects with amplitude modulation the model will consider only radial loads and a fixed load zone angle of 155°. The distribution of the load in the load zone will be sinusoidal with a fundamental function having an amplitude from 0 to 1. This is used to modulate the amplitude of the pulsing force, the square wave function to simulate vibration from rotating shaft/inner race defect (or static shaft/outer race defects) and rollers with defects that pass through the bearing load zone. The pulse force is built in the GUI code prior to calling the ODE23 solver and is set up with the peak force to vary with shaft speed to give a response representative of real data. The period of the pulse force is set at $0.05 \times T$ where T is the period between the modeled defect impacts. The centrifugal force is constructed as time progresses in the iterative process of the solver and it is summed with the impulse force for that moment in time extracted from the array in the GUI code.

Machine component selection in GUI's selects information from an array. The array for each machine component is structured in the same order with the spectrum maximum frequency, lines of resolution for the spectrum, the shaft speed for that component, and the bearing defect frequencies for the bearing in the chosen machine component, fundamental

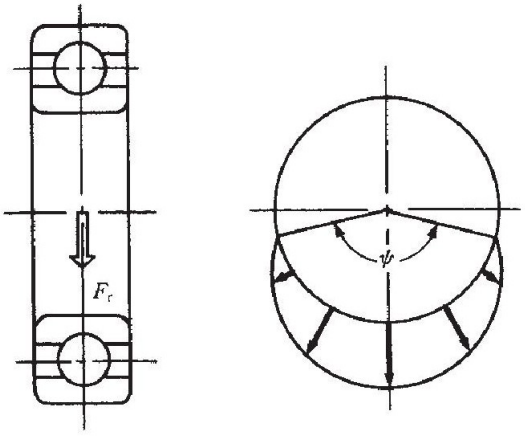


Figure 3.3: This figure shows typical internal bearing load distribution for a bearing with radial load only (NSK 2015)

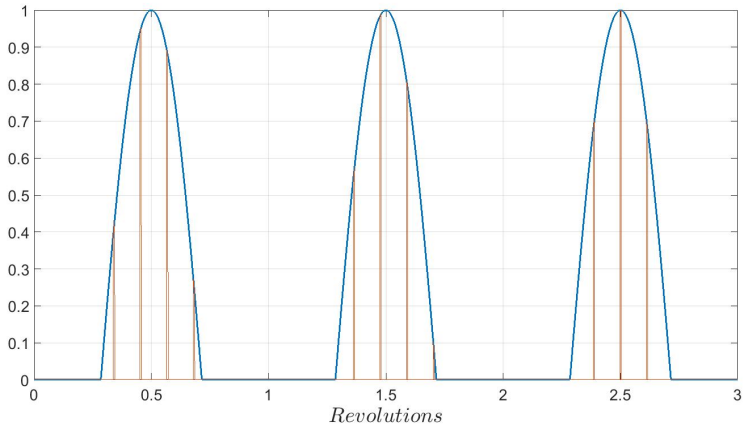


Figure 3.4: This figure shows the effect a sinusoidal load distribution and 155° bearing defect load zone has on a $8.8Hz$ square pulse train, as an example of modulating defect amplitudes from the virtual training program.

train ratio, ball spin ratio, outer race defect ratio and inner race defect ratio, ratios as a function of shaft speed. At the same time an image of the machine component is displayed. There is a variable *mod_switch* in the code for each of the chosen defects that has a value of 0 when there is no amplitude modulation and a value of 1 to apply the modified sinusoid to the defect impulse force. The modulation ratio is taken as the fundamental train ratio for the bearing ball/roller defect or is set as 1 for the rotating shaft inner race defect.

Table 3.1: This table shows the machine components, shaft speeds and bearing details used in bearing defect generation withing the training program. The gearbox 2nd shaft is based on an input speed of 25Hz and a geared speed reduction of 26T/33T. Bearing details FTF, BSF, ORDF and IRDF are in orders as a factor of shaft speed.

Machine Components	Brng Number	Speed (Hz)	FTF	BSF	ORDF	IRDF
Tail Pulley	23040	1.67	0.45	4.74	11.2	13.8
Crusher Shaft	24156	7.05	0.441	4.074	9.711	12.289
GBox 2nd Shaft	32312	19.62	0.4	2.45	6.47	9.53
Motor NDE	6318	25	0.39	2.09	3.09	4.91
Motor DE	NU318	25	0.4	2.43	5.21	7.79

3.3.4 Equipment Knowledge

Being able to recognize machinery such as motors, generators, fans, pumps, rolling mills, compressors gearboxes is required. A basic understanding of the faults that can be present is also required. This is included by presenting images of machinery when choose a machine component and a defect. This is an aside to the function of the popupmenu's that define the input parameter for simulated data collection/generation and resulting system output.

3.3.5 Acceptance Testing

Acceptance testing is conducted to ensure a machine meets a predetermined acceptable level of vibration. This is often done on commissioning a new machine, a reconditioned or repaired machine. Testing at this phase also provides a baseline data sample as a reference to measure future changes by. Use the table from ISO10816-1 to grades the severity of vibration on non rotating parts for classes of machines based on broadband velocity RMS

Figure 3.5: An example of a machine displayed in the Level 1 section of the virtual training software package, a fully enclosed air cooled electric motor driving a centrifugal water pump. The impeller is overhung off one end of the shaft with supporting bearings at the motor side of the pump barrel. The pump inlet is in line with the rotational axis of the pump and the fluid outlet is vertically up (Image (Bureau Veritas 2008-2015))

vibration levels. The table lists four levels of assessment as good, acceptable, satisfactory for short term operation and damaging and categorizes four machine classes. For example a $200kW$ conveyor motor has an RMS velocity level of 2.1mm/s. This makes it a class 2 machine with an acceptable level of vibration. This concept is presented in Level_1_9 and the user tested on it in Level_1_10.

3.4 GUI Design

The GUI's were initially written on paper in terms of concepts to be displayed and layout of controls. Suitable display of text for concepts, equations relating to concepts, graphing of vibration as waveform and or spectrum and controls for the user to interact with the program were sketched for each GUI before using the Matlab Graphical User Interface Design Environment (GUIDE) to place appropriate displays and buttons in a screen setting. As the program developed some concepts were suited to previously made GUIs being modified others required new design. Initial values for sliders and buttons, maximum and minimum values, some colors and the degree to which they are used by each GUI have been set in the GUIDE control inspector to make the program operate as desired. In operation these control values can be modified by writing code to change them that is called by the user prompting the GUI to perform a function. The GUI components have

been positioned in GUIDE manually, numerically or using the snap to grid function. Text displayed with controls is set inside the GUIDE control inspector.

Once GUIDE has generated the figure it opens a template in a Matlab file with headings for the GUI, headings for creation of objects in the GUI and opens functions for the operation of the controls. It also checks some computer background colors and sets control default colors and plotting axes colors in some brief code within the template Matlab file.

The code to operate the GUI's and present the material as a training program is written under function and callback headings to operate slider, pushbutton, radiobuttons and popupmenus controls and under axes and edit box titles to display system output. This code performs the calculations required to generate the simulated vibration data and display it. GUI's that will plot separate Matlab figures, of the data displayed in the GUI, extract the plot information from data sent to the Matlab workspace.

Where the GUI uses a single degree of freedom model or the two degree of freedom model to simulate a vibration response either the SDOF_SOLVER.m file or TDOF_SOLVER functions written for this program is called to construct systems governing differential equations in a form for the iterative solution performed by the Matlab ODE23 solver. The ODE23 solver is time constrained by a time matrix pre built as tspan in the GUI code to achieve a binary length file and generate data in the fashion of what a user is likely to encounter in industry. The binary length data file avoids the need to append zeros to a file or the need to cut it short to achieve the binary length required by the Matlab fft function. The representation of the fft calculation has been manipulated to simulate only digitizing 78.125% of the Nyquist rate to generate a frequency spectrum in the fashion of what a user is likely to encounter in industry. This makes becoming familiar with the sampling properties described in ISO18436-2 easier as the user is visualizing lines of resolution, number of lines, frequency maximum, time to sample and sample sizes and the relationship between them. Some of the simulating GUI's where transients occur at the beginning of the simulation have extended waveforms by a time written into the code and the time constraint tspan is then extended by a required amount of samples and then that number of samples removed from the start of the generated response waveform.

Images displayed in GUI's are .jpg files stored in the Matlab source folder and called when the GUI is opened or when requested by a popupmenu selection.

Text explaining the concepts covered in the GUI's are .txt files stored in the Matlab source folder and called when the GUI is opened.

Equations displayed in GUI's has been written into code using the Matlab Latex Interpreter and are called when the GUI is opened.

3.5 Testing the Program

The graphing of the fft's output as the frequency spectra show fundamentals forcing frequency peaks that align input force sets for simulated run speeds of shafts and defect impact frequencies. These can be checked by plotting Matlab figures from the GUI's and using the Matlab Data Cursor to inspect the frequency and by using the readout on the GUI for the calculated defect frequency. This allows input to be checked against system output where waveforms are generated directly from sinusoids.

The single degree of freedom tests include using code from the system in stand alone files and setting the input mass, stiffness and damping values to those set in Balachandran (2009) example 5.4 on pages 197-199. The following figure 3.7 on page 91 matches that in Balachandran (2009) figure 5.9(iii) on page 199. The single degree of freedom system is also tested by using Level_2_6 GUI and adjusting slider bars and recording system output amplitudes and values. The damping ratio for this test was set at 0.26 and the natural frequency was set at 20Hz. With these settings the simulated shaft speed was adjusted with the slider control from 1 to 50Hz and the output damping ratio and peak displacement amplitude response was recorded. The amplitude response was then calculated using equation 2.21 on page 30. These results are shown in table 3.2 on 89 and plotted in figure 3.8 on page 91 and are similar to that presented in figure 2.5 on 30. Key points of similarity are being near zero at a low frequency ratio, peaking at a frequency ratio of 1 and appearing to approach an asymptote of 1 as the frequency ratio increases beyond 2.

The two degrees of freedom system is tested by using the USQ MEC3403 Dynamics 2 Study Book (2010.2) Exercise 7.8 as a comparison. The code for the system was tested by modifying it slightly to input the matrices for mass, damping and stiffness as per the exercise and comparing the result to that in the activity feedback section of the Study Book. The response result from the written code in figure 3.9 on page 92 is the same as

Table 3.2: This table shows frequency readings (Hz), the system damping ration, the displacement amplitude readings and calculated amplitude response values for Level_2_6 using equation 2.21 on page 30.

Level_2_6 Values	1	2	3	4	5	6	7	8	9	10	11	12
Frequency	1	4	7	10	15	20	25	29	35	40	45	50
Damping Ratio	0.05	0.2	0.35	0.5	0.75	1	1.24	1.44	1.74	1.99	2.24	2.48
Amplitude	0.0006	0.01	0.035	0.08	0.24	0.48	0.46	0.4	0.34	0.32	0.3	0.28
Amplitude Response	0.0025	0.04	0.14	0.33	1.27	9.71	2.79	1.91	1.49	1.33	1.25	1.19

in the USQ Dynamics 2 Study Book.

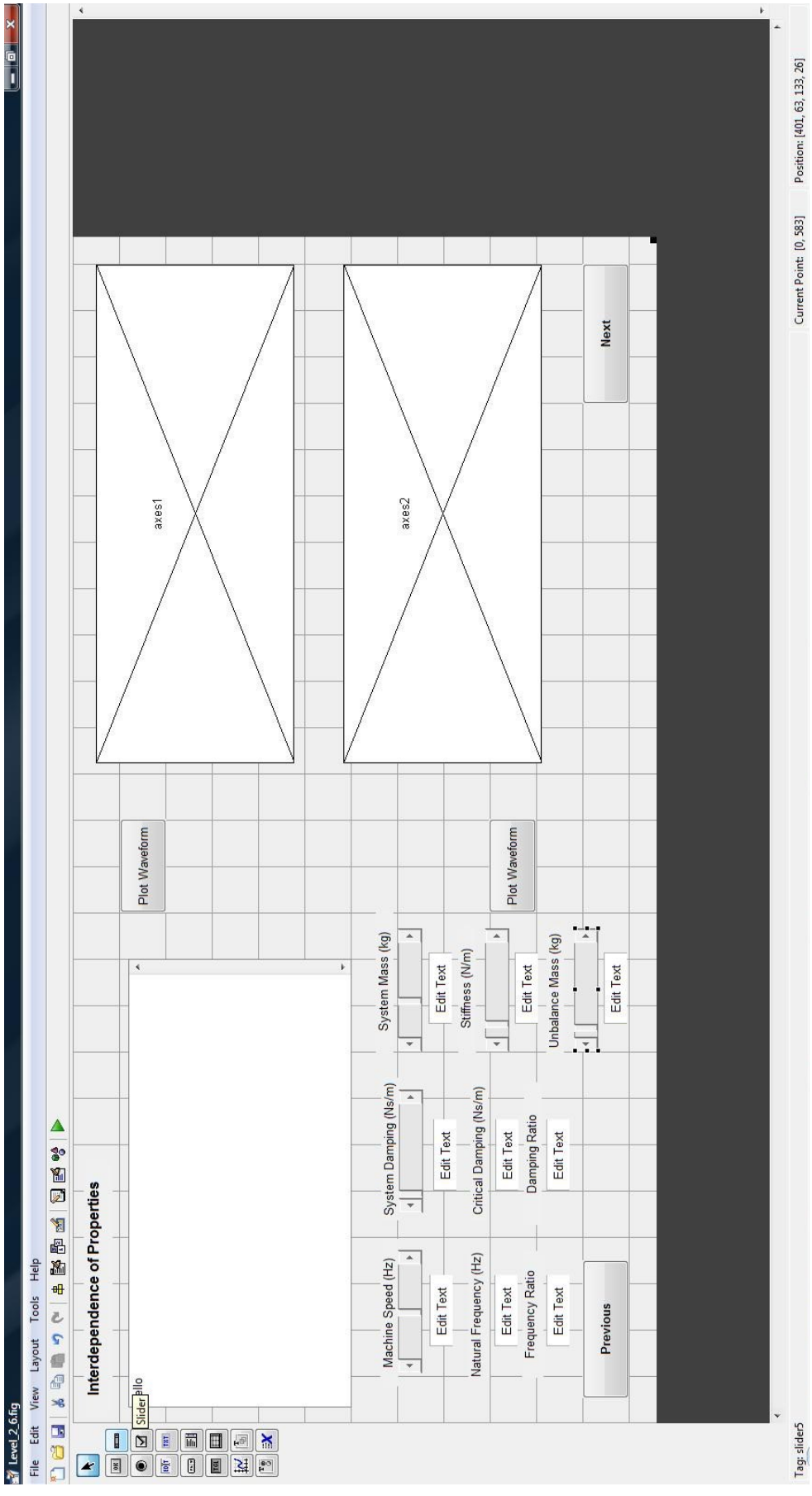


Figure 3.6: This figure shows one of the Matlab graphical user interfaces in Matlab GUIDE where the components for the GUI can be selected from the toolbar on the left and positioned on the grid.

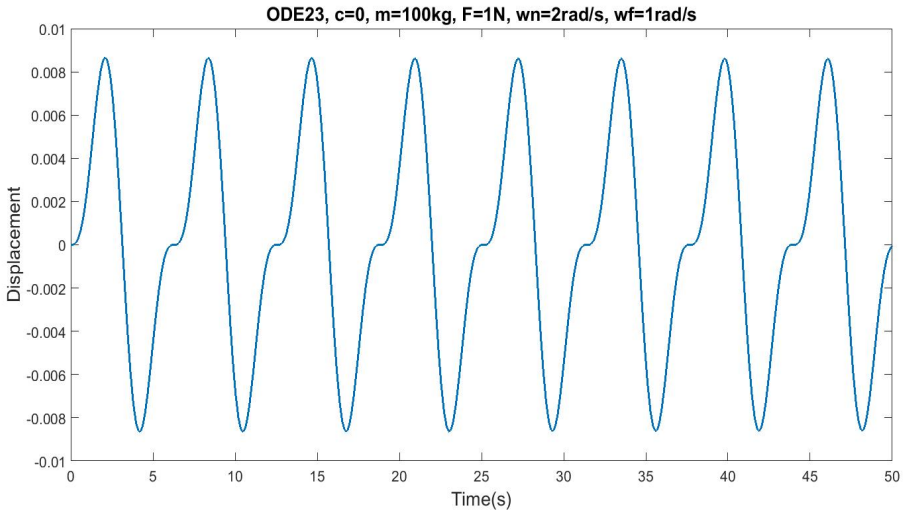


Figure 3.7: This graph is extracted from the program and used to test the output of the single degree of freedom system against a worked example in Magrab (2009) pg 199.

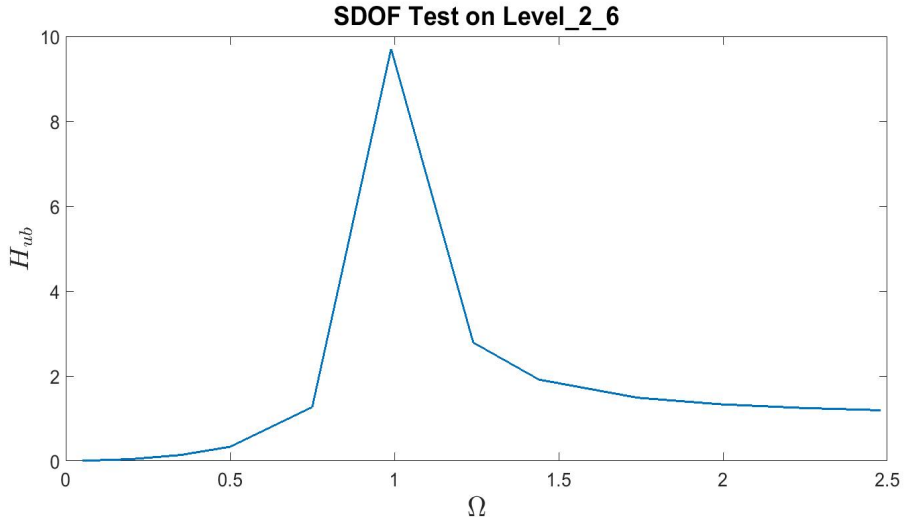


Figure 3.8: This graph is a plot of the recored and calculated data from the the Level_2_6 GUI and is used test the amplitude response of single degree of freedom used in the program. Frequency ration on the horizontal axis and calculated amplitude response on the vertical axis.

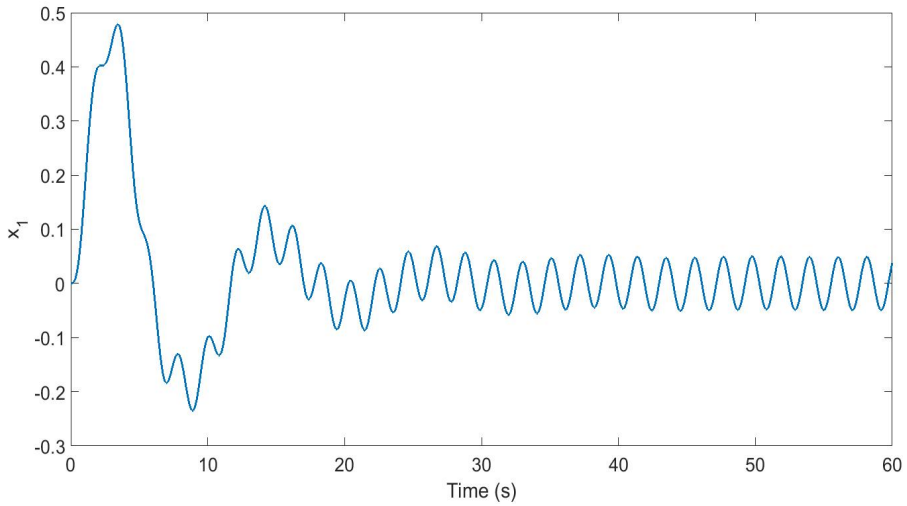


Figure 3.9: This graph is extracted from a section of the Level_2_4 code and used as a comparison to test the function of the two degrees of freedom component of the program.

Chapter 4

Results

4.1 Overview of program

To cover the Level 1 and Level 2 technician vibration topics and design functional requirements the outcome of the program is slightly different in terms of the number of GUI's and the concepts which each covers when compared to the original design flowchart figure 3.1 on page 79. For current flowchart of the program layout see figure 4.1 on page 96

Topics presented in level 1 include principles of vibration, simple harmonic motion, the relationship between period and frequency, basic understanding of phase, managing frequency and amplitude in different units and unit conversions, The user is given an awareness of the time and frequency domains, natural frequencies, resonance, critical speeds and their effect on vibrations, data acquisition parameters of resolution (LOR), maximum frequency (FMax) and time in a sample based on these settings. Signal processing, fault familiarization and recognition of common faults such as looseness, unbalance, misalignment, bearing damage and cavitation are introduced by program simulations. Level 1 provides for recognition of motors, conveyor pulley, crusher, gearboxes motors and a basic understanding of the faults that can be present in them by presenting machine images with faults and concepts. Introduction to acceptance testing is also present with a test for the user on this concept.

Level 2 concepts presented in GUIs for the program include an introduction to integration differentiating, enveloping, band pass filters, demodulation, crest factor, kurtosis, phase reference, resonance, recognition of a vibrating systems properties of frequency, stiffness, mass, damping ,isolation and the interdependence of these properties. Amplitude response, frequency ratio and damping ratio are introduced. The user is introduced to the fft and fft leakage. The user can apply rectangular, Hanning, Hamming and Blackman Harris windows and observe their effect on amplitude and frequency. Averaging to reduce noise is presented with the ability to run 10 averages of a signal. The user can run simulated bearing damage with the ability filter and envelope the acceleration response. Acceleration data can be played as sound at the press of a control button. Data throughout level 2 can be plotted external to the GUI to analyse the data.

Table 4.1: This table shows the Matlab file names for each of the GUI's and the corresponding screen name displayed in the GUI. The file name is also visible in a bar at the top of the GUI when in use.

Matlab File Name	GUI Screen Name
Open.m	Level Selection
Level_1_1.m	Simple Harmonic Motion
Level_1_2.m	The Concept of Phase
Level_1_3.m	Time and Frequency Domains
Level_1_4.m	Frequency and Amplitude Units
Level_1_5.m	Data Sampling Properties
Level_1_6.m	Natural Frequencies
Level_1_7.m	Basic Fault Signals
Level_1_8.m	Bearing Fault Signals
Level_1_9.m	Acceptance Testing
Level_1_10.m	Condition Assessment Test
End_Level_1.m	End Level_1
Level_2_1.m	Integration and Differentiation
Level_2_2.m	The FFT
Level_2_3.m	Signal Processing
Level_2_4.m	Bearings and Signal Processing
Level_2_5.m	Effects of Windows
Level_2_6.m	Interdependence of Properties

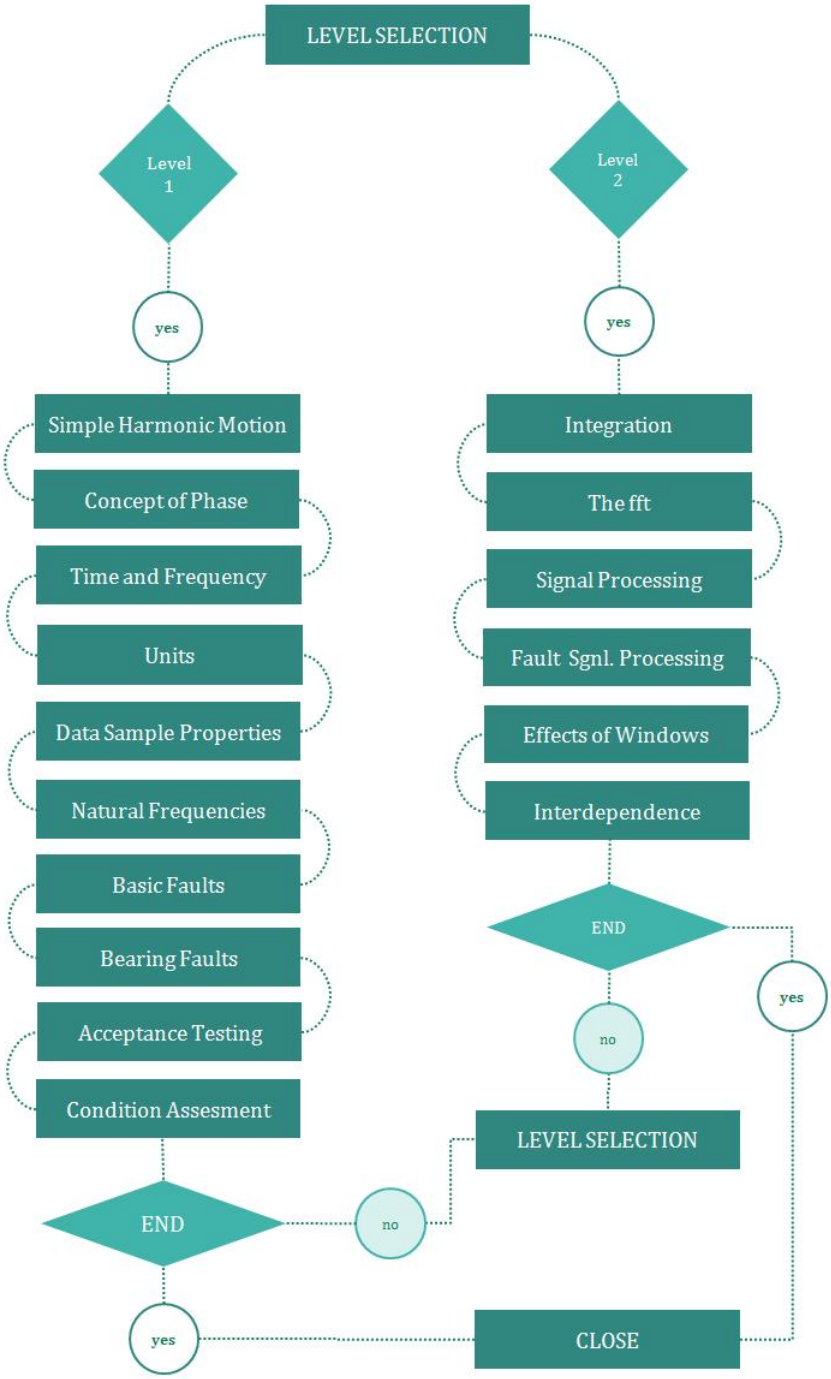


Figure 4.1: This shows the program flowchart and levels, each with an associated GUI.

4.2 Program and GUI Functions

The following are descriptions of the GUI's used in the training program, the concepts covered in them, an overview of their functions and the way in which the written code for the program operates when GUI controls are operated.

4.2.1 Open

This small GUI is the opening of the program. It allows the user to go into the level 1 section of the program or the level 2 section by code running the requested level when the pushbutton is clicked on and closing the Open GUI. See figure 4.2 on page 108

4.2.2 Level_1_1

Level_1_1 present simple harmonic motion (SHM) with an explanation in the of SHM, Hookes law and the use of spring constants and the relationship of linear SHM to the concept of rotation SHM. The GUI has equations displayed in for a sinusoid, Hookes law, Newtons second law and the relationship between frequency and period. The GUI graphs a sinusoid on a plot with amplitude and time axes. The GUI has a slider to change the frequency of the plotted sinusoid and a slider control to change the amplitude of the sinusoid. As these sliders are moves the program recalculates and plots the waveform. A start button begins animation of the blue edit box at the bottom right of the screen. The linear animation runs for the 10 seconds plotted on the axis at the frequency and amplitude presented in the plot. The *Start* pushbutton is deactivated until the animation is complete and then it is reactivated. The *Previous* pushbutton closes this level and runs *Open* and the *Next* pushbutton closes this level and runs

4.2.3 Level_1_2

This GUI present the concept of phase and builds on what is present in *Level*_1_1. Text explains the concept of phase and an equation for a sinusoid shows the position of ϕ to cause a change in phase. The GUI graphs two sinusoids on a plot with amplitude and time axes. The GUI has a slider to change the frequency of the plotted sinusoids. The

frequency is the same for each of the sinusoids. There is a slider control to change the amplitude of each sinusoid. There is a slider control to change the relative phase angle of these sinusoids ±90°. As these sliders are moves the program recalculates and plots the waveforms. A start button begins animation of the blue and red edit boxes at the bottom right of the screen. The linear animation runs for the 10 seconds plotted on the axis at the frequency, amplitudes and phase relationship presented in the plot. The *Start* pushbutton is deactivated until the animation is complete and then it is reactivated. The *Previous* pushbutton closes this level and runs *Level_1_1* and the *Next* pushbutton closes this level and runs *Level_1_3*. See figure 4.4 on page 110

4.2.4 Level_1_3

This GUI introduces the user to the concept of the time domain and its representation in the frequency domain by explaining the concept in text. Graphically the concept is displayed by presenting a single sinusoid and plotting the frequency spectrum for this sinusoid. The amplitude and frequency of the sinusoid are adjusted by the slider bars at which point the program recalculates the fft and plots the frequency spectrum and time waveform. The *Previous* pushbutton closes this level and runs *Level_1_2* and the *Next* pushbutton closes this level and runs *Level_1_4*. See figure 4.5 on page 111

4.2.5 Level_1_4

This GUI presents various frequency and amplitude units to the user. For frequency this is done by the display of text explaining Hertz, cycles per minute, orders and conversion between them. Units of amplitude as displacement, velocity and acceleration are explained covering metric and imperial conversion and acceleration as Gs. Presentation of these units as RMS, peak and peak to peak is covered. Equations for conversion of RPM to Hz is displayed and the equation for calculating the RMS level of a sinusoid. The GUI plots a single sinusoid as a time waveform and is presents as a peak in the frequency spectrum. Positive and negative peak values and the RMS value are highlighted in the waveform. The peak amplitude and RMS amplitude levels are displayed as peaks together in the frequency spectrum. Radio buttons allow the amplitude to be switched between metric and imperial units. The GUI has sliders to change the frequency and amplitude of the sinusoid. As these sliders are moves the program recalculates and plots the waveform

peak and RMS levels, presents the frequency spectrum as peak and RMS values. The *Previous* pushbutton closes this level and runs *Level_1_3* and the *Next* pushbutton closes this level and runs *Level_1_5*. See figure 4.6 on page 112

4.2.6 Level_1_5

This GUI introduces the user to data sampling properties. Text in the GUI explains the way in which vibration data collectors often require some pre collection parameters to be set. Terms and properties such as FMax maximum frequency, LOR lines of resolution, length of time waveform and frequency resolution are covered. The has two popupmenus, one for the user to select from FMax values of 20kHz, 10kHz, 5kHz, 2kHz, 1kHz, 500Hz, 200Hz, 100Hz and 50Hz. The other popupmenu offers LOR values of 25600, 12800, 6400, 3200, 1600, 800 and 400. The code to run the GUI uses these values to produce and plot a binary length sinusoidal waveform and the frequency spectrum is plotted using FMax and lines of resolution specified by the user, using 78.125% of the Nyquist rate. The complete waveform is plotted to familiarize the user with the length of time in the waveform due to the FMax and LOR settings. The GUI has slider controls to change the frequency and amplitude of the sinusoid. As these sliders are moves the program recalculates and plots the waveform and frequency spectrum. The *Previous* pushbutton closes this level and runs *Level_1_4* and the *Next* pushbutton closes this level and runs *Level_1_6*. See figure 4.7 on page 113

4.2.7 Level_1_6

This GUI introduces the user to the concept of natural frequency. Text explains natural frequency as a linear concept and expands to explain how this applies to rotating machinery with rotational speed as a forcing frequency and how the amplitude response/-transmissibility can be influence by a systems stiffness and mass. Equations are displayed for the relationship between natural frequency, system mass and stiffness, the frequency ratio, the ratio between system forcing frequency and natural frequency. The equation for transmissibility for a system with rotating unbalance mass is also displayed. Slider controls are used to vary the values of $ForcingFrequency$, $ForcingAmplitude$ and $Stiffness$ for which the values are displayed in text boxes. There is also a $Mass$ slider control for adjusting system mass. The code running the GUI uses these slider values and the am-

plitude response/transmissibility equation to calculate the system transmissibility. This system transmissibility value is plotted as a red circle on the transmissibility curve for a system with rotating unbalance mass. The user can see this circle move when the sliders are moved and the value displayed as Tr in a text box. The $ForcingAmplitude$ from the slider bar is displayed in a text box and this amplitude is magnified by the transmissibility value. This magnified waveform is plotted in the GUI. This allows the user to see the relationship between natural frequency, stiffness, mass, between forcing frequency and natural frequency and visualize the effect of resonance. The damping ration is fixed in this GUI at 0.3. The *Previous* pushbutton closes this level and runs $Level\_1\_5$ and the $Next$ pushbutton closes this level and runs $Level\_1\_7$. See figure 4.8 on page 114

4.2.8 Level_1_7

This GUI introduces the user to machine defects synchronous with shaft speed. A popupmenu $SelectDefect$ gives the options of unbalance, misalignment, looseness, gear damage and pump vane pass as synchronous faults. Random vibrations of pump cavitation and friction are also selectable in this popupmenu. The popupmenu $SelectMachineSpeed(Hz)$ allows for selections of 50Hz, 25Hz, 16.6Hz and 8.3Hz. Different Fmax and LOR are used for each of these shaft speeds in calculating the system response. The defect fundamental frequency is displayed in text boxes in the GUI as Hz and orders of shaft speed. Text for this GUI covers the concept of faults that are known as synchronous and explains unbalance, misalignment and looseness faults. The program code is calculating the response of a single degree of freedom system to input forces. The unbalance force is calculated at the iterative time step. There are four pulse train input forces constructed and summed for the complete time sequence prior to calling the ODE23 solver and the function file $SDOF\_SOLVER$. The ODE23 solver iterative process is time constrained by a preconstructed array to generate a waveform with an evenly space sample period across it and to produce the binary length wave file. In this GUI there is a delay of $dt = 0.1second$ in the code to allow for the transient at the beginning of the wave to settle out in some of the simulations. This adds 0.1 seconds worth of samples (rounded) to the end of the simulation and removes the same number of samples from the beginning of results, removing the transient and retaining the desired file length. The resulting velocity response is graphed as a waveform and frequency spectrum. 78.125% of the Nyquist rate is plotted in the frequency spectrum. The $PlotWaveform$ and $PlotSpectrum$ plots the graphs

displayed in the GUI as separate Matlab figures by extracting the information that has been sent to the workspace. These figures allow closer inspection of the data with the tools offered in the figure toolbar. The *Previous* pushbutton closes this level and runs *Level_1_6* and the *Next* pushbutton closes this level and runs *Level_1_8*. See figure 4.9 on page 115

4.2.9 Level_1_8

This GUI introduces the user to non-synchronous faults in the way of bearing defects. The text for this GUI covers the rolling element bearing faults and the nature of them that makes them non-synchronous. A popupmenu $NonSynchronousDefects$ gives the options of bearing cage defect, bearing ball defect, bearing outer race defect and bearing inner race defect. The popupmenu $SelectMachineComponent(Hz)$ allows for selections of conveyor pulley, crusher rotor, gearbox second shaft, motor NDE and motor DE. This menu also selects information pertaining to a real machine, i.e. its shaft rotational speed and a bearing fault ratio set. This also sets the FMax and LOR for the simulation run in this GUI and presents an image relevant to the machine component selected. The machine shaft speed in Hz and defect fundamental frequency as Hz and orders of shaft speed for the current simulation are displayed in the GUI. The code operating this GUI is similar to that of *Level_1_7*. The unbalance force is calculated at the iterative time step though its unbalance mass is reduced in this code. The pulse train input forces is constructed and summed for the complete time sequence prior to calling the ODE23 solver. In this GUI the pulse train is set either as a constant or modulating amplitude depending on the bearing defect selected in the popupmenu. The modified sinusoid presented in Chapter 3 is employed in the case of requiring amplitude modulation. The ODE23 solver iterative process is time constrained by a pre-constructed array to generate a waveform with an evenly space sample period across it and to produce the binary length wave file. The resulting velocity response is graphed as a waveform and frequency spectrum. 78.125% of the Nyquist rate is plotted in the frequency spectrum. The $PlotWaveform$ and *PlotSpectrum* plots the graphs displayed in the GUI as separate Matlab figures by extracting the information that has been sent to the workspace. These figures allow closer inspection of the data with the tools offered in the figure toolbar. The *Previous* pushbutton closes this level and runs *Level_1_7* and the *Next* pushbutton closes this level and runs *Level_1_9*. See figure 4.10 on page 116

4.2.10 Level_1_9

This level introduces the user to Acceptance testing. This is achieved by explaining the reasons testing is conducted, introducing ISO 10816 part 1, the four machine evaluation condition levels, and the four machine classes. The table displayed in the GUI categorizes these evaluation conditions based on machine class and RMS machinery velocity mm/s. The text explains the use of this table and offers an example in applying it. There is a popupmenu $ClassofMachine$ where the user has the choice of a class 1, class 2, class 3 or class 4 category machine. Selecting a machine will display an image of a machine in this category. An RMS level for the vibration waveform plotted is displayed in the GUI next to the waveform. The $ChangeMachine$ pushbutton randomly generates another waveform of the same frequency but different amplitude. The RMS is displayed and the code assesses the machine evaluation based on the RMS level for this class of machine as good (A), acceptable (B), satisfactory for short term operation (C) or damaging (D) based on the standard table displayed. The $PlotWaveform$ and $PlotSpectrum$ plots the graphs displayed in the GUI as separate Matlab figures by extracting the information that has been sent to the workspace. These figures allow closer inspection of the data with the tools offered in the figure toolbar. The $Previous$ pushbutton closes this level and runs $Level\_1\_8$ and the $Next$ pushbutton closes this level and runs $Level\_1\_10$. See figure 4.11 on page 117

4.2.11 Level_1_10

This level is a condition assessment test for the level 1 user, based on ISO10816-1 presented in $Level\_1\_9$. There is a popupmenu $ClassofMachine$ where the user has the choice of a class 1, class 2, class 3 or class 4 category machine. Selecting a machine will display an image of a machine for this category. An RMS level for the vibration waveform plotted is displayed in the GUI next to the waveform. The test for the user is to determine the machine evaluation condition from the given table as good (A), acceptable (B), satisfactory for short term operation (C) or damaging (D) based on the RMS level and the class of machine selected. The correct answer will highlight bring up a green box with the word $Correct$ or it will be red $Incorrect$. The $ChangeMachine$ pushbutton randomly generates another waveform of the same frequency but different amplitude. The waveform is generated with the single degree of freedom system similar to that of $Level\_1\_7$

with dominant unbalance and only a small 6.3 order defect present. The correct answer is sent to a box in the GUI by an if, $elseif$ loop and is not visible to the user. Another if, $else$ loop is set up for each pushbutton A, B, C and D to evaluate the users answer as correct or incorrect and to set the color green or red. Selecting $ChangeMachine$ or $ClassofMachine$ resets the machine evaluation process and resets the background color. The $PlotWaveform$ and $PlotSpectrum$ plots the graphs displayed in the GUI as separate Matlab figures by extracting the information that has been sent to the workspace. These figures allow closer inspection of the data with the tools offered in the figure toolbar. The *Previous* pushbutton closes this level and runs *Level_1_9*. At this stage the *Next* pushbutton closes this level and runs the *EndLevel_1* GUI. See figure 4.12 on page 118

4.2.12 End Level_1

This GUI gives the user the options of *ReturntoLevel_1_10*, open *LevelSelection* or *CloseProgram*. See figure 4.13 on page 119

4.2.13 Level_2_1

This GUI covers the topic of integration and differentiation by way of explaining the concepts and the way it is presented in the GUI. Equations for integration and differentiation are displayed in the GUI. A displacement sinusoid is constructed in Matlab and the differentiation of it once is used to present velocity and twice to present acceleration. This is plotted to show the relationship between them in an ideal case. The amplitude and frequency of the displacement sinusoid are adjustable by slider controls in the GUI. A simple animation of the three colored boxes traces out the plotted waves. The boxes are named D (displacement), V (velocity) and A (acceleration) with colors matching those in the plot, begins when the *Start* pushbutton is used. $PlotWaveform$ plots the currently displayed waveform in a separate Matlab figure for close inspection. The *Previous* pushbutton closes this level and runs *LevelSelection* and the *Next* pushbutton closes this level and runs *Level_2_2*. See figure 4.14 on page 120

4.2.14 Level_2_2

This level offers an explanation of the discrete and fast Fourier transforms of the time waveform and converting the time domain into the frequency domain and the use of binary length signals for the fft. A randomly generated waveform with six frequencies is compiled. Frequency is from0-200Hz and amplitude from 0-5 units. The FMax is set at 200Hz and LOR at 800. The $ChangeWaveform$ button generates a new waveform, calculates the fft and plots the frequency spectrum. The $Previous$ pushbutton closes this level and runs $Level\_2\_1$ and the $Next$ pushbutton closes this level and runs $Level\_2\_3$. See figure 4.15 on page 121

4.2.15 Level_2_3

This GUI introduces the user to signal processing. The text in the GUI covers filtering, frequency bands and enveloping a signal to remove the unwanted part of a signal. It also explains crest factor and kurtosis as a way of evaluating data and covers averaging of multiple spectra of a noisy signal to remove unwanted noise. These are the topics presented for the user in the GUI. Waveforms are generated in a similar fashion to that in $Level\_2\_2$ with the addition of upper and lower sidebands added to one of the randomly generated frequencies. The FMax is set at 200Hz and LOR at 800. A random level of noise is also generated with each use of the $ChangeWaveform$ pushbutton. The RMS level, crest factor and kurtosis are displayed for each plotted waveform. The filter band is a pass band. The filter, its parameters are generated using Matlabs $firpmord$ and $firpm$ functions including the filter delay calculation. The generated filter is applied to the constructed waveform for plotting and the fft is performed on the filtered waveform and plotted. At the same time the Hilbert transform function in Matlab is applied to the filtered waveform data creating an envelope and exported to the workspace. When the filter radiobutton is used the envelope button is enabled, and when pressed the envelope the waveform and envelope are plotted together and the frequency spectrum of the Hilbert transform envelope is plotted with the spectrum of the filtered waveform. When the $FilterBand$ radiobutton is pressed the $SamplesinAverage$ radiobutton is disabled, visa versa. There is a slider control for the user to select the number of samples to average. When the averaging begins program extracts all of the frequencies, amplitudes and level of noise from the currently plotted waveform from the workspace. These are used to generate

a new waveform with variation in base amplitudes and noise level to replicate variation that might occur in a real machine. A delay allows for plotting each of the waveforms and their spectrum used in the average process. Each generated spectrum is added to the previous in a loop and when the number set on the slider control is reached the average spectrum is plotted removing noise and averaging frequency amplitudes. Average RMS, crest factor and kurtosis is also displayed in the GUI. The *Previous* pushbutton closes this level and runs *Level_2_2* and the *Next* pushbutton closes this level and runs *Level_2_4*. See figure 4.16 on page 122

4.2.16 Level_2_4

This GUI covers the Non-Synchronous faults introduces in Level_1_8 with the two degree of freedom system covered in Chapter 3 in the background giving a simulated vibration response. The function file $TDOF\_SOLVER.m$ is called by level 4 to assemble the state A and B matrices for the system, input the forcing matrix for the ODE23 solver used. The unbalance force is calculated at the iterative time step. In this GUI the pulse train is set either at constant or modulating amplitude depending on the bearing defect selected in the popupmenu. The modified sinusoid presented in Chapter 3 is employed in the case of requiring amplitude modulation. The ODE23 solver iterative process is time constrained by a pre-constructed array to generate a waveform with an evenly space sample period across it and to produce the binary length wave file. The resulting response for $Mass1$ with unbalance force applied graphed as a waveform and frequency spectrum can be viewed in *Acceleration* or $Velocity$ on using the so named radiobuttons. This GUI also combines the filtering function of Level_2_3 to the simulated bearing defects as an example of the application of demodulation to a realistic vibration signal from a bearing defect. As in *Level_1_8* the following bearing faults, Inner Race Defect Frequency, outer Race Defect Frequency, Ball Spin Frequency, Cage Defect Frequency can be displayed for a conveyor pulley, a crusher rotor, a gearbox shaft, and both ends of a motor. While viewing the acceleration response the $FilterBand$ radiobutton is enabled. Set the lower and upper band pass filter settings to view the desired band. Once the data is filtered and displayed operating the *Envelope* radiobutton will plot the Hilbert transform envelope on the filtered waveform and plot a frequency spectrum of the envelope itself. This has the effect of removing the high frequency signal and highlighting the defect forcing frequency. The shaft speed is displayed as Hz and the defect fundamental frequency as Hz and orders

of shaft speed. At any stage the waveform and spectrum viewed in the GUI can be plotted as a separate figure for closer inspection use the Matlab zoom, cursor and other functions for analysis. Figure 4.18 on page 124 is a plot from this GUI of the Hilbert transform as an envelope plotted on a filtered waveform for simulated inner race data, showing how the higher frequency component of the waveform is removed by the envelope. Sound for the acceleration waveforms can also be played. The *Previous* pushbutton closes this level and runs *Level_2_3* and the *Next* pushbutton closes this level and runs *Level_2_5*. See figure 4.17 on page 123

4.2.17 Level_2_5

This GUI introduces windows and their effects on the time and frequency domain. In this GUI they are applied to the time domain and the shape of the waveform can be seen change when applied. The FMax and LOR are set at 200 making a coarse spectrum. This makes it easier to see the effect of the window. When the GUI is opened the windows for these data settings are created and sent to the workspace. The waveform is randomly generated with some frequencies purposefully close together. The Change Waveform pushbutton generates a new waveform. The waveform is sent to the workspace. When a 'Windows' radiobutton is operated the appropriate window function and current waveform are extracted from the workspace to create the modified time waveform and then conduct the fft and display the resulting frequency spectrum. This allows the user to view the effect the chosen window has on the amplitude and frequency in the spectrum where peaks can change height or disappear. Because of the effect windows have on waveform amplitude across the time domain the results of the fft must be increased by a factor to compensate. For the Hanning window the correction factor is 2, Hamming 1.85 and Blackman Harris 2.36. The Previous pushbutton closes this level and runs Level_2_4 and the Next pushbutton closes this level and runs Level_2_6. See figure 4.19 on page 125

4.2.18 Level_2_6

This GUI demonstrates the interdependence of machine properties that influence vibration. The text explains the concepts to be observed in the GUI. For the single degree of freedom system working behind the GUI there are slider controls for Machine Speed, System Damping, System Mass, Stiffness, and Unbalance Mass. The current value of

each of these slider controls is displayed in the GUI. Other system properties calculated and displayed are the natural frequency of the system, the frequency ratio, the systems critical damping factor and the current damping ratio. The FMax is set at 2000Hz and the LOR at 6400. The unbalance force is calculated at the iterative time step. There is no pulse train force for this system, however the same function file as used by other GUIs, SDOF_SOLVER.m is used so for this purpose it is made of zeros. The ODE23 solver is used and the iterative process is time constrained by a pre-constructed array to generate a waveform with an evenly space sample period across it and to produce the binary length wave file. In this GUI there is a delay of dt = 1 second in the code to allow for the transient at the beginning of the wave to settle out in some of the simulations. This adds 1 seconds worth of samples (rounded) to the end of the simulation and removes the same number of samples from the beginning of results, removing the transient and retaining the desired file length. The orange trace in the graphs are the position of the unbalance mass and provides a phase reference to observe the effect the change in properties has on the phase of the response relative to the forcing function of unbalance. The displacement response is in the top plot and the velocity response in the bottom plot. Plot the complete waveform as a Matlab figures for further inspection or analysis. The Previous pushbutton closes this level and runs Level_2_5 and the Next pushbutton closes this level and runs the Open GUI Level Selection. See figure 4.20 on page 126

4.3 GUI Images

There are 10 Level 1 GUI in the program Level_1_1 to Level_1_10. The level section has 6 GUI's from Level_2_1 to Level_2_6. The following images show these GUI's, the Open (Level Selection) GUI and the End Level_1 GUI.

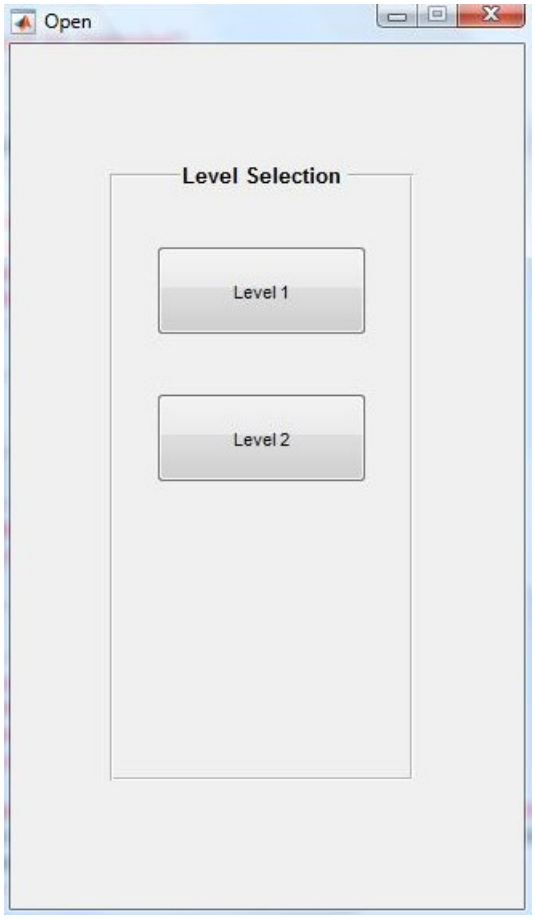


Figure 4.2: This shows the Open.m GUI for initial entry into the program. It allows the user to enter Level 1 or the Level 2 section of the program. Space has been left to include a Level 3 section

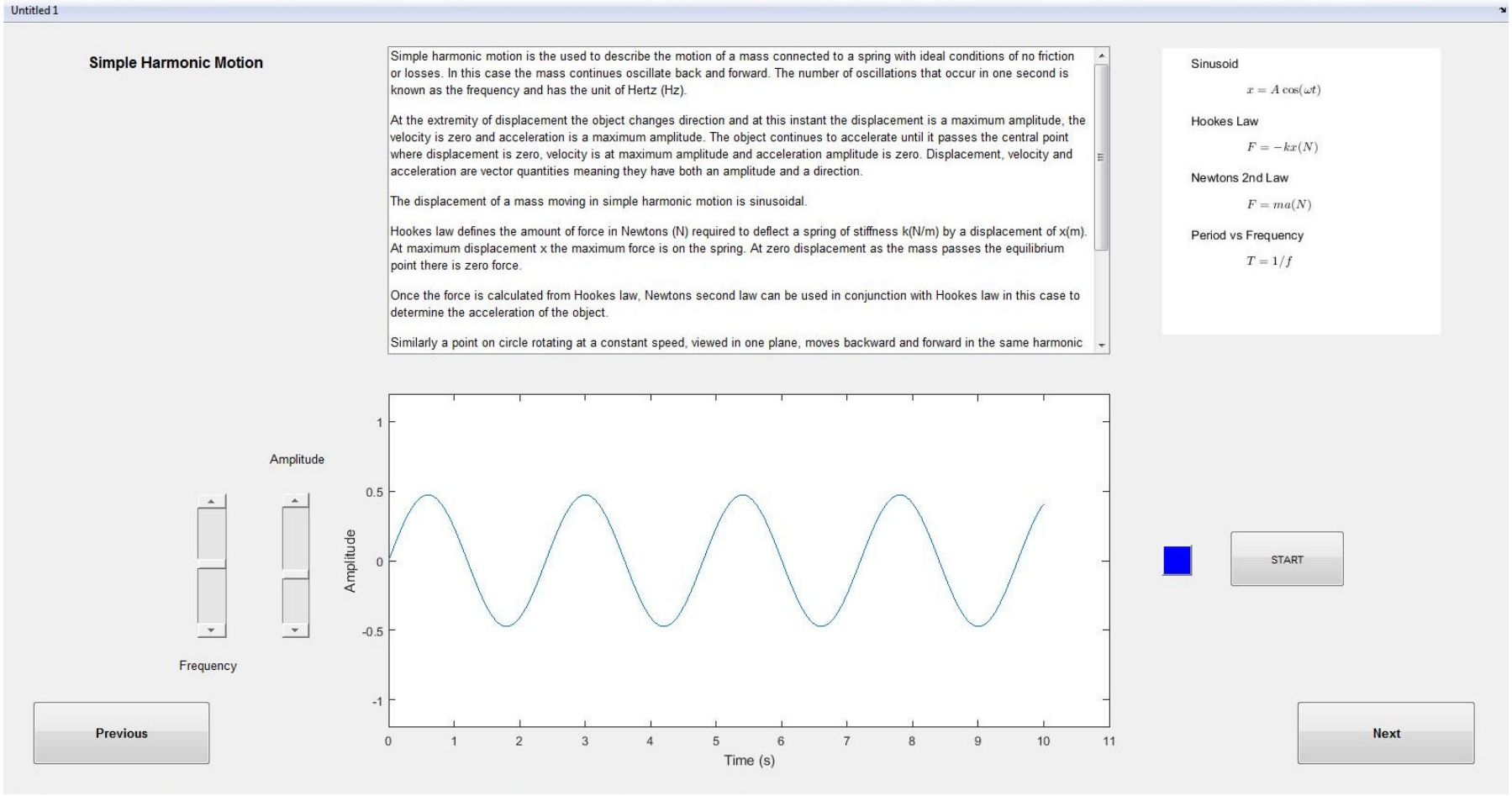


Figure 4.3: This screen shot shows the Level_1_1 GUI. It explains simple harmonic motion as a mass on a spring oscillating back and forward and displays a sine wave. The two sliders in the GUI allow the user to change the amplitude and frequency of the sine wave. The 'Start' pushbutton begins the blue box moving in the pattern and frequency displayed in the sine wave

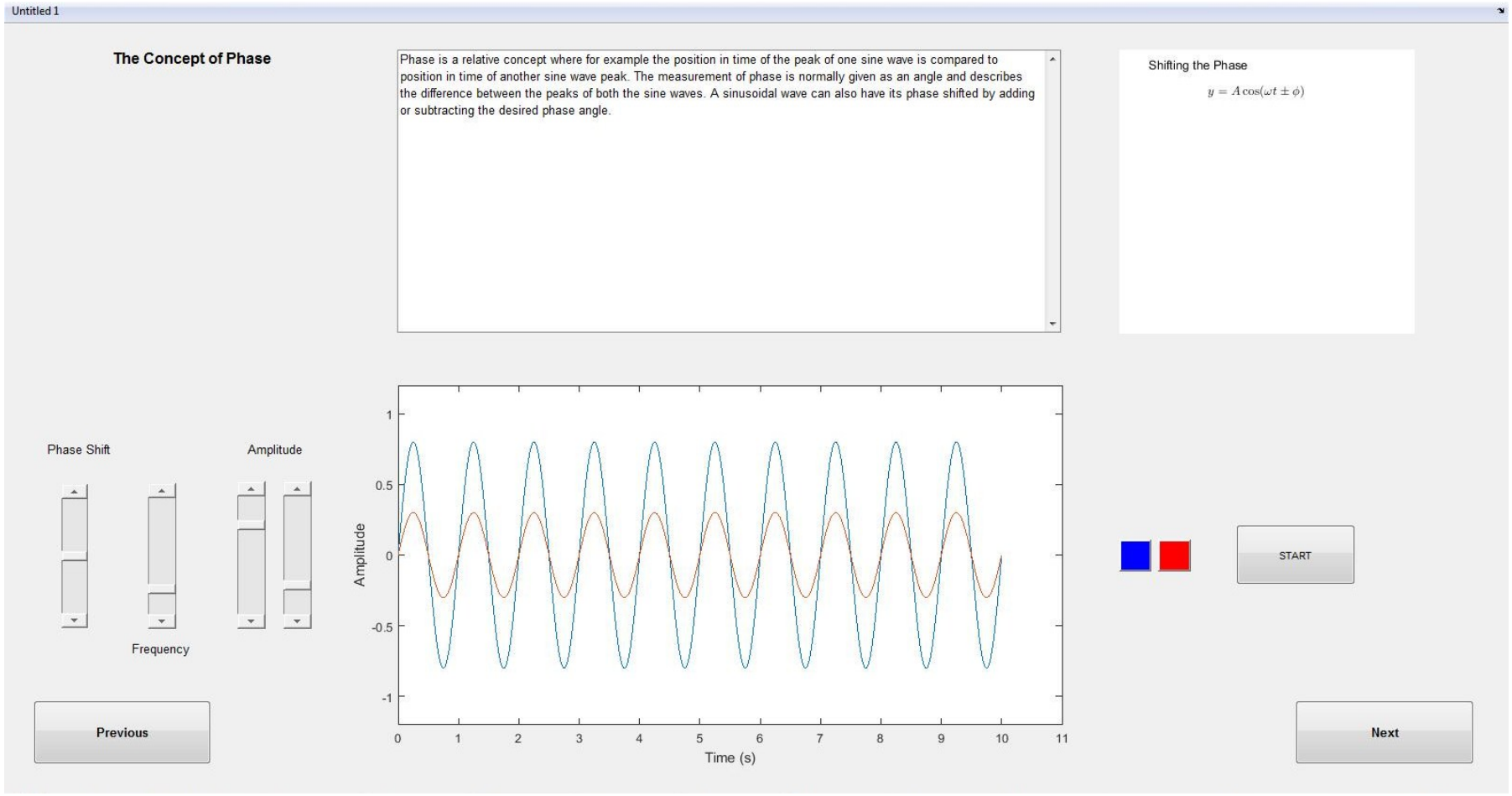


Figure 4.4: This screen shot shows the Level_1_2 GUI. It covers the concept of phase. It is similar to Level_1_1 with the addition of another sinusoid displaying simple harmonic motion. There are two sliders in the GUI allow the user to change the amplitude, one slider to adjust the frequency of the waves that each remain at the same frequency. The slider bar on the far left of the GUI adjusts the phase angle. The 'Start' pushbutton begins the blue and red boxes moving in the pattern and frequency displayed in the sine waves to demonstrate two sinusoids and the concept of phase.

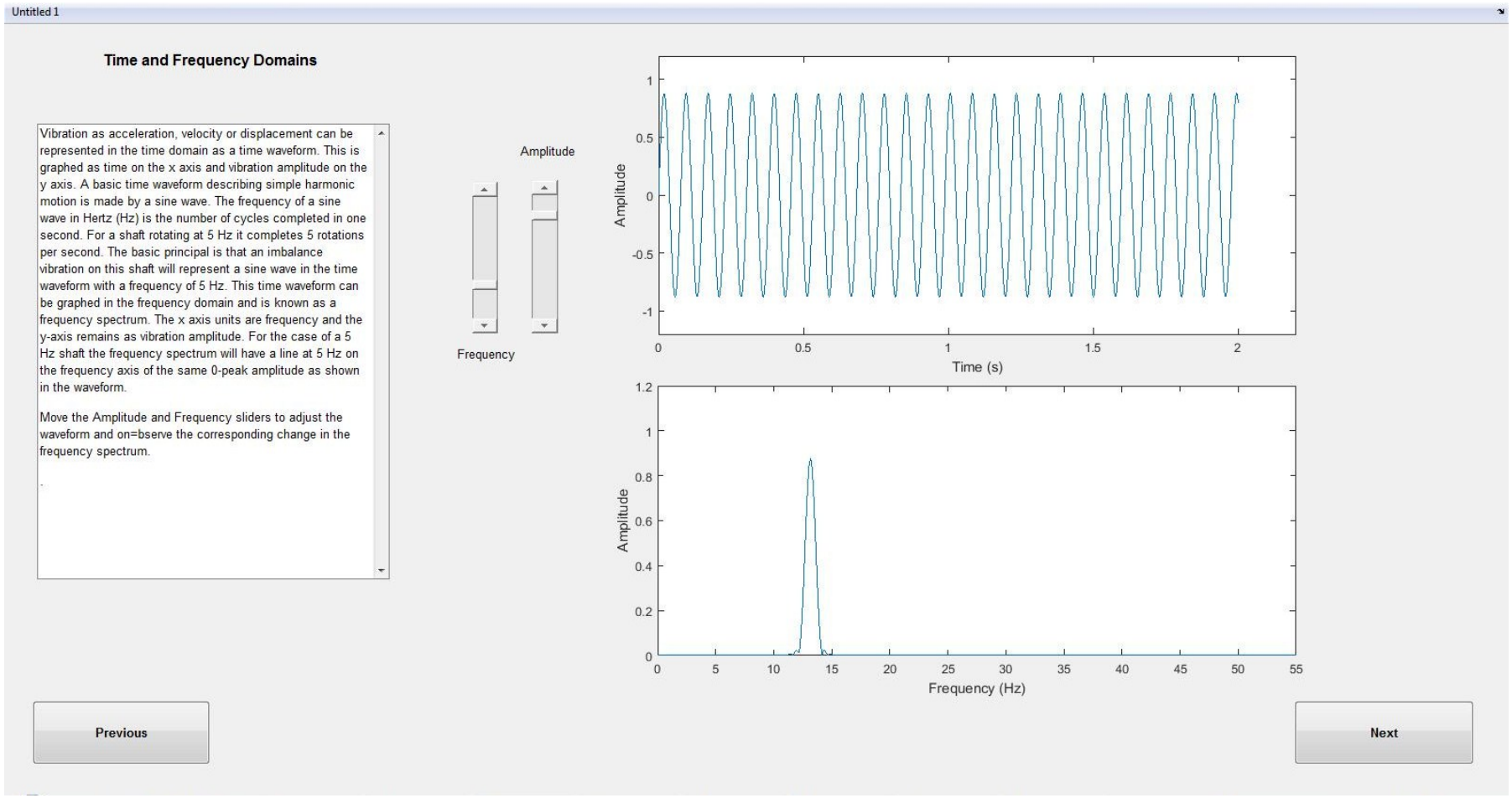


Figure 4.5: This screen shot shows the Level_1_3 GUI. It introduces the Level 1 user to the time domain and the frequency domain by displaying a simple waveform and frequency spectrum of the waveform. The user can adjust the amplitude and frequency with the slider bars.

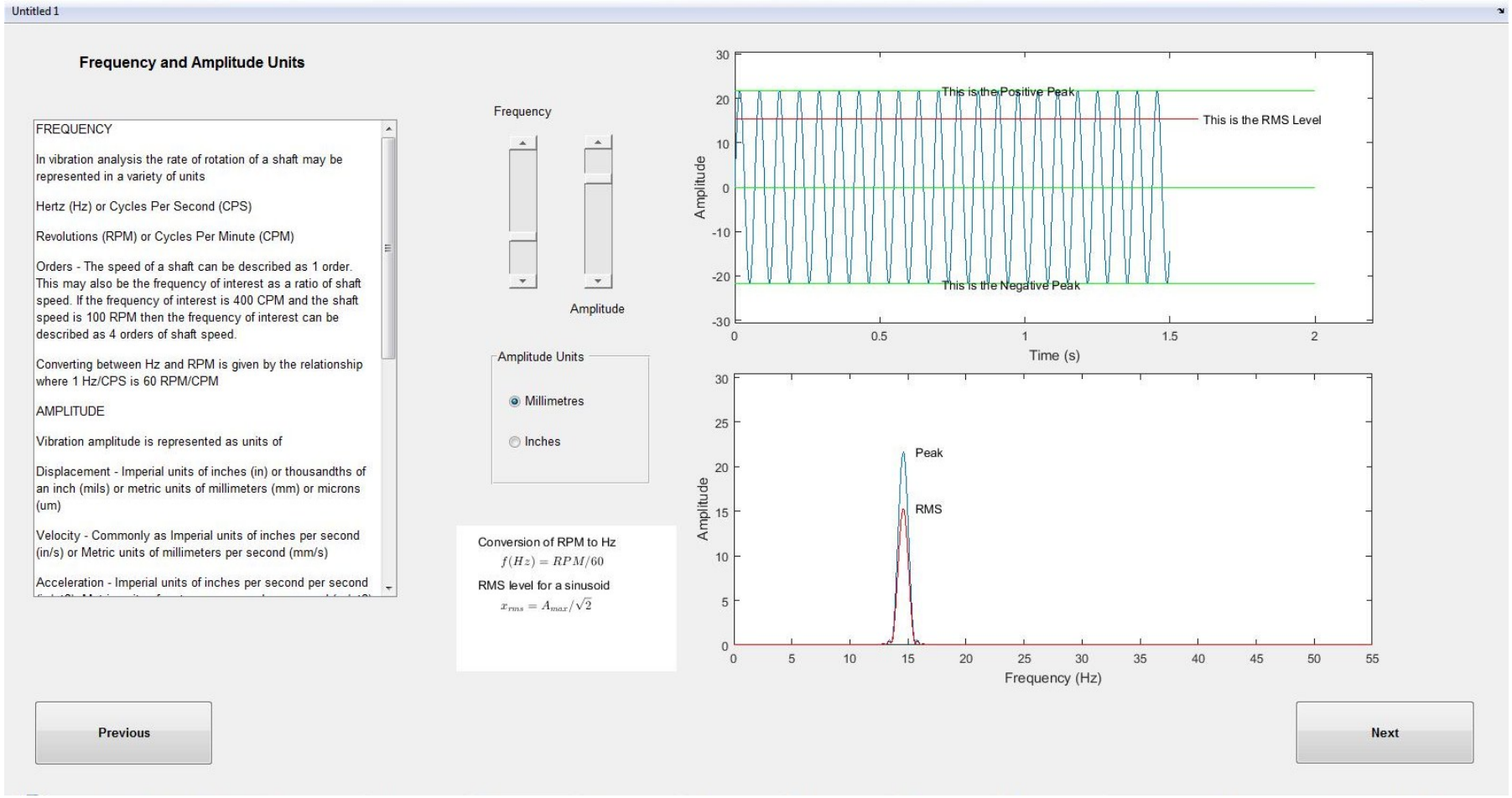


Figure 4.6: This screen shot shows the Level_1_4 GUI. This builds on the previous GUIs by explaining and displaying frequency units and amplitude units in the time and frequency domain. The user can adjust the amplitude and frequency with the slider bars. Units can be changed between millimetres and inches. RMS and peak levels are introduced to the user.

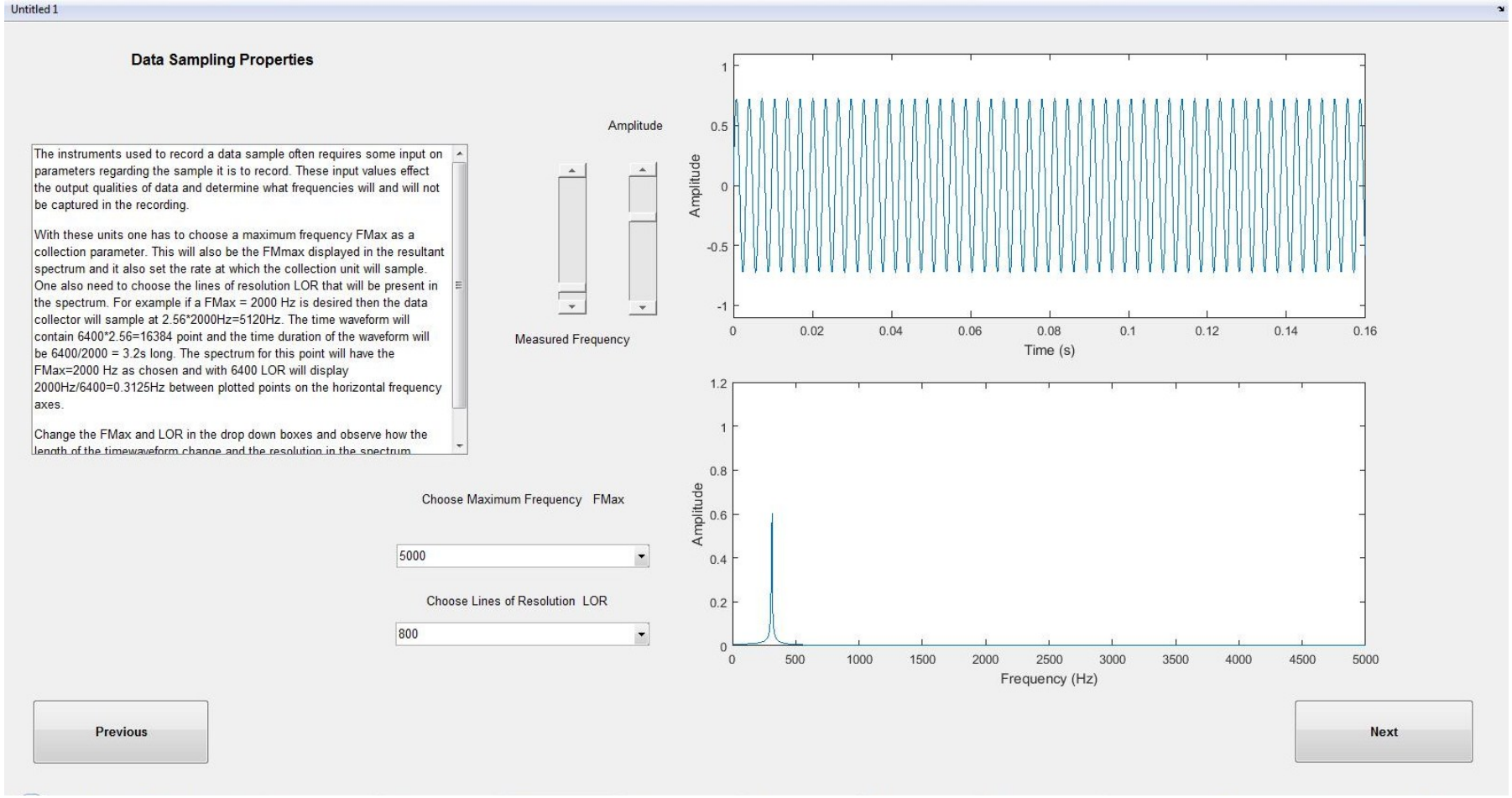


Figure 4.7: This screen shot shows the Level_1_5 GUI. It introduces the user to commonly used data sampling properties, the maximum frequency and lines of resolution. The length of the time waveform based on these inputs is displayed with associated frequency spectrum. The user can adjust the amplitude and frequency of the sinusoid with the slider bars.

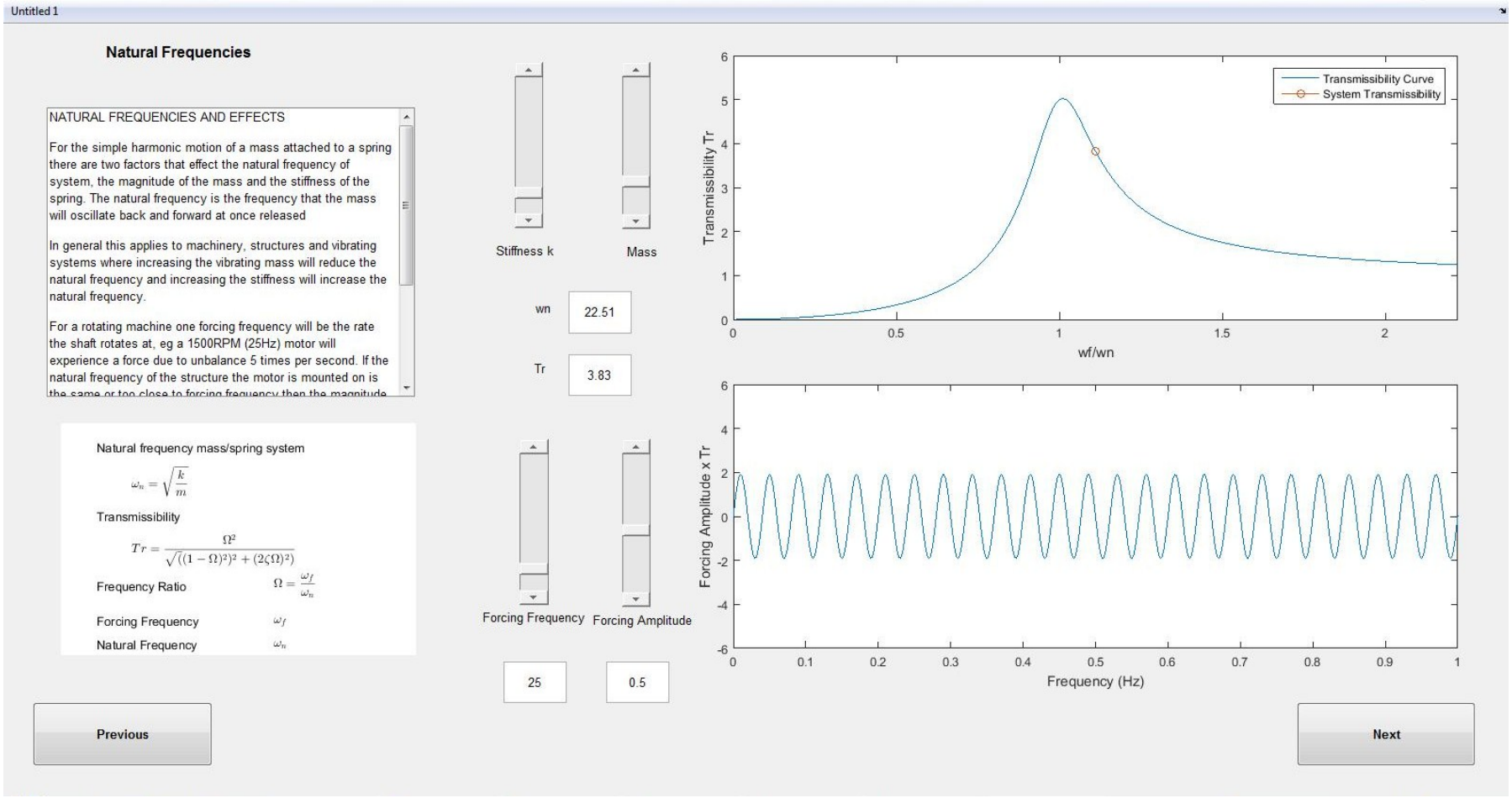


Figure 4.8: This screen shot shows the Level_1_6 GUI. It shows the user the effect that changing the stiffness and the mass has on a single degree of freedom system with rotating unbalance vibration. It allows the user to change the system properties on the slider bars and observe the change in vibration amplitude.

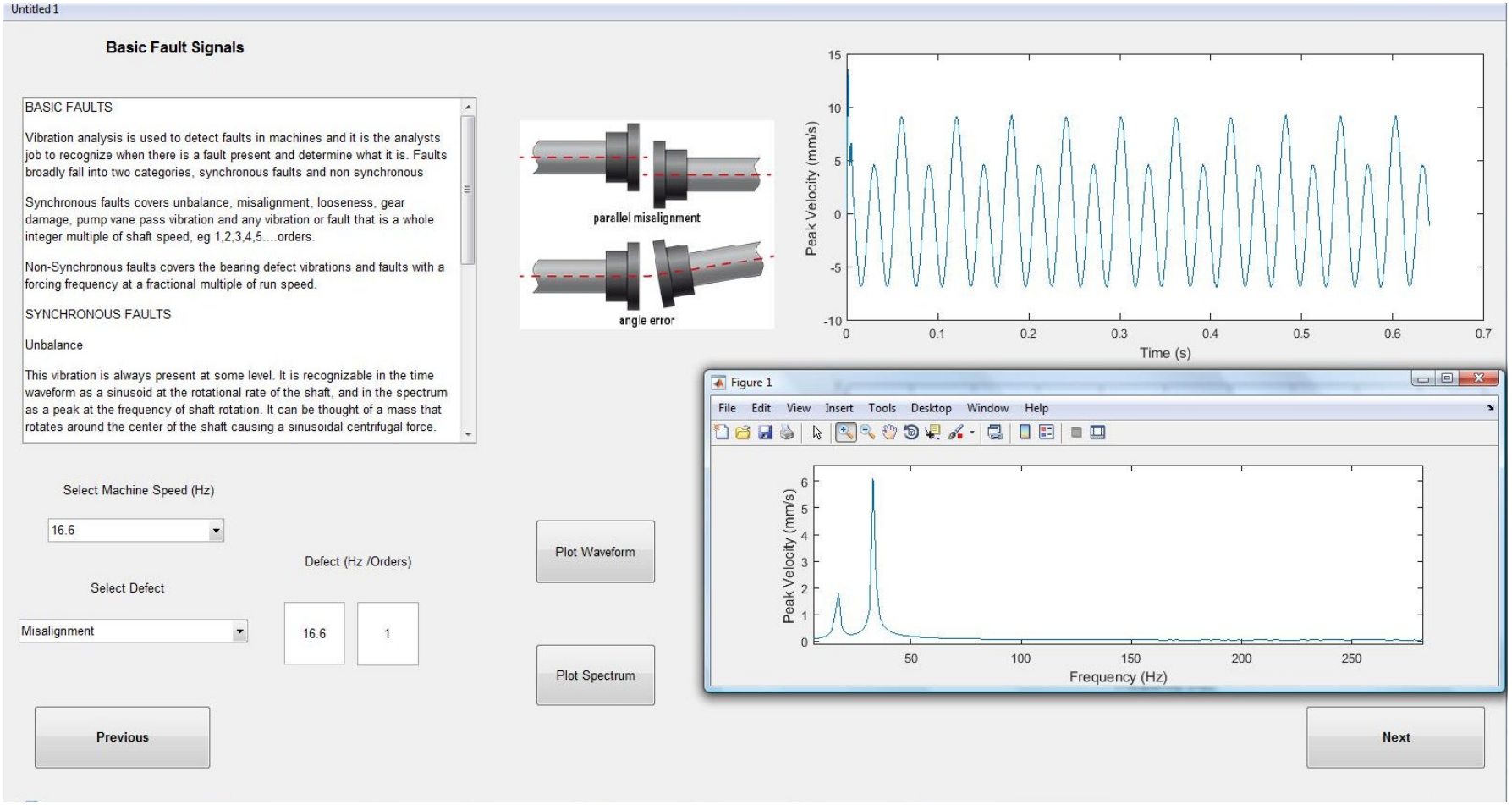


Figure 4.9: The Level_1_7 GUI allows the user to select from several synchronous defects and machine speed and observe the response of a single degree of freedom system with otherwise fixed parameters. The waveform and spectrum can also be plotted to allow close inspection with plot tools and features.

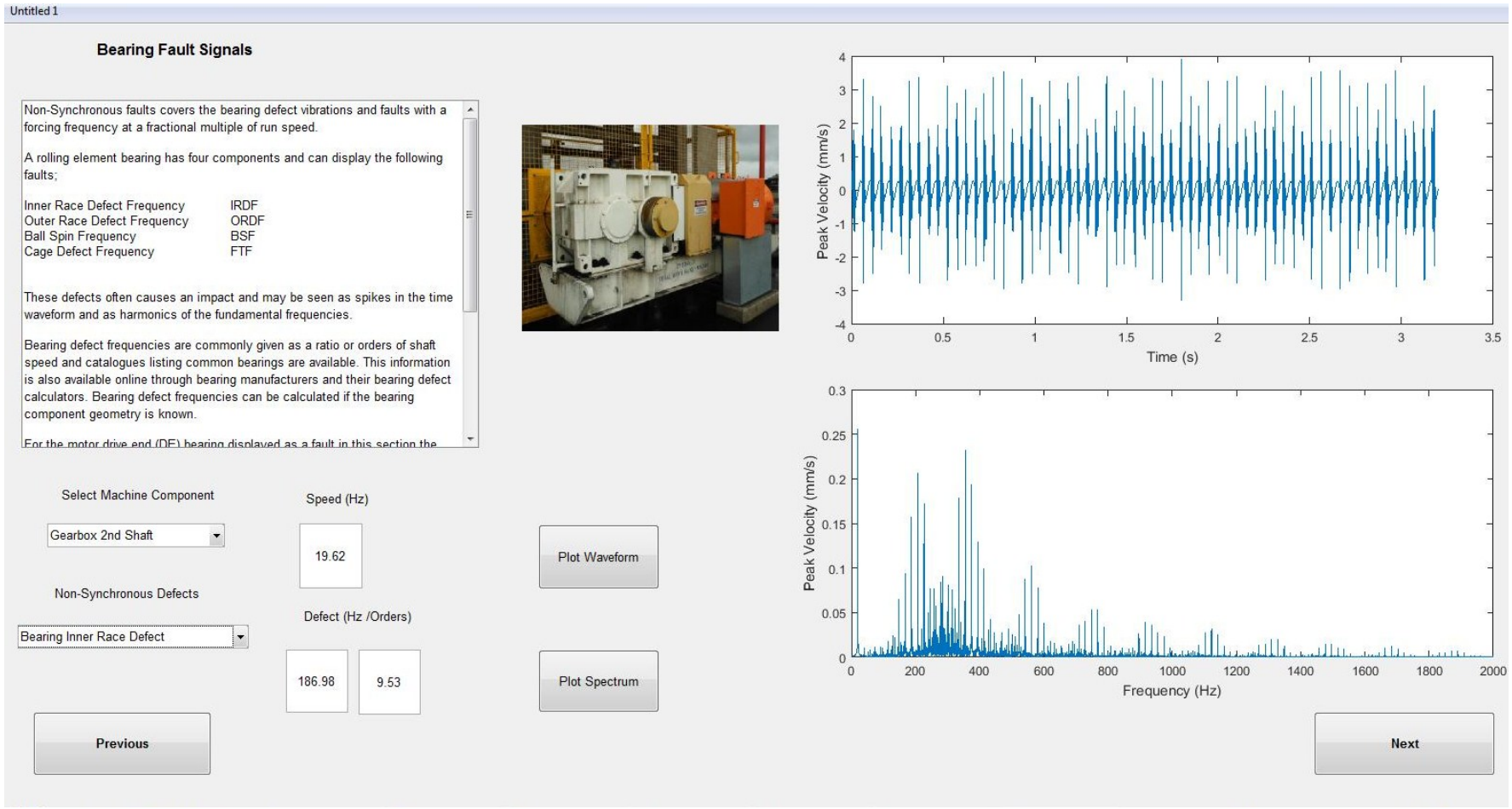


Figure 4.10: The Level_1_8 GUI allows the user to select from bearing defects. Here selecting a machine component changes the machine speed. The response of a single degree of freedom system is plotted. The waveform and spectrum can also be plotted to allow close inspection with plot tools and features.

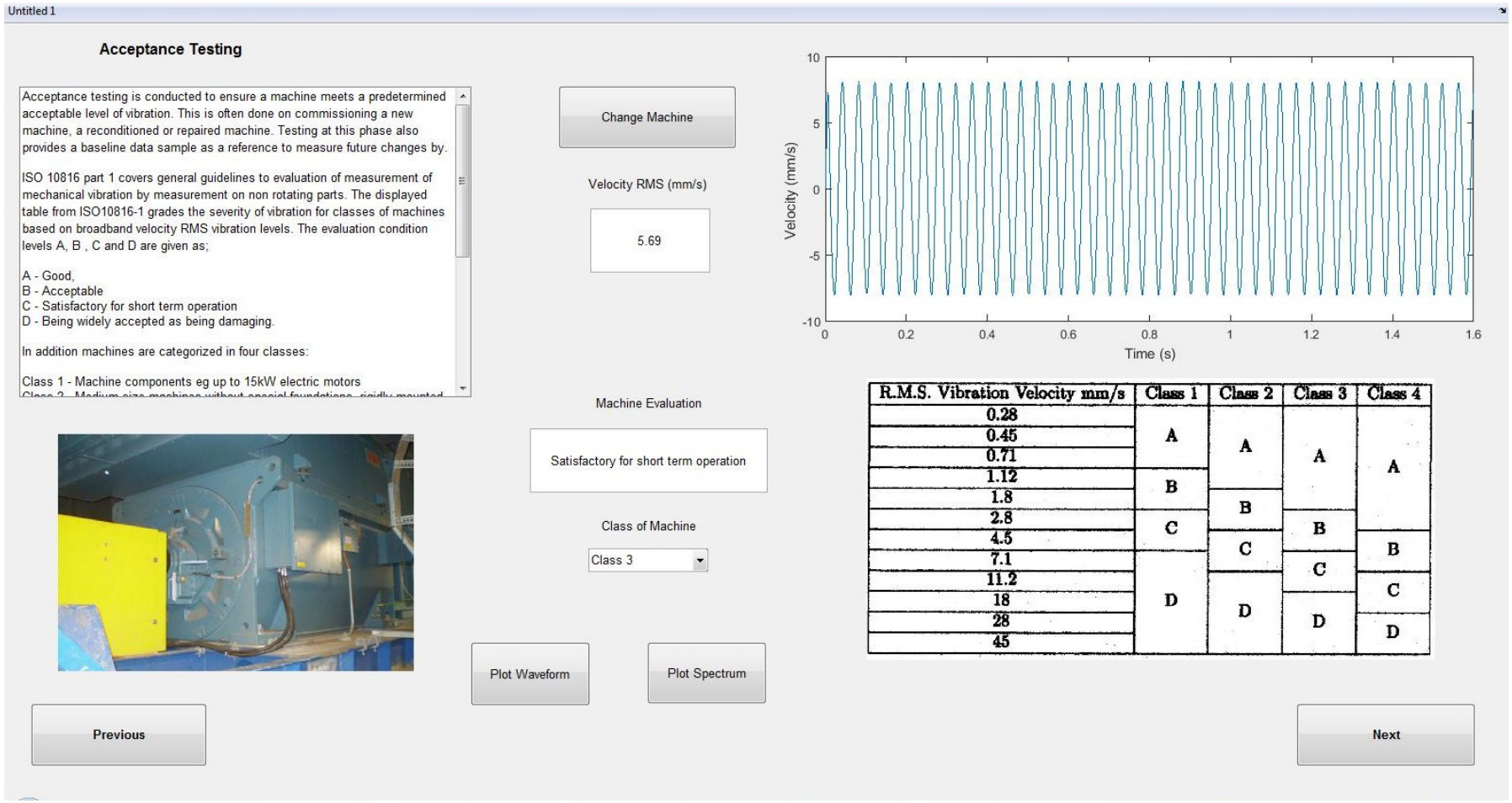


Figure 4.11: The Level_1_9 GUI introduces the user to acceptance testing. It allows the user to select a machine class based the given ISO10816-1 table. By using the "Change Machine" pushbutton an unbalance mass is randomly generated and the response of a single degree of freedom system is plotted. Based on the RMS level the machine evaluation condition is displayed. A machine image of the class of machine selected is displayed. The associated waveform and spectrum can be displayed in separate Matlab figures for closer inspection.

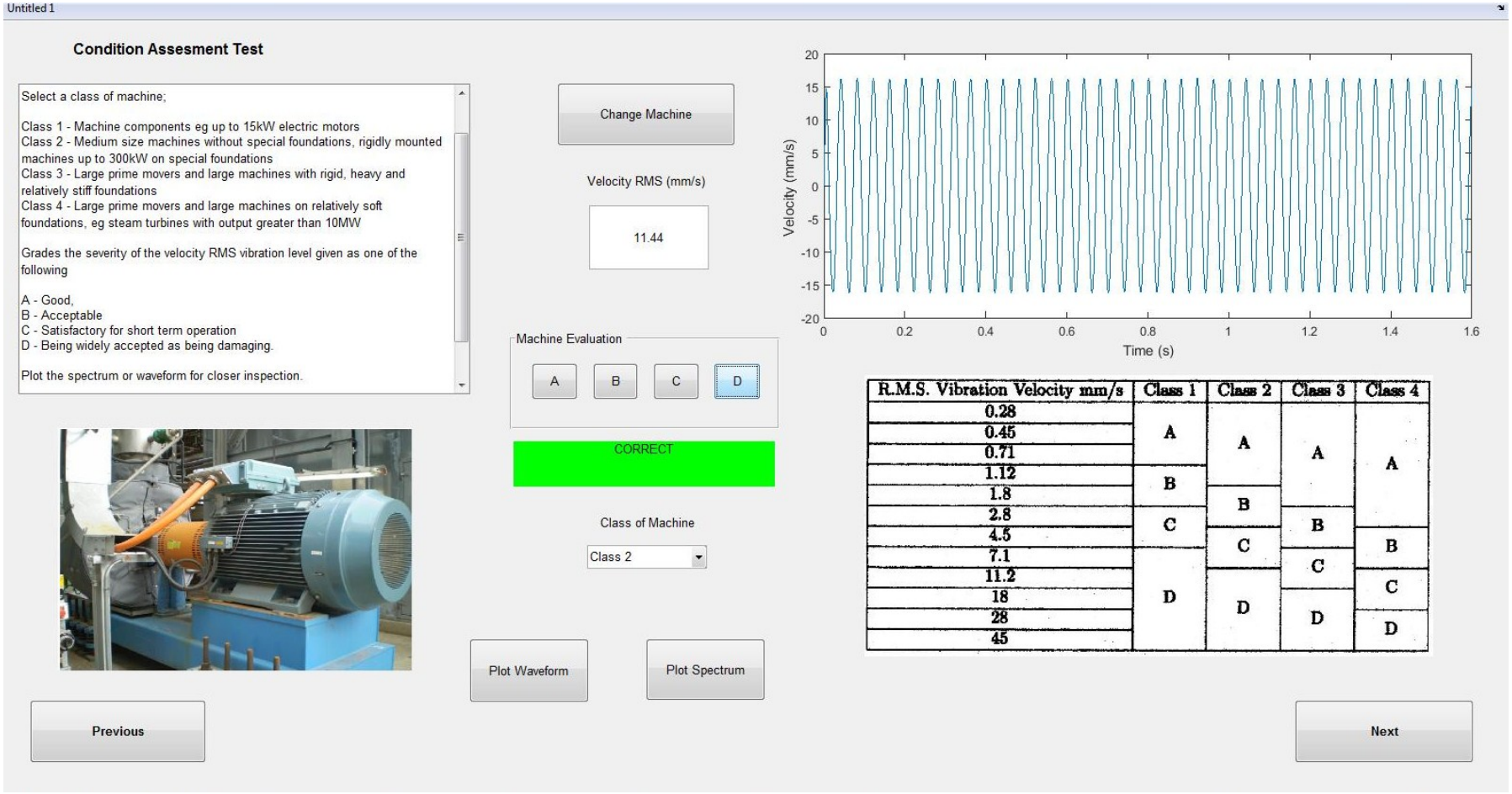


Figure 4.12: The Level_1_10 GUI allows the user to select a machine class based the ISO10816-1 table. By using the "Change Machine" pushbutton an unbalance mass is randomly generated and the response of a single degree of freedom system is plotted. The user can then evaluate the machine vibration level by selecting A-Good, B-Acceptable, C-Satisfactory for short term operation or D-Being widely accepted as damaging. The box below the selection pushbuttons will light up green if correct and red if incorrect. The associated waveform and spectrum can be displayed in separate Matlab figures for closer inspection

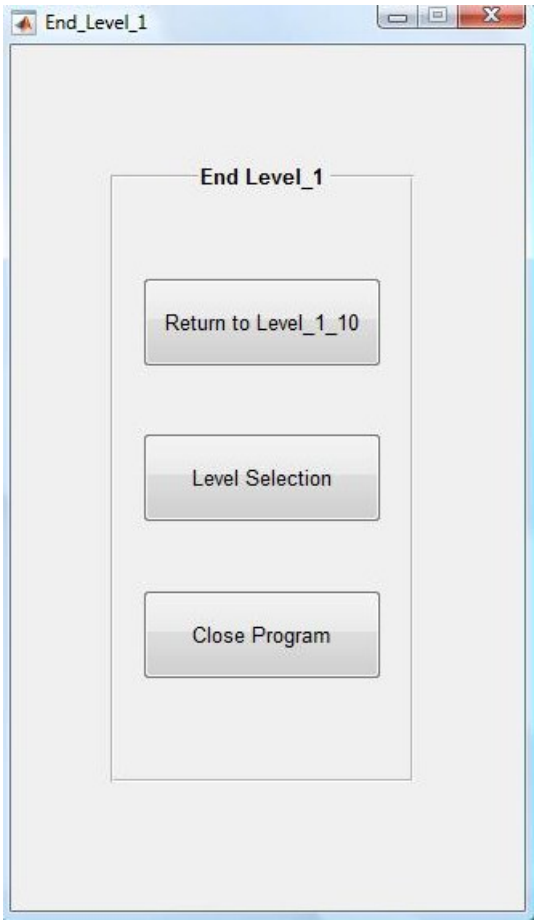


Figure 4.13: This shows the End_Level_1 GUI. It allows the user to open the Level Selection GUI, return to Level_1_10 or close the program at use of a pushbutton.

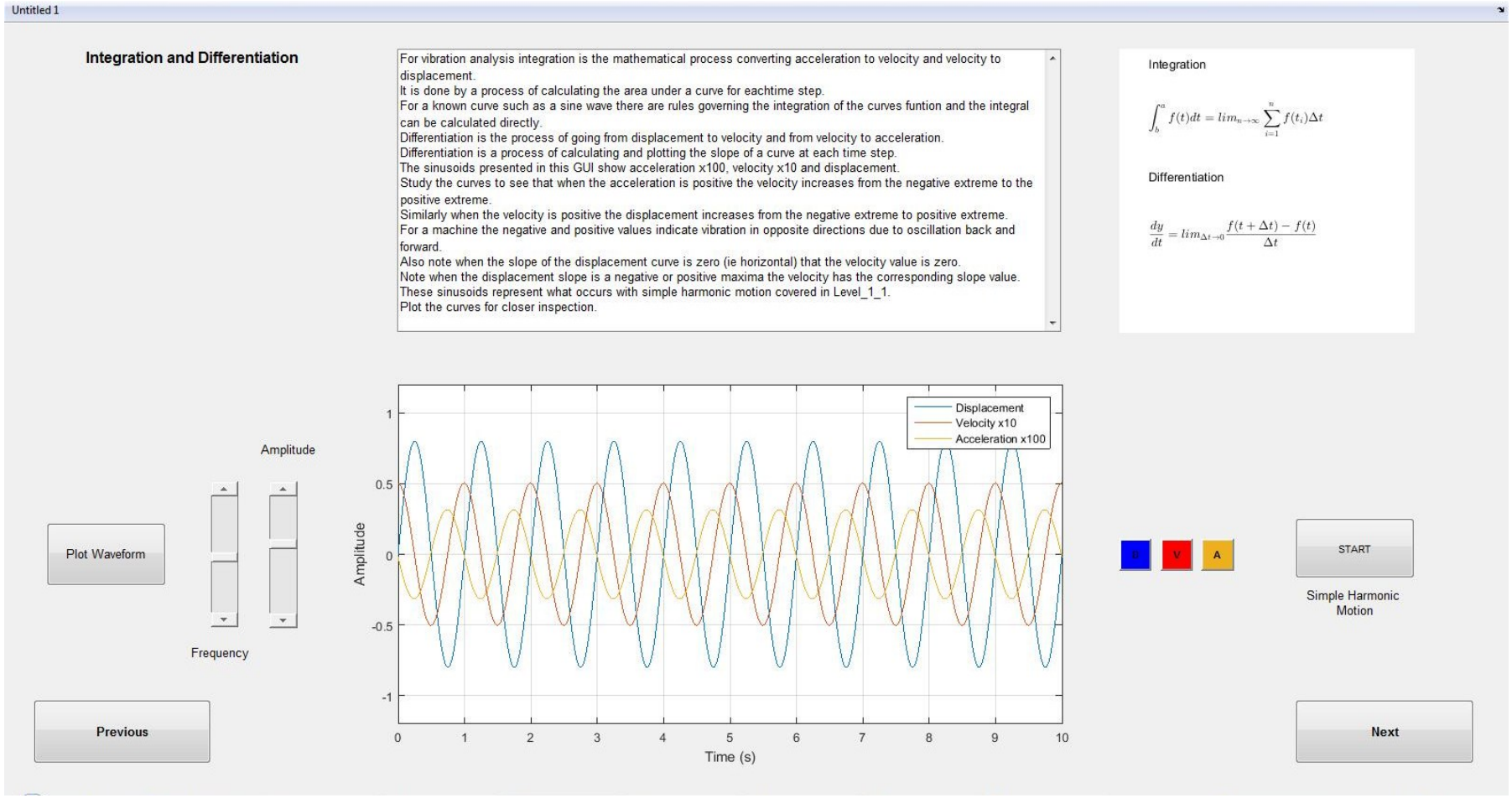


Figure 4.14: The Level_2_1 GUI introduces the user to the concepts of integration and differentiation. As well as an explanation of the processes and showing equations for each the GUI demonstrates it by allowing the user to visualize the relationship between displacement (D), velocity (V) and acceleration (A) shown as sinusoids. Sliders allow the user to adjust the frequency and amplitude. The start pushbutton begins an animation of the blocks D, V and A to further show the phase and amplitude relationships. The waveform displayed in the GUI can be plotted as a separate figure for closer inspection

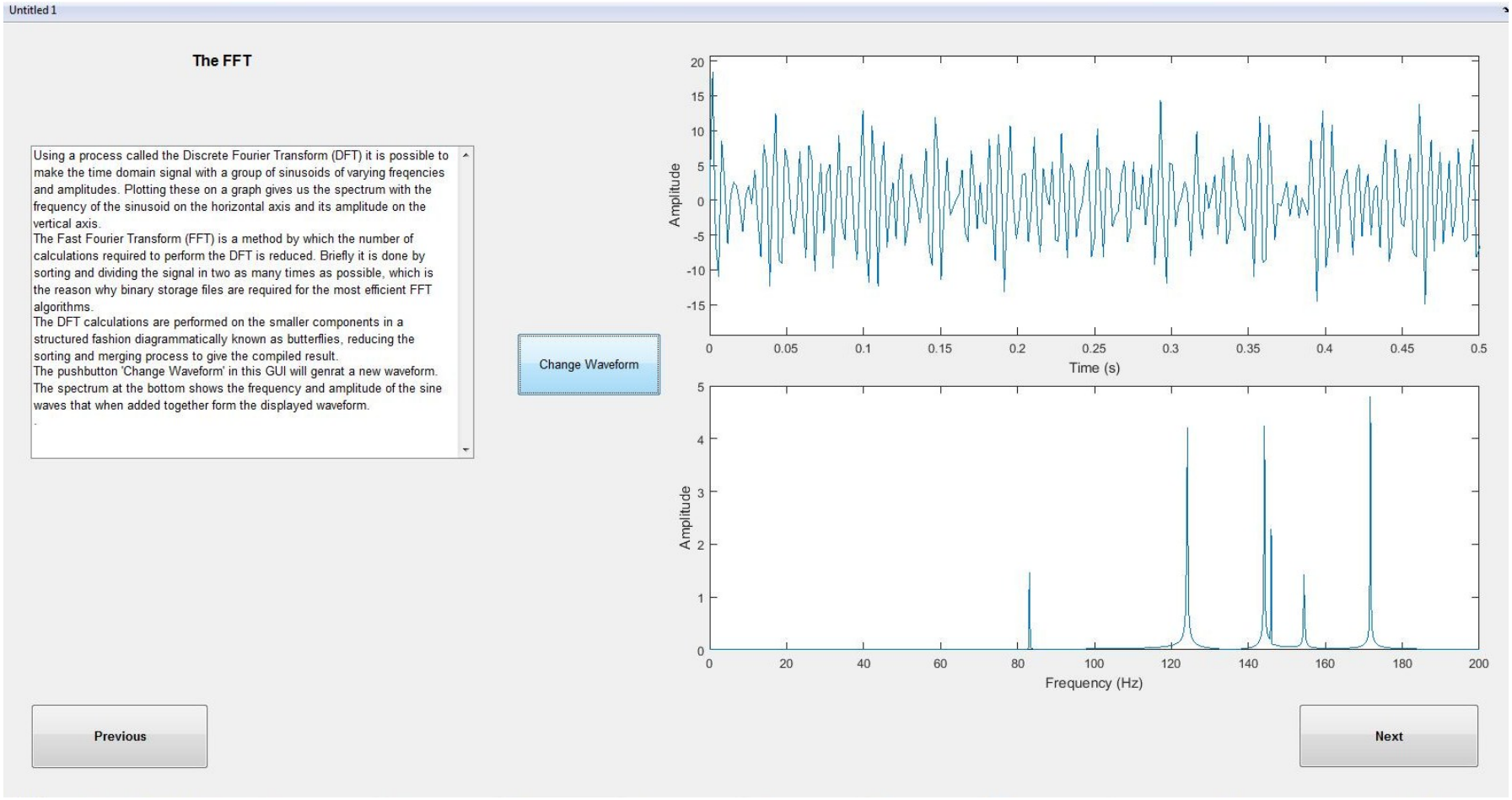


Figure 4.15: The Level_2_2 GUI introduces the user to the fast Fourier transform. It offers a simple explanation of the process, show a randomly generated waveform, conducts the fft on that waveform and graphs it constituents in the frequency spectrum, each peak representing a sinusoid in frequency and amplitude. The change waveform pushbutton randomly generates another waveform with the same FMax and LOR settings

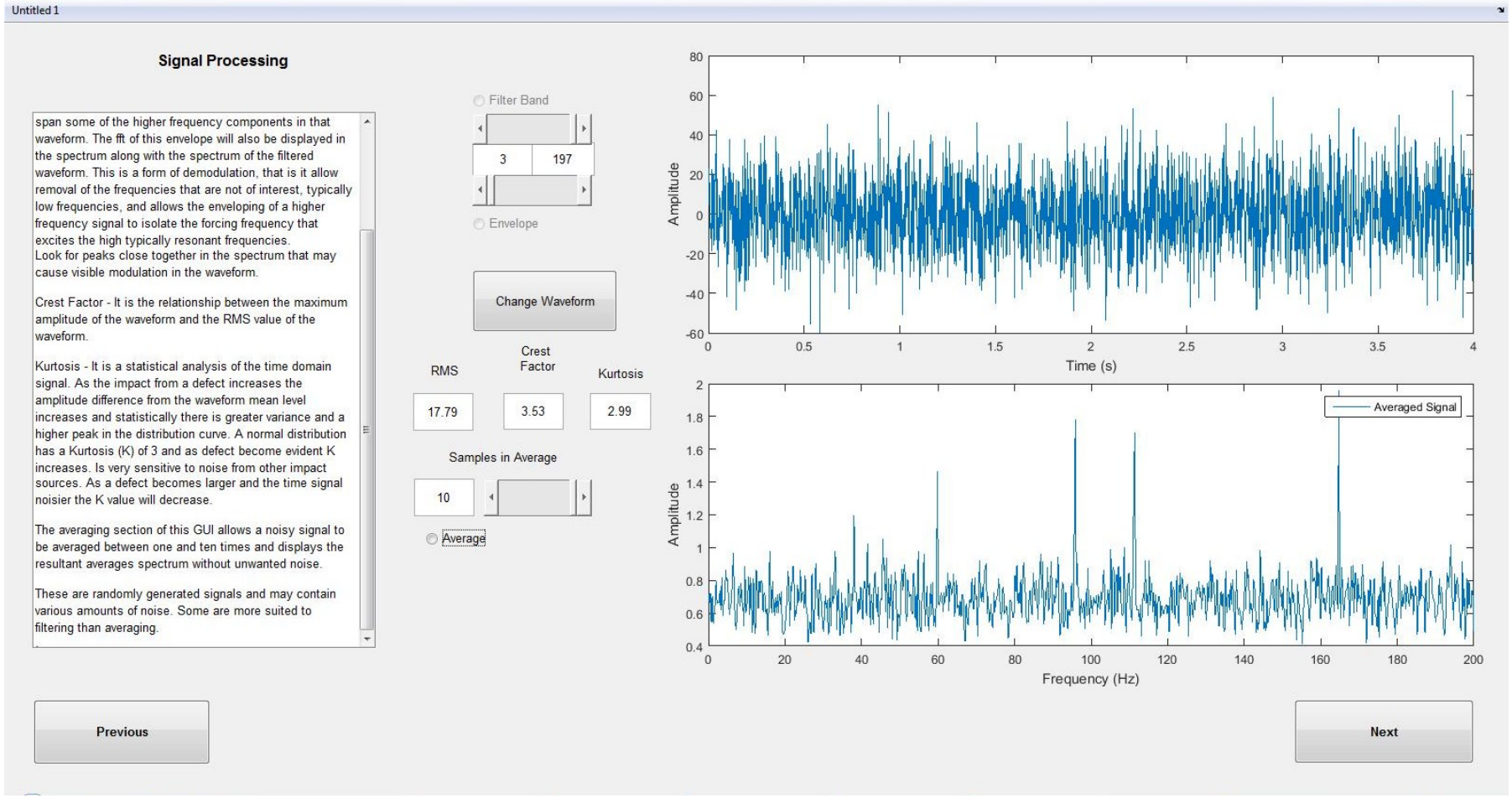


Figure 4.16: The Level_2_3 GUI introduces the user to the frequency band filtering and enveloping in conjunction to display the effect of demodulation. It also has the function of introducing averaging to reduce the amount of noise in the averaged spectrum and better separate the frequencies of interest from the noise, helping find the needle in the haystack.

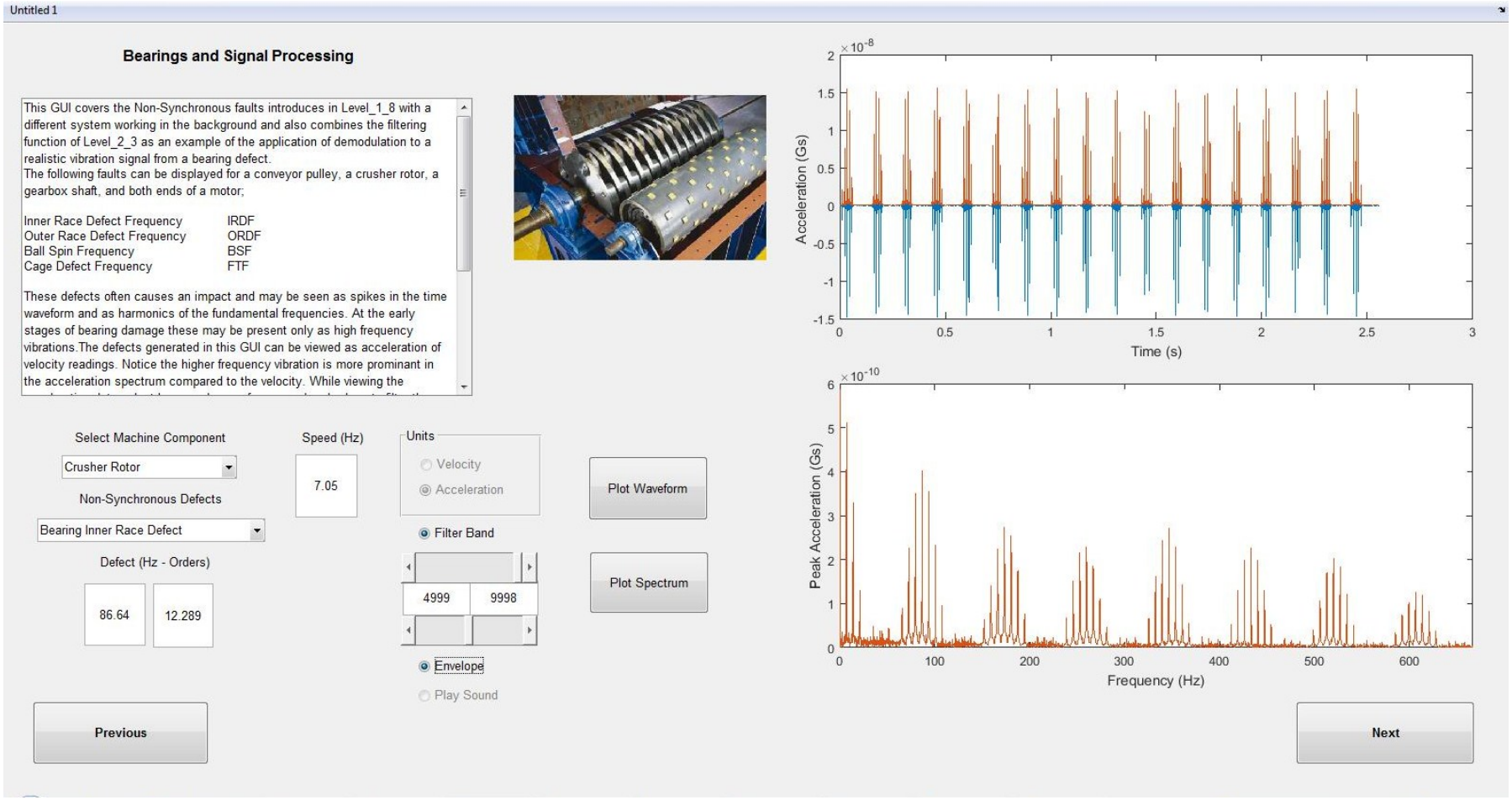


Figure 4.17: The Level_2_4 GUI combines the filtering function of Level_2_3 and the non-synchronous bearing faults introduced in Level_1_8 with a two degree of freedom system working in the background and provides an example of the application of demodulation to a realistic vibration signal from a simulated bearing defect. This image shows a filtered waveform with envelope in orange and the spectrum from the fft of the envelope. The spectrum shows the defect rate harmonics with shaft speed sidebands. Sound for the acceleration waveforms can be played.

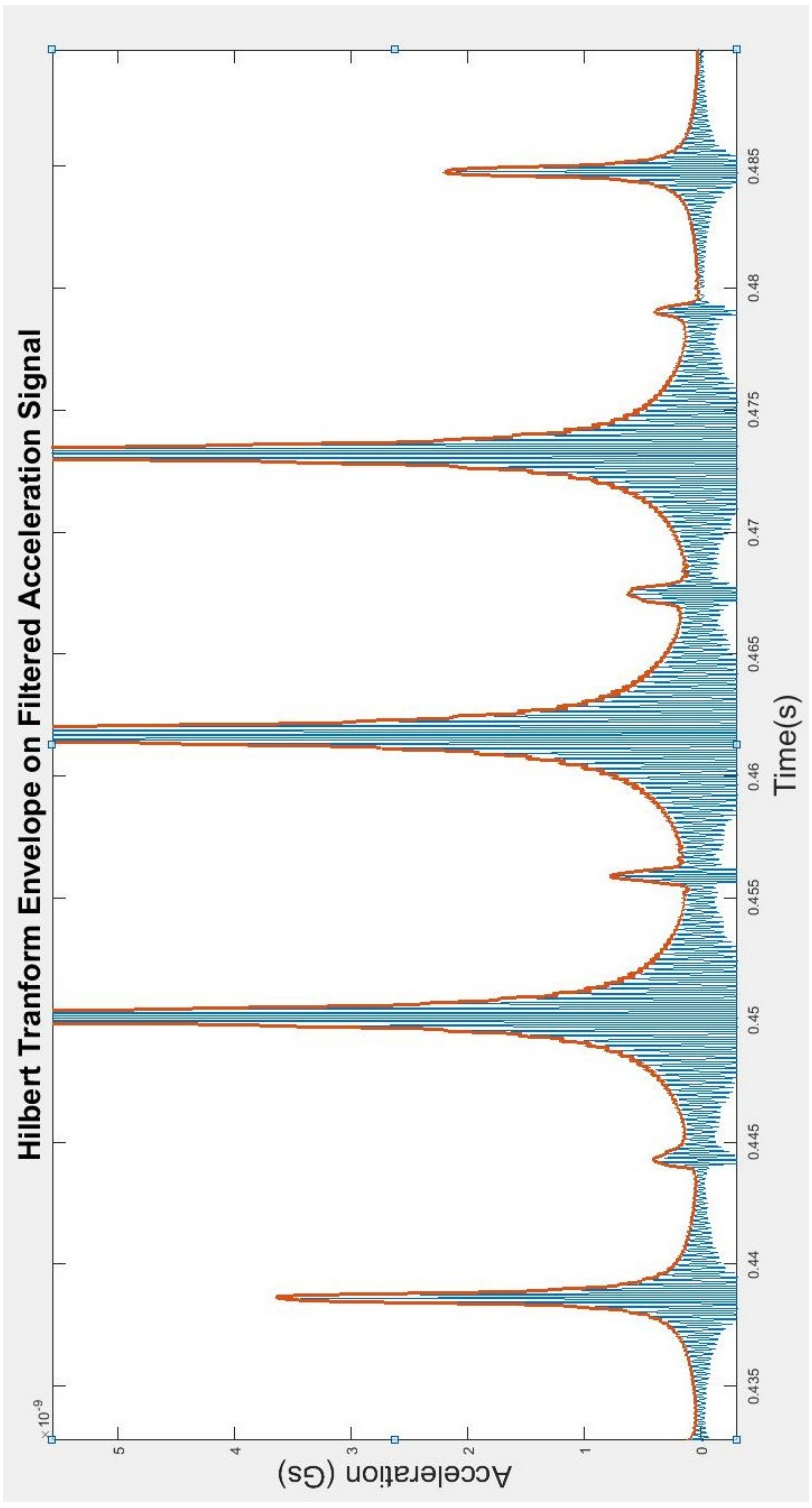


Figure 4.18: This figure shows a close up of the enveloped waveform in the GUI of Level_2_4. It is a 5000-10000Hz filtered signal (in blue) of a simulated inner race defect and shows the Hilbert transform as an envelope (in orange) on the waveform.

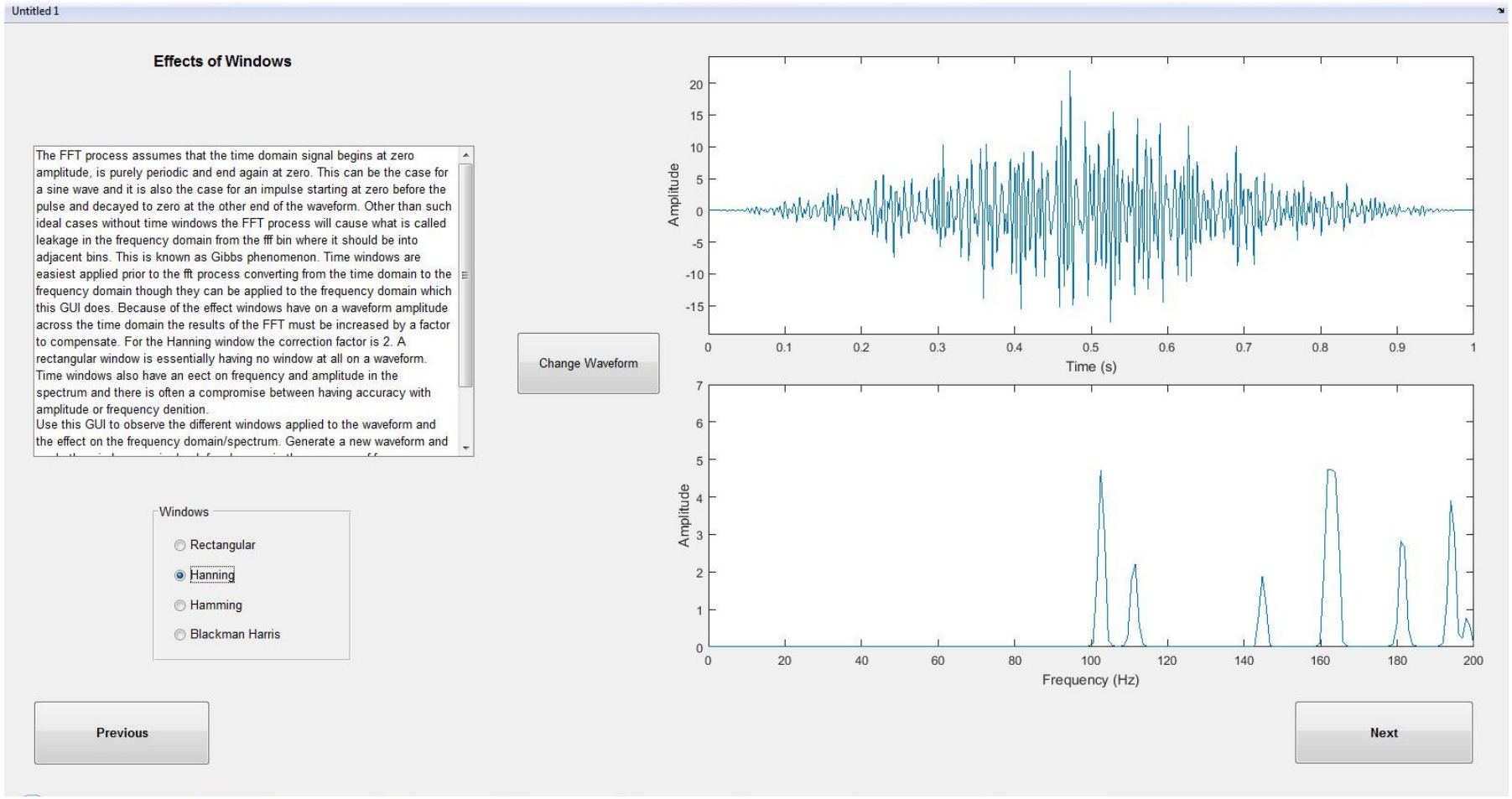


Figure 4.19: This figure shows the GUI of Level_2_5. A waveform is randomly generated and different windows are applied to the waveform and resulting frequency spectrum is displayed. Changes in the waveform are observed dependent on the window shape and changes to accuracy in the representation of frequencies and amplitudes can be observed in the frequency spectrum.

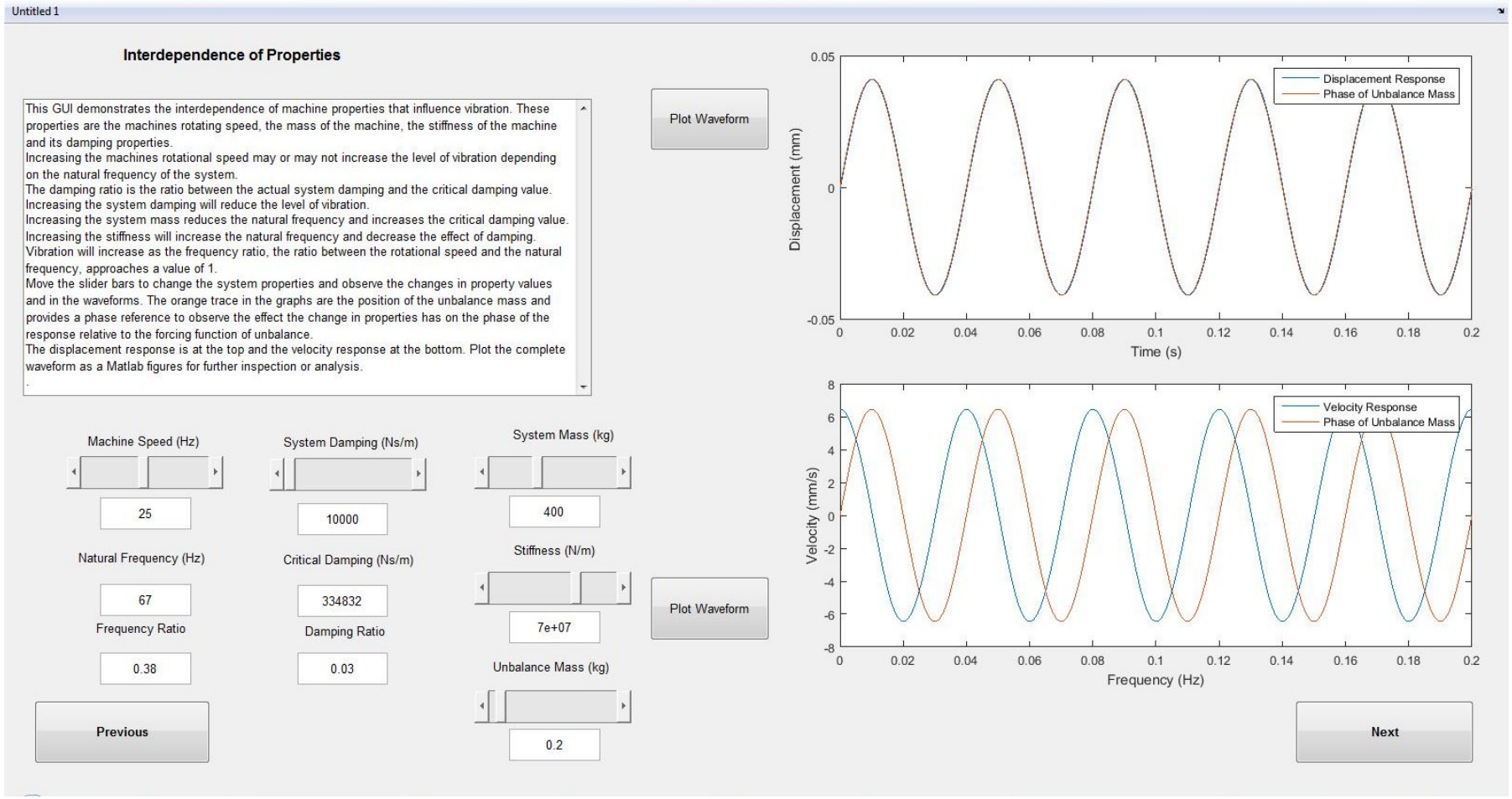


Figure 4.20: This figure shows the GUI of Level_2_6. This GUI works on a single degree of freedom system where the user can change the machine rotational speed, its damping factor, system mass, stiffness and the unbalance mass. The changes in natural frequency, frequency ration, critical damping value and damping ratio are output to the GUI screen. The displacement and velocity waveform responses are plotted relative to the phase of the unbalance mass .

Chapter 5

Conclusion and Further Work

5.1 Conclusion

The primary goal of developing a vibration training and awareness program was to create a tool to familiarize clients, plant owners and technicians with the techniques used in vibration condition monitoring of rotating machinery. The program has been created by researching condition monitoring practices, international standard and vibration theory on the subject. The program is the result of including the key aspects of this research in a Matlab program and with graphical user interfaces to operate the program in the background the user has an interactive learning tool. The user can make changes in the displayed system to data collection parameters directly or by selecting a range of common faults for output of a simulated vibration response. This is done to familiarize the user with the common form of vibration data and to provide a virtual view of a machine in terms of vibration patterns. With explanation of concepts, changes the user makes with on screen controls produce an observable change. The user has option to plot data for analysis beyond that provided by the graphical user interface. In this way the user can gain an improved understanding of vibration monitoring from basic concepts to signal processing and the interdependence of the properties effecting the vibration response. This understanding will help the technician and plant owner better understand the role vibration analysis can play in detecting developing machine faults, helping them prevent expensive plant downtime.

5.2 Further Required Work

The following is work that is required to complete the level 1 and level 2 sections of the program as well as some issues with it in its current form.

Across Program

- There appears to be some letter or number issues in GUI .txt file possibly caused going from Windows Vista to Windows 10. A complete inspection of GUI text and its spelling is required.
- Inclusion of code 'evalin('base','clear')' in more GUIs to clear the workspace when opening

- Include more waitbars in the program
- Completion of the checklist for comprehensive review of the software.
- A program help package describing the function, operation and specification of the system has not been compiled.

Level_1

- The user at this stage is not given information on available transducers; rather the program is focused on the output of transducers in a horizontal plane. Simulating data acquisition with a simulated data collector, uploading and download data is not covered by the program for level 1 user.
- There is a fault in Level_1_5 where a slider disappears, possibly going outside range. It happens when another slider control is used.
- Text in Level_1_7 only covers three of faults presented in the GUI, unbalance misalignment and looseness. There is no text related to gear damage, vane pass, pump cavitation or friction. Images are displayed only for the unbalance and misalignment cases. Friction is not set up to give a plotted response Level_1_7.
- Level_ 1_8 code does not allow for a time delay if a transient is present in the initial response of the single degree of freedom system.

Level 2

- Setting of pass filter lower and upper frequency bands needs to be managed by code so the user cannot set them out of bounds.
- The two degrees of freedom system in Level_2_4 is slow and could be sped up by changing the LOR and FMax. For the concepts of this level the established single degree of freedom system could also be used. The two degree of freedom model was used in Level_2_4 with the plan of achieving a system with a low frequency response and a high frequency response, one to simulate a relatively low structural response and the other the high frequency response to an early stage defect to be used in enveloping. Further work is required on this.

Further desirable work

More selectable components, drivers, intermediaries, and driven components and combining these in a system for which the vibration response to fault forces could be simulated is yet to be developed. This would add the level of complexity with multiple degrees of freedom where a vibration from the driven component is transferred through the structure to the drive component. This would be along the lines of the coplanar balancing with a rigid shaft that is required as a part of a complete Level 2 training package. The system as it is currently written in Matlab does not accommodate this degree of complexity.

Also desirable functions are

- Provide for simulated response to be fed as a output signal to a real data collector to record the signal.
- Provide the user with a tactile sense of the simulated vibration signal.

References

Ahmadi, A. M., Petersen, D. & Howard, C. (2015), 'A Nonlinear Dynamic Vibration Model of Defective Bearings - The Importance of Modelling the Finite Size of Rolling Elements', *Mechanical Systems and Signal Processing* **52-53, Issue 1**, 309–326.

Balachandran, B. & Magrab, E. B. (2009), *Vibrations*, 2nd edn, Cengage Learning, Toronto, Canada.

Bureau Veritas (2008), 'Equipment photos and vibration traces extracted from database files, Singleton NSW office'.

Commtest New Zealand (2015), 'Products portable solutions'. `http://www.commtest.com/products/portable`.

Drew, T. (1999), 'Get Ready for More Soot: Is Your Soot Monitoring Program Up to the EPAs New Challenge?', *Practicing Oil Analysis Magazine* pp. 17–22. `http://www.machinerylubrication.com/Read/223/photometry-engine-oil-soot`.

Emerson Process Management (2015), *CSI 2140 Machinery Health Analyzer*. `http://www2.emersonprocess.com/siteadmincenter/PM%20Asset%20Optimization%20Documents/ProductReferenceAndGuides/2140_ref_UserGuide.pdf`.

Engineers Australia (2010), Code of Ethics, Reference, Engineers Australia. `https://www.engineersaustralia.org.au`.

Institute, M. (2010), *Vibration Training Course Book Category 3*, Mobius Institute, chapter Appendix B, ISO Standards. Revision 31-03-10.

ISO (2014), Condition Monitoring and Diagnostics of Machines Requirements for Qualification and Assessment of Personnel Part 2: Vibration Condition Monitoring and Diagnostics, Standard ISO18436-2:2014, International Organization for Standarization. `http://www.saiglobal.com.ezproxy.usq.edu.au/online/autologin.asp`.

LDS Group (2003), Understanding FFT Windows. 1203http://www.physik.uni-wuerzburg.de/~praktiku/Anleitung/Fremde/ANO14.pdf.

Lin, Ran, T., Tan, Andy, Howard, Ian, Pan, Jie, Crosby, Peter, Mathew & Joseph (2011), Development of a Diagnostic Tool for Condition Monitoring of Rotating Machinery, *in* 'ICOMS Asset Management Conference Proceedings', Asset Management Council Limited, Oakleigh, VIC, Australia, pp. 1–9. http://eprints.qut.edu.au/41753/.

Miller, J. L. & Kitaljevich, D. (2000), In-line Oil Debris Monitor for Aircraft Engine Condition Assessment, *in* 'Aerospace Conference Proceedings, Big Sky, Montana, USA', Vol. 6, Institute of Electrical and Electronics Engineers, MT, USA, pp. 49–56. http://www.ewp.rpi.edu/hartford/~ernesto/F2006/EP/Aids/Papers/Figueroa-Rodriguez/Miller.pdf.

Mohanty, R. A. (2014), *Machinery Condition Monitoring Practices and Principles*, CRC Press, FL. USA, pp. 82–86. https://books.google.com.au/books?id=TXXSBQAAQBAJ&pg=PP1&dq=M%7Dachinery+%7BC%7Dondition+%7BM%7Donitoring+%7BP%7Dractices+and+%7BP%7Drinciples&hl=en&sa=X&ved=0CDEQ6AEwAGoVChMIuc2c3qHfxwIVoimmCh3Y5gCd#v=onepage&q=M%7Dachinery%20%7BC%7Dondition%20%7BM%7Donitoring%20%7BP%7Dractices%20and%20%7BP%7Drinciples&f=false.

Moubray, J. (1997), *Reliability Centred Maintenance*, 2nd edn, Industrial Press Incorporated, New York, New York.

NSK Ltd (2015), *Bearing Internal Load Distribution and Displacement.* http://www.nsk.com/services/basicknowledge/technicalreport/05distribution.html.

Osgood, B. (2007), *The Fourier Transform and its Applications*, Stanford University, chapter 6. https://see.stanford.edu/materials/lsoftaee261/book-fall-07.pdf.

Safe Work Australia (1991), Ergonomic Principles and Checklists for the Selection of Office Furniture and Equipment, Reference, Safe Work Australia. http://www.safeworkaustralia.gov.au/sites/SWA/about/Publications/Documents/31/Ergonomic.

Safe Work Australia (2013), Guide for Managing the Risk of Fatigue at Work, Reference, Safe Work Australia. http://www.safeworkaustralia.gov.au/sites/SWA/about/Publications/Documents/825/Managing-the-risk-of-fatigue.pdf.

Rockwell Automation (2015), *XM120 and XM121 Dynamic Measurement Modules*. http://www.ab.com/en/epub/catalogs/5668943/5675109/5675111/5675276/index.html.

Scheffer, C. & Girdhar, P. (2004), *Practical Machinery Vibration Analysis and Predictive Maintenance*, revised edn, Elsevier, chapter 3, pp. 29–30.

SKF USA (2014), *Condition Monitoring Essentials*. http://www.skf.com/binary/101-48656/CM-P1-11704-6-EN-Condition-Monitoring-Essentials-Catalog.pdf.

Wiggelinkhuizen, E., Verbruggen, T., Braam, H., Rademakers, L., Xiang, J. & Watson, S. (2008), 'Assessment of Condition Monitoring Techniques for Offshore Wind Farms', *Journal of Solar Energy Engineering* **130**. http://www.ewea.org/ewec2007/allfiles2/136_Ewec2007fullpaper.pdf.

Appendix A

Project Specification

University of Southern Queensland

FACULTY OF ENGINEERING AND SURVEYING

ENG4111/4112 Research Project

PROJECT SPECFICATION

FOR:	Richard Amos Little
TOPIC:	CONDITION MONITORING OF ROTATING MACHINERY - VIBRATION ANALYSIS
SUPERVISOR:	Robert Fulcher
PROJECT AIM:	Predicting failure problems with rotating machines using condition monitoring techniques. Determine faults most readily detected using vibration analysis and create a virtual software package for awareness and training purposes.
PROGRAM:	Version 4, 18/3/15

1. Research condition monitoring (CM) techniques, and associated theory, used with rotating machinery
2. Research suitability and success of various CM techniques in predicting failure problems with various types of rotating machinery.
3. Grade the success of various CM techniques to machine classes based on findings from literature review.
4. Research safety issues associated with condition monitoring of machines. Section 10.7 and appendix C of the project resource book (PRB).
5. Cover implications, consequential effects and any ethical issues related to the project and potential outcomes. Section 10.5 and appendix H of PRB.
6. Define desirable features and learning outcomes from use of a training unit.
7. Research the suitability and availability of software packages to create a virtual training unit. See section 10.8 of PRB.
8. Complete the literature review of evidence supporting all research thus far including the citation of other work and provision of references at an international standard. See section 10.4, 11.8, appendix F of PRB and USQ library material.

9. SUBMIT PRELIMINARY REPORT JUNE 3 IN STYLE FOR DISSERTATION WITH ASPECTS COMPLETED FOR INSERTION DIRECTLY INTO DISSERTATION. Section 10.3 of PRB.
10. Design and create a virtual training unit based on critique of research and literature review.
11. Set out conclusions and recommendations based on project results with respect to initial objectives and with supporting evidence of outcomes.
12. Explore ideas for potential future work on the topic.
13. Prepare a presentation of project work for delivery to a university and industry audience as part of the annual project conference on USQ campus. 21-25 September 2015
14. Complete Dissertation. Section 11 and Appendix E of PRB.
15. SUBMIT DISSERTATION 29 OCTOBER 2015, APPENDIX G

AGREED:

R LITTLE	(Student)	R FULCHER	(Supervisor)
Date : 18/03/2015		Date : 18/03/2015	

Appendix B

Supporting Information

B.1 Vibration Plots

The following vibration waveforms and spectra are included to give reference to the patterns that defects create. This reference is for comparison with output from the modeled systems programmed into the of the virtual training program.

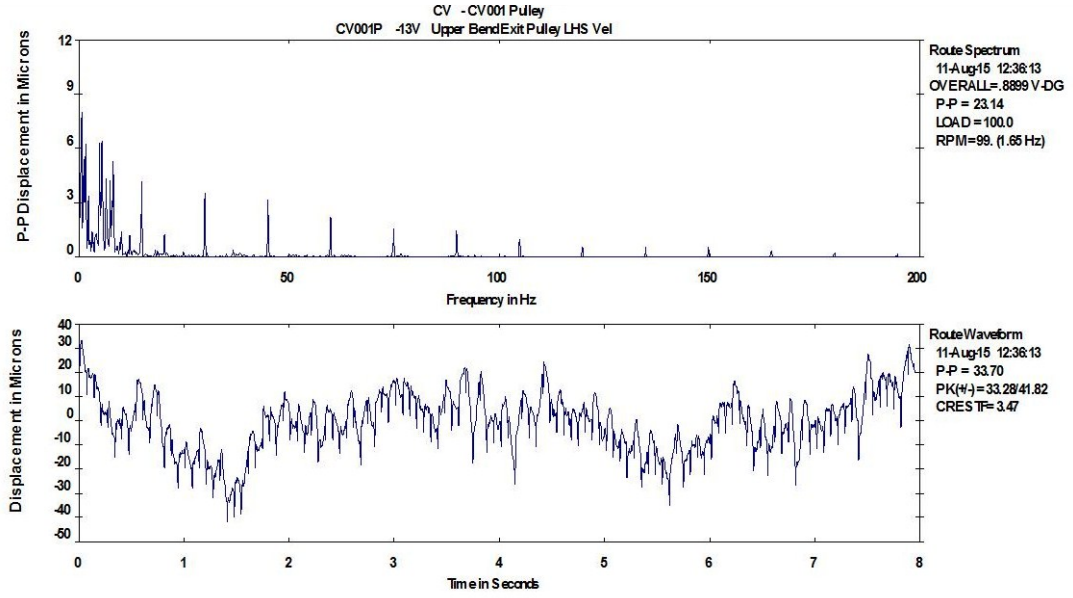


Figure B.1: This image shows a conveyor bend pulley outer race defect with the defect impact rate of 15Hz. The spectrum and waveform are in displacement units of microns. (Bureau Veritas 2008-2015)

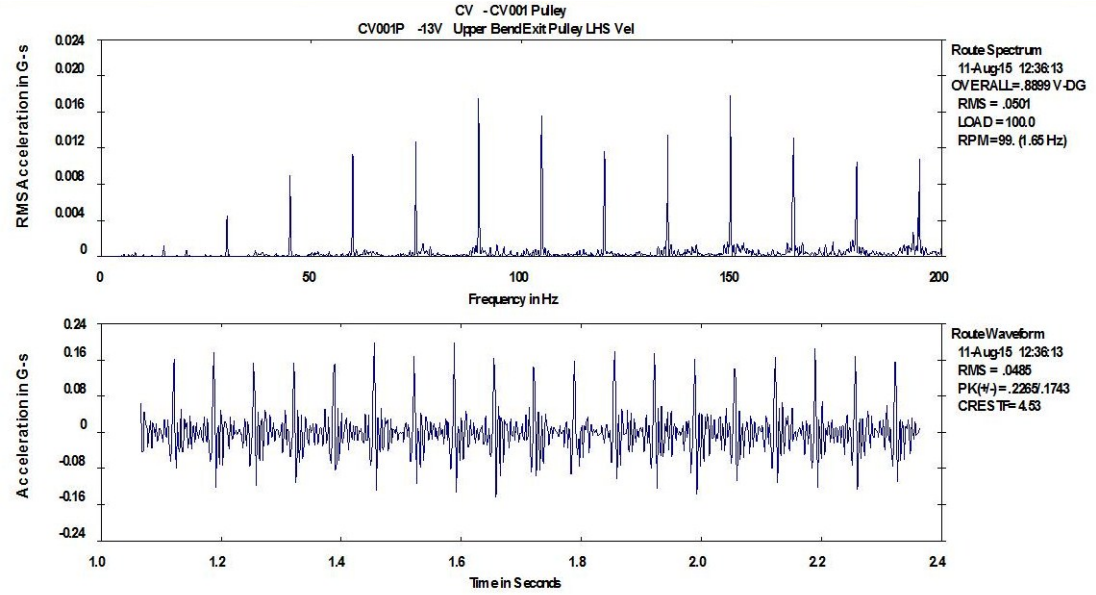


Figure B.2: This image shows the conveyor bend pulley outer race defect with the defect impact rate of 15Hz with spectrum and waveform in acceleration units of $G's$. (Bureau Veritas 2008-2015)

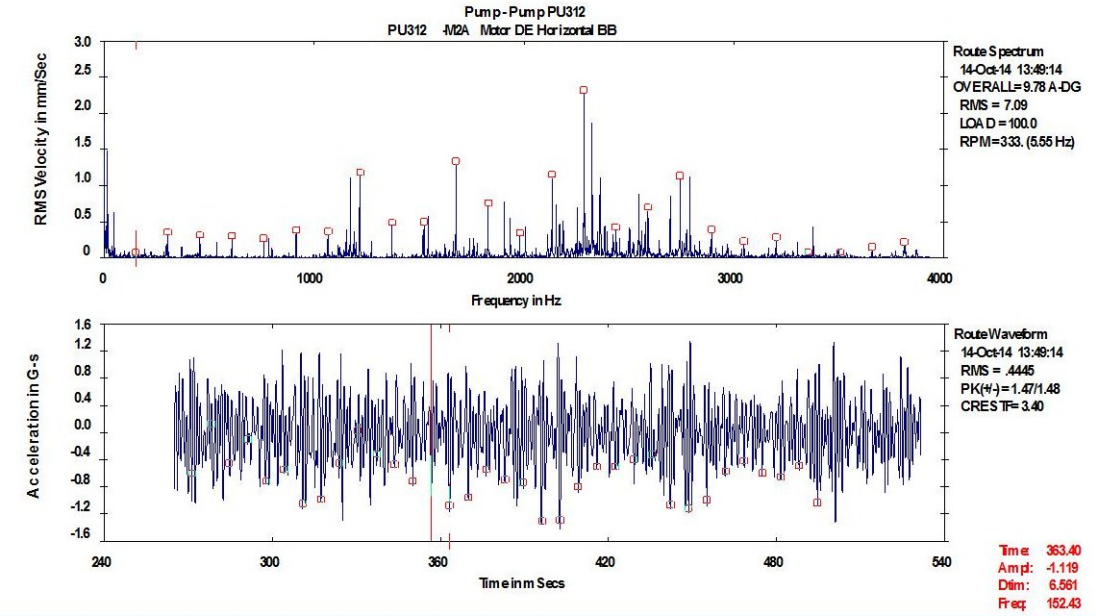


Figure B.3: This image shows a pump motor outer race defect with the defect impact rate of 154Hz. The spectrum shows approximately 0-4000Hz in velocity units of mm/s. (Bureau Veritas 2008-2015)

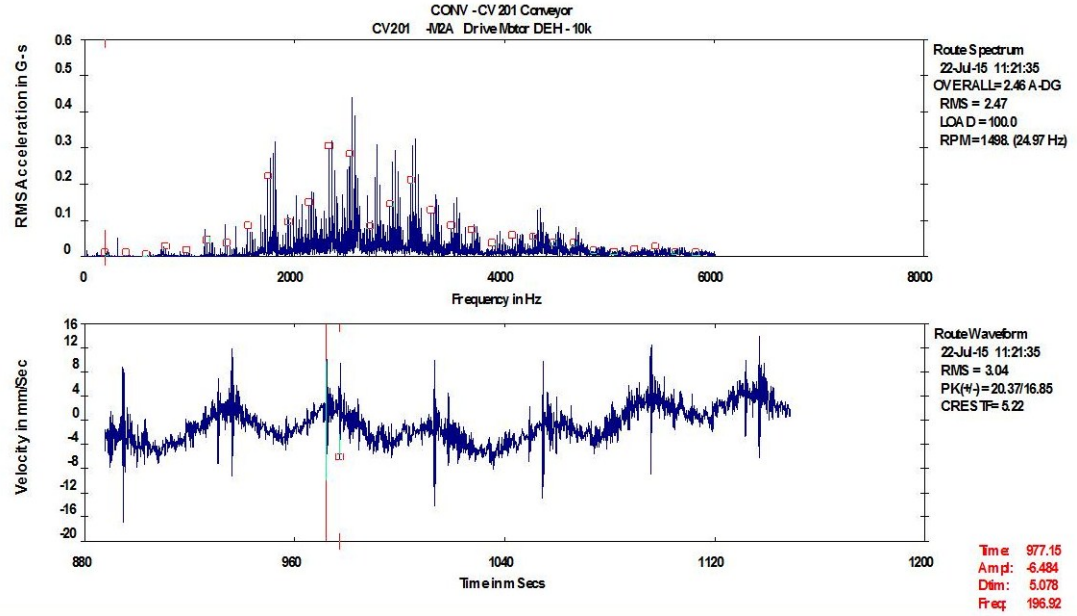


Figure B.4: This figure shows vibration from a pump 25Hz pump motor with an inner race defect at 97Hz with the spectrum in acceleration g's and the waveform in velocity mm/s. Note the sidebands in the spectrum and the impact modulation in the waveform. (Bureau Veritas 2008-2015)

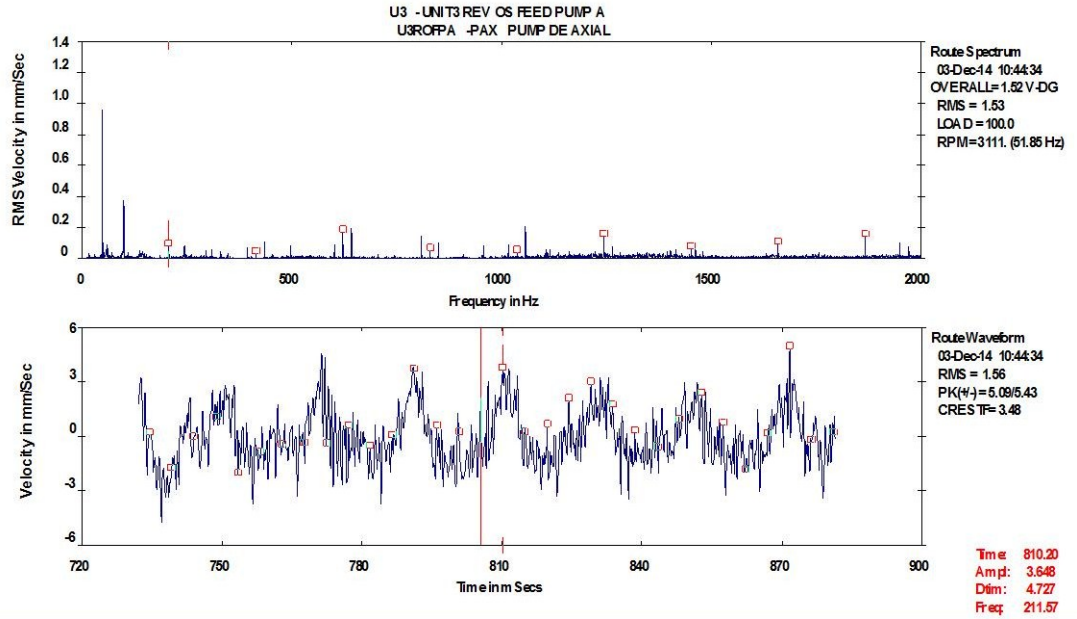


Figure B.5: This image shows a 50Hz pump with a bearing roller defect at a rate of 211Hz. The spectrum and waveform show 0-2000Hz in velocity units of mm/s. (Bureau Veritas 2008-2015)

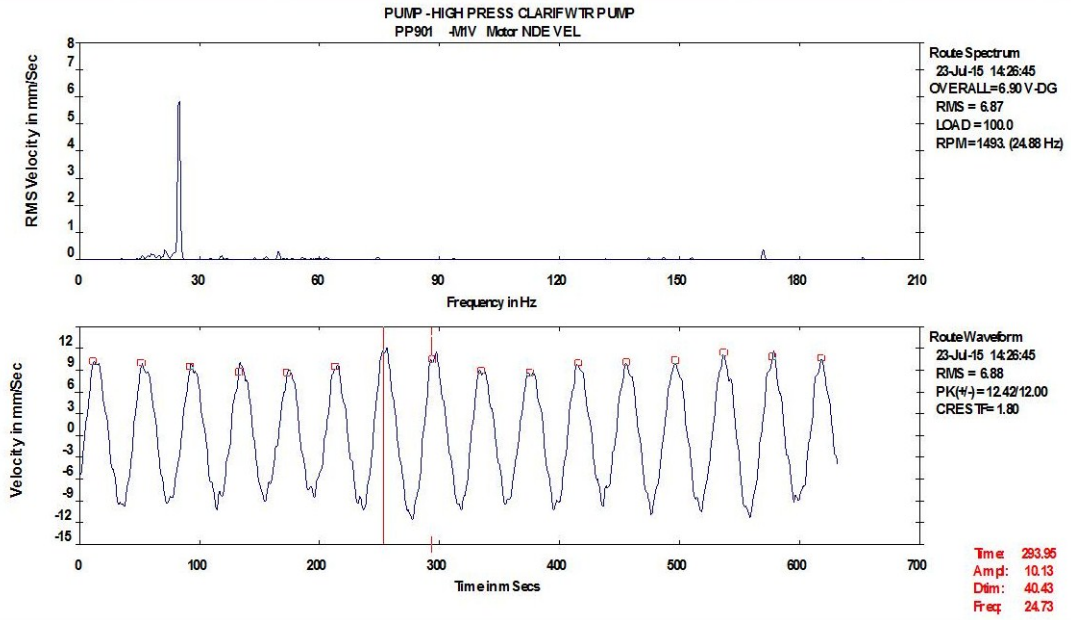


Figure B.6: This image shows a 25Hz pump motor with an unbalance vibration. The spectrum and waveform are in velocity units of mm/s. (Bureau Veritas 2008-2015)

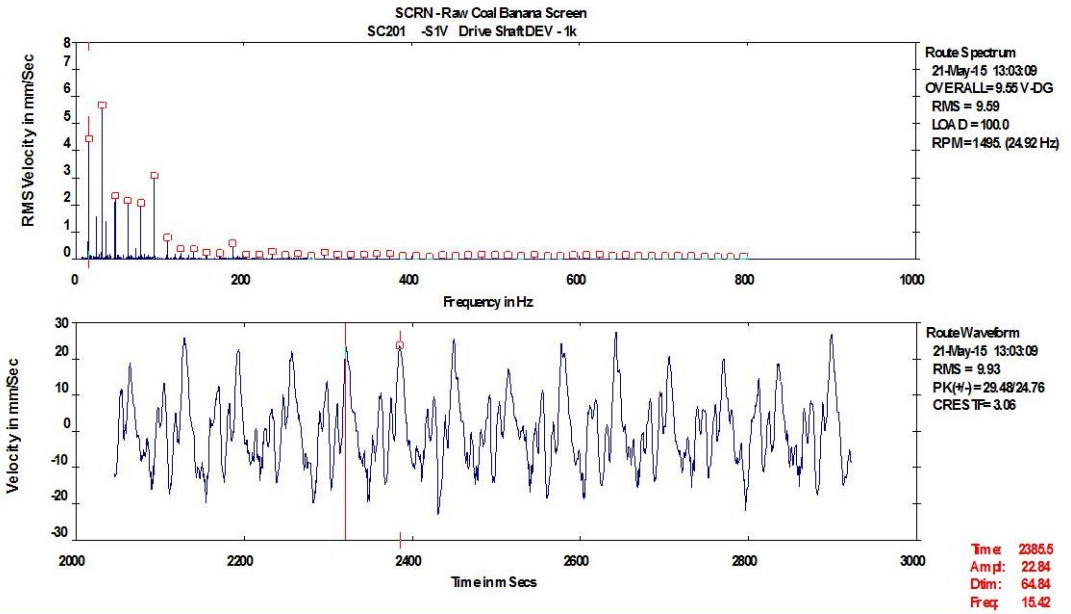


Figure B.7: This image shows looseness in a product screen jack shaft with multiple harmonics in the spectrum and impact in the waveform at run speed 15.4Hz. Units are velocity mm/s. (Bureau Veritas 2008-2015)

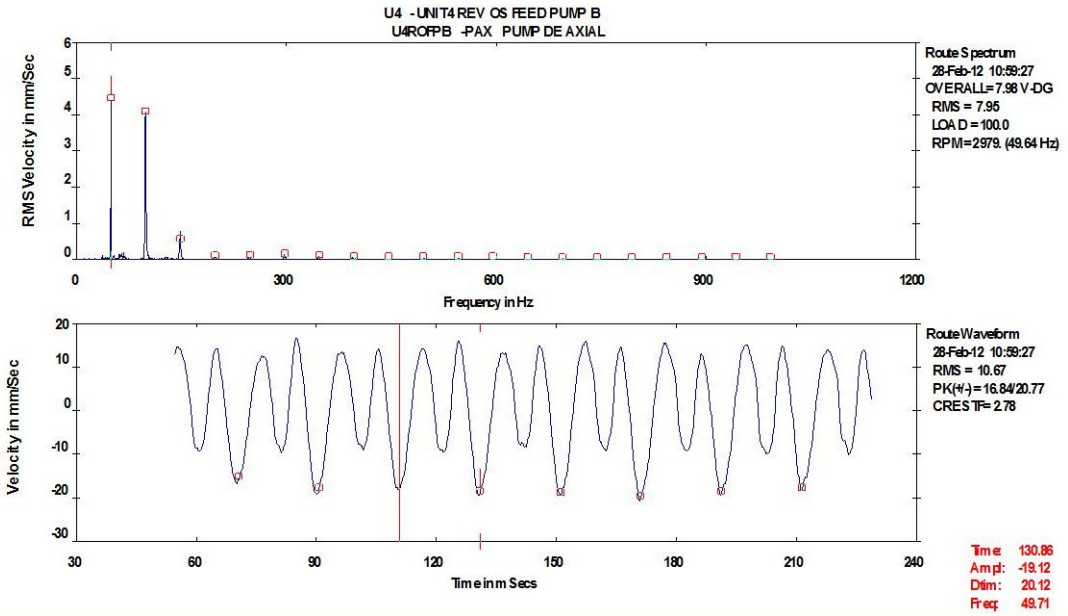


Figure B.8: This image shows an example of motor-pump misalignment in a 50Hz drive with the typical waveform pattern and elevated second harmonic in the frequency spectrum.(Bureau Veritas 2008-2015)

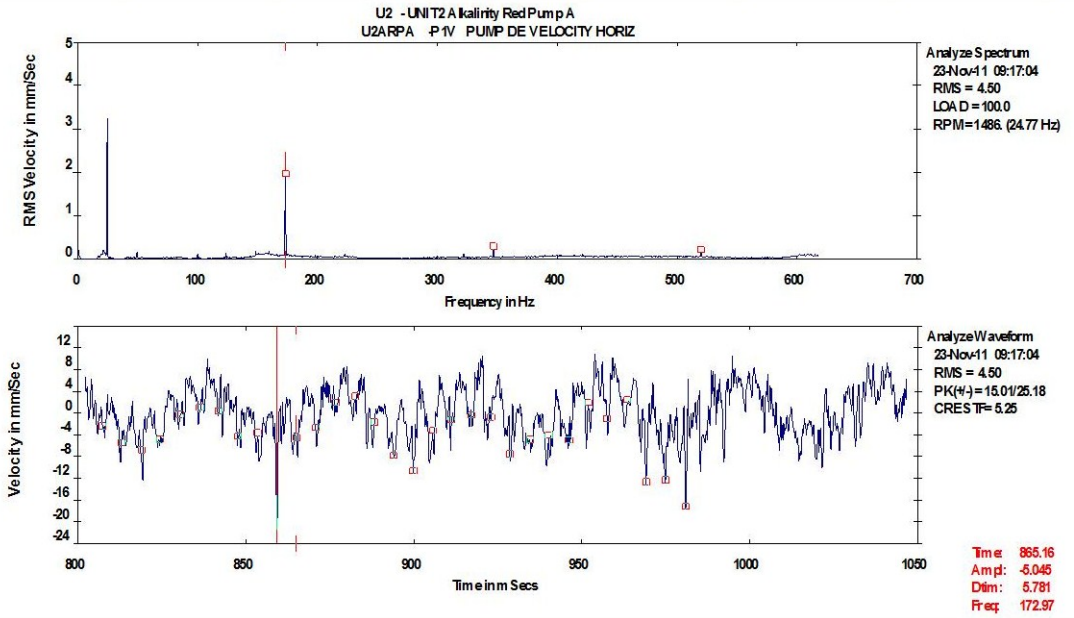


Figure B.9: This image shows a 25Hz pump with vane pass vibration at 7 orders of run speed highlighted in the spectrum and waveform. The sunits are velocity mm/s. (Bureau Veritas 2008-2015)

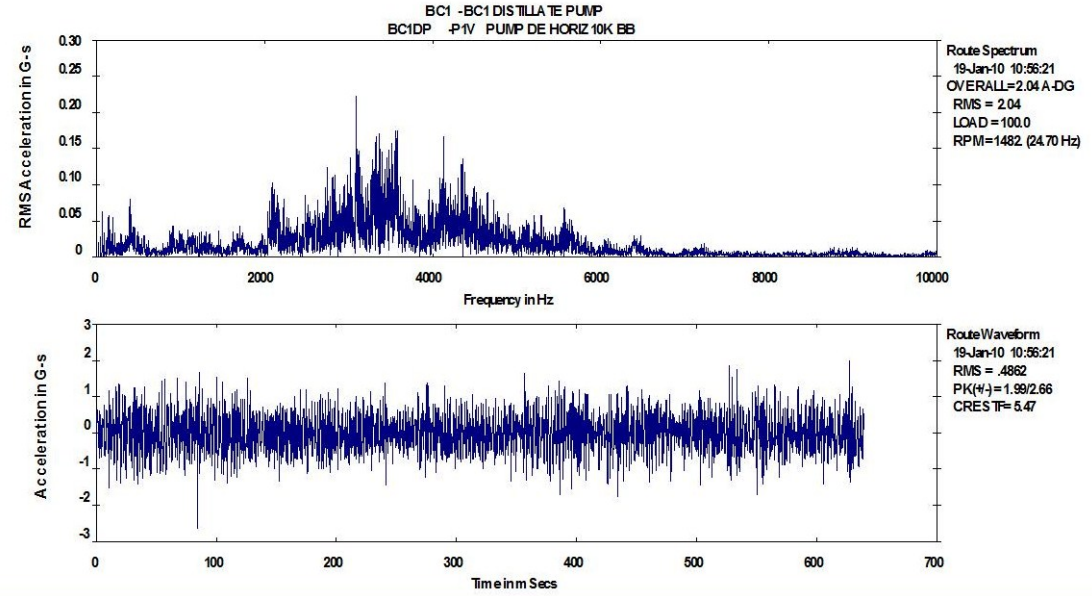


Figure B.10: This image shows a 25Hz pump displaying cavitation as random vibration across a broad frequency rage. The spectrum and waveform are in acceleration units of g's. (Bureau Veritas 2008-2015)

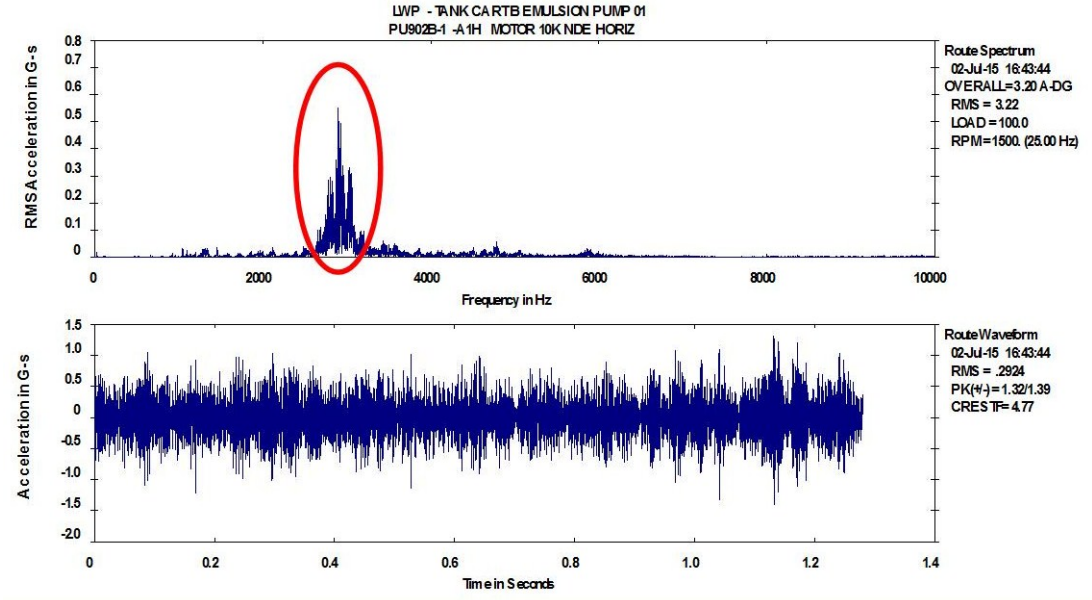


Figure B.11: This image shows a pump motor with friction as a concentrated area of elevated random vibration in the spectrum. The spectrum and waveform are in acceleration units of g's. (Bureau Veritas 2008-2015)

Appendix C

Program Code

C.1 Program Code Overview

This is Matlab code for training program. The general structure is raised once the GUI is built in the Matlabs graphical user interface development environment (GUIDE). The titles, function headings for opening the level, GUI controls, axes and text boxes are raised automatically when the GUI is generated giving a structure that code is written in to perform the function of the controls on the users request and to display texts and graphing of vibration simulations. Titles have been modified where it helps explain the function of GUI operations. Some brief GUIDE auto generated code remains unchanged and is required by the program to generate the GUI when it is called and where computer background colors are checked and some default colors set to generate the GUI screen.

The display of images, text, values, plotting and operation of all GUI pushbuttons, radiobuttons, sliders etc is operated by code written to present the training/awareness program. This code also modifies some color output for animations, the test of Level_1_10 and in some plotting functions.

The function file for the single degree of freedom system at the end of this appendix is called by the GUI code throughout the program for iteratively solving the response of the system to the input forcing functions. The two degree of freedom function file is called by Level_2_4 to solve displacement and velocity response for mass 1 and mass 2 due to the forcing function acting on mass 1, and only reslts for mass 1 are used.

In addition to the code in this appendix the GUIDE generated .fig figure files for each level are required. Images of GUI's are shown in Chapter 4.

The program can be started with the command run'Open' in the Matlab command window.

C.2 Open

```
function varargout = Open(varargin)
% OPEN M-file for Open.fig

% Last Modified by GUIDE v2.5 12-Sep-2015 14:25:46

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Open_OpeningFcn, ...
                   'gui_OutputFcn',  @Open_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before Open is made visible.
function Open_OpeningFcn(hObject, eventdata, handles, varargin)

% Choose default command line output for Open
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% --- Outputs from this function are returned to the command line.
function varargout = Open_OutputFcn(hObject, eventdata, handles)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)

run('Level_1_1')
close('Open')

% --- Executes during object creation, after setting all properties.
function pushbutton1_CreateFcn(hObject, eventdata, handles)

% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)

run('Level_2_1')
close('Open')

%End of OPEN.m
```

C.3 Level_1_1

```
    function varargout = Level_1_1(varargin)
% LEVEL_1_1 M-file for Level_1_1.fig
%
% Last Modified by GUIDE v2.5 11-Sep-2015 17:19:23

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Level_1_1_OpeningFcn, ...
                   'gui_OutputFcn',  @Level_1_1_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before Level_1_1 is made visible.
function Level_1_1_OpeningFcn(hObject, eventdata, handles, varargin)

% Choose default command line output for Level_1_1
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

%Displays text explanation in GUI edit1
fid=fopen('Level_1_1.txt');
C = textscan(fid,'%s','Delimiter','\n');
C=C{1};
fclose(fid);
```

```
set(handles.edit1,'String',C);

%Calculates and plots a sine wave
t=0:0.01:10;
w=get(handles.slider2,'Value');
A=get(handles.slider1,'Value');
sine=A*sin(w*t);
plot(handles.axes1,t,sine)
set(handles.axes1,'xlim', [0 11])
set(handles.axes1,'ylim',[-1.2 1.2])
axes(handles.axes1);
xlabel('Time (s)')
ylabel('Amplitude')

% --- Outputs from this function are returned to the command line.
function varargout = Level_1_1_OutputFcn(hObject, eventdata, handles)

% Get default command line output from handles structure
varargout{1} = handles.output;

function edit1_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        ,get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Recalculates and plots wave when amplitude is adjusted
function slider1_Callback(hObject, eventdata, handles)

t=0:0.01:10;
w=get(handles.slider2,'Value');
A=get(handles.slider1,'Value');
sine=A*sin(w*t);
plot(handles.axes1,t,sine)
set(handles.axes1,'xlim', [0 11])
set(handles.axes1,'ylim',[-1.2 1.2])
axes(handles.axes1);
xlabel('Time (s)')
```

```
ylabel('Amplitude')

% --- Executes during object creation, after setting all properties.
function slider1_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor')...
        ,get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Recalculates and plots wave when frequency is adjusted
function slider2_Callback(hObject, eventdata, handles)

t=0:0.01:10;
w=get(handles.slider2,'Value');
A=get(handles.slider1,'Value');
sine=A*sin(w*t);
plot(handles.axes1,t,sine)
set(handles.axes1,'xlim', [0 11])
set(handles.axes1,'ylim',[-1.2 1.2])
axes(handles.axes1);
xlabel('Time (s)')
ylabel('Amplitude')

% --- Executes during object creation, after setting all properties.
function slider2_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor')...
        ,get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)

% --- Executes on button press in Previous.
function Previous_Callback(hObject, eventdata, handles)

run('Open')
close('Level_1_1')

% --- Executes on button press in Next.
function Next_Callback(hObject, eventdata, handles)
```

```
run('Level_1_2')
close('Level_1_1')

% --- Executes during object creation, after setting all properties.
function Amplitude_CreateFcn(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function axes2_CreateFcn(hObject, eventdata, handles)

'callback'; SHM

% --- Executes on button press in radiobutton1.
function radiobutton1_Callback(hObject, eventdata, handles)

% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)

% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)

%Runs basic animation of simple harmonic motion
set(handles.pushbutton6,'enable','off');
a=get(handles.pushbutton5,'position');
A1=get(handles.slider1,'Value');
w=get(handles.slider2,'Value');
tstep=0.1;
t=0:tstep:10;
pause on
for n=1:max(size(t))
    y1=a(2)+0.18*A1*sin(w*t(n));
    pause(tstep)
    set(handles.pushbutton5,'position',[a(1) y1 a(3) a(4)])

    if n>=max(size(t))
    set(handles.pushbutton5,'position',[a(1) 0.289 a(3) a(4)])
    set(handles.pushbutton6,'enable','on');
    end
end

%END Level_1_1
```

C.4 Level_1_2

```
function varargout = Level_1_2(varargin)
% LEVEL_1_2 M-file for Level_1_2.fig

% Last Modified by GUIDE v2.5 17-Aug-2015 07:14:54

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Level_1_2_OpeningFcn, ...
                   'gui_OutputFcn',  @Level_1_2_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Level_1_2 is made visible.
function Level_1_2_OpeningFcn(hObject, eventdata, handles, varargin)

% Choose default command line output for Level_1_2
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

%Displays text explanation for concept presented in GUI
fid=fopen('Level_1_2.txt');
C = textscan(fid,'%s','Delimiter','\n');
C=C{1};
```

```
fclose(fid);
set(handles.edit1,'String',C);
%Calculates and plots sine waves with phase shift
t=0:0.01:10;
f=get(handles.slider2,'Value');
w=2*pi*f;
A=get(handles.slider1,'Value');
p=get(handles.slider3,'Value');
a=get(handles.slider4,'Value');
phi=p*pi;
sine=A*sin(w*t);
sine2=a*sin(w*t+phi);
plot(handles.axes1,t,sine,t,sine2)
xlabel('Time (s)')
ylabel('Amplitude')
set(handles.axes1,'xlim', [0 11])
set(handles.axes1,'ylim',[-1.2 1.2])

% --- Outputs from this function are returned to the command line.
function varargout = Level_1_2_OutputFcn(hObject, eventdata, handles)

% Get default command line output from handles structure
varargout{1} = handles.output;

function edit1_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        ,get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Recalculates and plots when amplitude(left) is changed.
function slider1_Callback(hObject, eventdata, handles)

t=0:0.01:10;
f=get(handles.slider2,'Value');
w=2*pi*f;
A=get(handles.slider1,'Value');
p=get(handles.slider3,'Value');
```

```
a=get(handles.slider4,'Value');
phi=p*pi;
sine=A*sin(w*t);
sine2=a*sin(w*t+phi);
plot(handles.axes1,t,sine,t,sine2)
xlabel('Time (s)')
ylabel('Amplitude')
set(handles.axes1,'xlim', [0 11])
set(handles.axes1,'ylim',[-1.2 1.2])

% --- Executes during object creation, after setting all properties.
function slider1_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor')...
        ,get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Recalculates and plots when Frequency is changed.
function slider2_Callback(hObject, eventdata, handles)

t=0:0.01:10;
f=get(handles.slider2,'Value');
w=2*pi*f;
A=get(handles.slider1,'Value');
p=get(handles.slider3,'Value');
a=get(handles.slider4,'Value');
phi=p*pi;
sine=A*sin(w*t);
sine2=a*sin(w*t+phi);
plot(handles.axes1,t,sine,t,sine2)
xlabel('Time (s)')
ylabel('Amplitude')
set(handles.axes1,'xlim', [0 11])
set(handles.axes1,'ylim',[-1.2 1.2])

% --- Executes during object creation, after setting all properties.
function slider2_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor')...
        ,get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
```

```
% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)

% --- Executes on button press in Previous.
function Previous_Callback(hObject, eventdata, handles)

run('Level_1_1')
pause off
close('Level_1_2')

% --- Executes on button press in Next.
function Next_Callback(hObject, eventdata, handles)

run('Level_1_3')
pause off
close('Level_1_2')

% --- Executes during object creation, after setting all properties.
function Amplitude_CreateFcn(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function axes2_CreateFcn(hObject, eventdata, handles)

% Shows equation for sin wave and phase adjustment
'callback';
text(0.1,0.95,'Shifting the Phase','FontSize',10)
text('Interpreter','latex','String'...
    ,'$$y=A\cos(\omega t \pm \phi)$$','Position',[.3 .85],'FontSize',10)


% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)

%Runs simple harmonic motion animation of two boxes as per the graph
set(handles.pushbutton3,'enable','off');
a=get(handles.pushbutton4,'position');
b=get(handles.pushbutton5,'position');
A1=get(handles.slider1,'Value');
A2=get(handles.slider4,'Value');
f=get(handles.slider2,'Value');
w=2*pi*f;
p=get(handles.slider3,'Value');
```

```
phi=p*pi;
tstep=0.01;
t=0:tstep:10;
pause on
for n=1:max(size(t))
    y1=a(2)+0.18*A1*sin(w*t(n));
    y2=b(2)+0.18*A2*sin(w*t(n)+phi);
    pause(tstep)
    set(handles.pushbutton4,'position',[a(1) y1 a(3) a(4)])
    set(handles.pushbutton5,'position',[b(1) y2 b(3) b(4)])
    if n>=max(size(t))
    set(handles.pushbutton4,'position',[a(1) 0.2944 a(3) a(4)])
    set(handles.pushbutton5,'position',[b(1) 0.2944 b(3) b(4)])
    set(handles.pushbutton3,'enable','on');
    end
end

% ---  pushbutton4 used in animation.
function pushbutton4_Callback(hObject, eventdata, handles)

% --- pushbutton5 used in animation.
function pushbutton5_Callback(hObject, eventdata, handles)

% --- Executes on button press in togglebutton1.
function togglebutton1_Callback(hObject, eventdata, handles)

a=get(handles.pushbutton4,'position');
disp(a)
tstep=0.1;
t=1:tstep:10;
pause on
for n=1:max(size(t))
    y=0.34+0.2*sin(3*t(n));
    pause(tstep)
set(handles.pushbutton4,'position',[a(1) y a(3) a(4)])
end

% --- Recalculates and plots when phase is changed.
function slider3_Callback(hObject, eventdata, handles)

t=0:0.01:10;
f=get(handles.slider2,'Value');
w=2*pi*f;
```

```
A=get(handles.slider1,'Value');
p=get(handles.slider3,'Value');
a=get(handles.slider4,'Value');
phi=p*pi;
sine=A*sin(w*t);
sine2=a*sin(w*t+phi);
plot(handles.axes1,t,sine,t,sine2)
xlabel('Time (s)')
ylabel('Amplitude')
set(handles.axes1,'xlim', [0 11])
set(handles.axes1,'ylim',[-1.2 1.2])

% --- Executes during object creation, after setting all properties.
function slider3_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor')...
        ,get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Recalculates and plots when amplitude (right) is changed.
function slider4_Callback(hObject, eventdata, handles)

t=0:0.01:10;
f=get(handles.slider2,'Value');
w=2*pi*f;
A=get(handles.slider1,'Value');
p=get(handles.slider3,'Value');
a=get(handles.slider4,'Value');
phi=p*pi;
sine=A*sin(w*t);
sine2=a*sin(w*t+phi);
plot(handles.axes1,t,sine,t,sine2)
xlabel('Time (s)')
ylabel('Amplitude')
set(handles.axes1,'xlim', [0 11])
set(handles.axes1,'ylim',[-1.2 1.2])

% --- Executes during object creation, after setting all properties.
function slider4_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

%END Level_1_2
```

C.5 Level_1_3

```
function varargout = Level_1_3(varargin)
% LEVEL_1_3 M-file for Level_1_3.fig

% Last Modified by GUIDE v2.5 17-Aug-2015 07:14:21

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Level_1_3_OpeningFcn, ...
                   'gui_OutputFcn',  @Level_1_3_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Level_1_3 is made visible.
function Level_1_3_OpeningFcn(hObject, eventdata, handles, varargin)

% Choose default command line output for Level_1_3
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
%Displays text for concepts covered in this GUI
fid=fopen('Level_1_3.txt');
C = textscan(fid,'%s','Delimiter','\n');
C=C{1};
fclose(fid);
```

```
set(handles.edit1,'String',C);

%Calculates and plots sinewave
TimeStep=0.001;
Time=2;
t=0:TimeStep:Time;
Frequency=get(handles.slider2,'Value');
w=2*pi*Frequency;
Amplitude=get(handles.slider1,'Value');
sine=Amplitude*sin(w*t);
plot(handles.axes1,t,sine)
set(handles.axes1,'xlim', [0 2.2])
set(handles.axes1,'ylim',[-1.2 1.2])

%Setting parameters for input into fft
SampleFrequency=1/TimeStep;
Length=Time/TimeStep;
%Hanning window
HanningLength=length(sine);
Transposesine=sine';
%This an amplitude correction factor of 2 for Hanning window
cf=2;
Sine=cf*Transposesine.*hanning(HanningLength);
% Turn wave into a binary length file for fft
NFFT = 2^(nextpow2(Length)+2);
Y = fft(Sine,NFFT)/Length;
f = SampleFrequency/2*linspace(0,1,NFFT/2+1);
plot(handles.axes2,f,2*abs(Y(1:NFFT/2+1)))
set(handles.axes2,'xlim', [0 55])
set(handles.axes2,'ylim',[0 1.2])
axes(handles.axes1);
xlabel('Time (s)')
ylabel('Amplitude')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Amplitude')

% --- Outputs from this function are returned to the command line.
function varargout = Level_1_3_OutputFcn(hObject, eventdata, handles)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

```
function edit1_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Recalculates and plots sinewave when amplitude is adjusted
function slider1_Callback(hObject, eventdata, handles)
%Calculates and plots sinewave
TimeStep=0.001;
Time=2;
t=0:TimeStep:Time;
Frequency=get(handles.slider2,'Value');
w=2*pi*Frequency;
Amplitude=get(handles.slider1,'Value');
sine=Amplitude*sin(w*t);
plot(handles.axes1,t,sine)
set(handles.axes1,'xlim', [0 2.2])
set(handles.axes1,'ylim',[-1.2 1.2])

%Setting parameters for input into fft
SampleFrequency=1/TimeStep;
Length=Time/TimeStep;
%Hanning window
HanningLength=length(sine);
Transposesine=sine';
%This an amplitude correction factor of 2 for Hanning window
cf=2;
Sine=cf*Transposesine.*hanning(HanningLength);
% Turn wave into a binary length file for fft
NFFT = 2^(nextpow2(Length)+2);
Y = fft(Sine,NFFT)/Length;
f = SampleFrequency/2*linspace(0,1,NFFT/2+1);
plot(handles.axes2,f,2*abs(Y(1:NFFT/2+1)))
set(handles.axes2,'xlim', [0 55])
set(handles.axes2,'ylim',[0 1.2])
axes(handles.axes1);
xlabel('Time (s)')
```

```
ylabel('Amplitude')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Amplitude')

% --- Executes during object creation, after setting all properties.
function slider1_CreateFcn(hObject, eventdata, handles)

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Recalculates and plots sinewave when frequency is adjusted
function slider2_Callback(hObject, eventdata, handles)

TimeStep=0.001;
Time=2;
t=0:TimeStep:Time;
Frequency=get(handles.slider2,'Value');
w=2*pi*Frequency;
Amplitude=get(handles.slider1,'Value');
sine=Amplitude*sin(w*t);
plot(handles.axes1,t,sine)
set(handles.axes1,'xlim', [0 2.2])
set(handles.axes1,'ylim',[-1.2 1.2])

%Setting parameters for input into fft
SampleFrequency=1/TimeStep;
Length=Time/TimeStep;
%Hanning window
HanningLength=length(sine);
Transposesine=sine';
%This an amplitude correction factor of 2 for Hanning window
cf=2;
Sine=cf*Transposesine.*hanning(HanningLength);
% Turn wave into a binary length file for fft
NFFT = 2^(nextpow2(Length)+2);
Y = fft(Sine,NFFT)/Length;
f = SampleFrequency/2*linspace(0,1,NFFT/2+1);
plot(handles.axes2,f,2*abs(Y(1:NFFT/2+1)))
set(handles.axes2,'xlim', [0 55])
```

```
set(handles.axes2,'ylim',[0 1.2])
axes(handles.axes1);
xlabel('Time (s)')
ylabel('Amplitude')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Amplitude')

% --- Executes during object creation, after setting all properties.
function slider2_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)

% --- Executes on button press in Previous.
function Previous_Callback(hObject, eventdata, handles)

run('Level_1_2')
close('Level_1_3')

% --- Executes on button press in Next.
function Next_Callback(hObject, eventdata, handles)

run('Level_1_4')
close('Level_1_3')

% --- Executes during object creation, after setting all properties.
function Amplitude_CreateFcn(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function axes2_CreateFcn(hObject, eventdata, handles)

%END Level_1_3
```

C.6 Level_1_4

```
function varargout = Level_1_4(varargin)
% LEVEL_1_4 M-file for Level_1_4.fig
%
% Last Modified by GUIDE v2.5 17-Aug-2015 07:13:34

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Level_1_4_OpeningFcn, ...
                   'gui_OutputFcn',  @Level_1_4_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before Level_1_4 is made visible.
function Level_1_4_OpeningFcn(hObject, eventdata, handles, varargin)

% Choose default command line output for Level_1_4
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

%Open a text document for insertion into GUI at edit1
fid=fopen('Level_1_4.txt');
C = textscan(fid,'%s','Delimiter','\n');
C=C{1};
fclose(fid);
```

```
set(handles.edit1,'String',C);

%Set initial conditions amplitude and frequency of displayed waveform
if (get(handles.radiobutton11,'Value') == get(handles.radiobutton11,'Max'))
    % Radio button is selected-take approriate action
    u=25.4;
else
    % Radio button is not selected-take approriate action
    u=1;
end
%
TimeStep=0.001;
Time=2;
t=0:TimeStep:Time;
Frequency=get(handles.slider2,'Value');
w=2*pi*Frequency;
Amplitude=u*get(handles.slider1,'Value');

%The following are needed to plot -peak, zero, peak and RMS lines
one=ones(size(t));
zero=zeros(size(t));
p=Amplitude*one;

%Makes the sine wave plotted in axes1 and for the fft process
%Also plots -peak, zero, peak and RMS lines
sine=Amplitude*sin(w*t);
PP=1500;
RM=1600;
plot(handles.axes1,t(1:PP),sine(1:PP),t,p,'g',t,-p,'g',t,zero,'g'...
    ,t(1:RM),0.707*p(1:RM),'r')
text(0.7,Amplitude+0.05,' This is the Positive Peak','Parent'...
    ,handles.axes1);
text(0.7,-(Amplitude+0.05),' This is the Negative Peak','Parent'...
    ,handles.axes1);
text(1.6,Amplitude/sqrt(2),' This is the RMS Level','Parent'...
    ,handles.axes1);
set(handles.axes1,'xlim', [0 2.2],'ylim',[-u*1.2 u*1.2])

%Setting parameters for input into fft
SampleFrequency=1/TimeStep;
Length=Time/TimeStep;
%Hanning window to reduce leakage in fft
HanningLength=length(sine);
```

```
Transposesine=sine';
%This an amplitude correction factor of 2. Energy calculation use cf 1.633
cf=2;
Sine=cf*Transposesine.*hanning(HanningLength);
% Next power of 2 for length
NFFT = 2^(nextpow2(Length)+2);
%Conduct fft and plot in axes2 as spectrum
Y = fft(Sine,NFFT)/Length;
f = SampleFrequency/2*linspace(0,1,NFFT/2+1);
plot(handles.axes2,f,2*abs(Y(1:NFFT/2+1)),f,sqrt(2)*abs(Y(1:NFFT/2+1)),'r')
set(handles.axes2,'xlim', [0 55],'ylim',[0 u*1.2])
%AXES LABELS
axes(handles.axes1);
xlabel('Time (s)')
ylabel('Amplitude')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Amplitude')

text(Frequency+1,Amplitude,'Peak','Parent',handles.axes2);
text(Frequency+1,Amplitude/sqrt(2),'RMS','Parent',handles.axes2);

% --- Outputs from this function are returned to the command line.
function varargout = Level_1_4_OutputFcn(hObject, eventdata, handles)

% Get default command line output from handles structure
varargout{1} = handles.output;

function edit1_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Recalculates and plots when amplitude is adjusted
function slider1_Callback(hObject, eventdata, handles)

if (get(handles.radiobutton11,'Value') == get(handles.radiobutton11,'Max'))
```

```
    % Radio button is selected-take approriate action
    u=25.4;
else
    % Radio button is not selected-take approriate action
    u=1;
end
%
TimeStep=0.001;
Time=2;
t=0:TimeStep:Time;
Frequency=get(handles.slider2,'Value');
w=2*pi*Frequency;
Amplitude=u*get(handles.slider1,'Value');

%The following are needed to plot -peak, zero, peak and RMS lines
one=ones(size(t));
zero=zeros(size(t));
p=Amplitude*one;

%Makes the sine wave plotted in axes1 and for the fft process
%Also plots -peak, zero, peak and RMS lines
sine=Amplitude*sin(w*t);
PP=1500;
RM=1600;
plot(handles.axes1,t(1:PP),sine(1:PP),t,p,'g',t,-p,'g',t,zero,'g'...
    ,t(1:RM),0.707*p(1:RM),'r')
text(0.7,Amplitude+0.05,' This is the Positive Peak','Parent'...
    ,handles.axes1);
text(0.7,-(Amplitude+0.05),' This is the Negative Peak','Parent'...
    ,handles.axes1);
text(1.6,Amplitude/sqrt(2),' This is the RMS Level','Parent'...
    ,handles.axes1);
set(handles.axes1,'xlim', [0 2.2],'ylim',[-u*1.2 u*1.2])

%Setting parameters for input into fft
SampleFrequency=1/TimeStep;
Length=Time/TimeStep;
%Hanning window to reduce leakage in fft
HanningLength=length(sine);
Transposesine=sine';
%This an amplitude correction factor of 2. Energy calculation use cf 1.633
cf=2;
Sine=cf*Transposesine.*hanning(HanningLength);
```

```
% Next power of 2 for length
NFFT = 2^(nextpow2(Length)+2);
%Conduct fft and plot in axes2 as spectrum
Y = fft(Sine,NFFT)/Length;
f = SampleFrequency/2*linspace(0,1,NFFT/2+1);
plot(handles.axes2,f,2*abs(Y(1:NFFT/2+1)),f,sqrt(2)*abs(Y(1:NFFT/2+1)),'r')
set(handles.axes2,'xlim', [0 55],'ylim',[0 u*1.2])

axes(handles.axes1);
xlabel('Time (s)')
ylabel('Amplitude')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Amplitude')

text(Frequency+1,Amplitude,'Peak','Parent',handles.axes2);
text(Frequency+1,Amplitude/sqrt(2),'RMS','Parent',handles.axes2);

% --- Executes during object creation, after setting all properties.
function slider1_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Recalculates and plots when frequency is adjusted
function slider2_Callback(hObject, eventdata, handles)
%
if (get(handles.radiobutton11,'Value') == get(handles.radiobutton11,'Max'))
    % Radio button is selected-take approriate action
    u=25.4;
else
    % Radio button is not selected-take approriate action
    u=1;
end
%
TimeStep=0.001;
Time=2;
t=0:TimeStep:Time;
Frequency=get(handles.slider2,'Value');
w=2*pi*Frequency;
Amplitude=u*get(handles.slider1,'Value');
```

```
%The following are needed to plot -peak, zero, peak and RMS lines
one=ones(size(t));
zero=zeros(size(t));
p=Amplitude*one;

%Makes the sine wave plotted in axes1 and for the fft process
%Also plots -peak, zero, peak and RMS lines
sine=Amplitude*sin(w*t);
PP=1500;
RM=1600;
plot(handles.axes1,t(1:PP),sine(1:PP),t,p,'g',t,-p,'g',t,zero,'g'...
    ,t(1:RM),0.707*p(1:RM),'r')
text(0.7,Amplitude+0.05,' This is the Positive Peak','Parent'...
    ,handles.axes1);
text(0.7,-(Amplitude+0.05),' This is the Negative Peak','Parent'...
    ,handles.axes1);
text(1.6,Amplitude/sqrt(2),' This is the RMS Level','Parent'...
    ,handles.axes1);
set(handles.axes1,'xlim', [0 2.2],'ylim',[-u*1.2 u*1.2])
%Setting parameters for input into fft
SampleFrequency=1/TimeStep;
Length=Time/TimeStep;
%Hanning window to reduce leakage in fft
HanningLength=length(sine);
Transposesine=sine';
%This an amplitude correction factor of 2.
cf=2;
Sine=cf*Transposesine.*hanning(HanningLength);
% Next power of 2 for length
NFFT = 2^(nextpow2(Length)+2);
%Conduct fft and plot in axes2 as spectrum
Y = fft(Sine,NFFT)/Length;
f = SampleFrequency/2*linspace(0,1,NFFT/2+1);
plot(handles.axes2,f,2*abs(Y(1:NFFT/2+1)),f,sqrt(2)*abs(Y(1:NFFT/2+1)),'r')
set(handles.axes2,'xlim', [0 55],'ylim',[0 u*1.2])

axes(handles.axes1);
xlabel('Time (s)')
ylabel('Amplitude')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Amplitude')
```

```
text(Frequency+1,Amplitude,'Peak','Parent',handles.axes2);
text(Frequency+1,Amplitude/sqrt(2),'RMS','Parent',handles.axes2);

% --- Executes during object creation, after setting all properties.
function slider2_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)

% --- Executes on button press in Previous.
function Previous_Callback(hObject, eventdata, handles)

run('Level_1_3')
close('Level_1_4')

% --- Executes on button press in Next.
function Next_Callback(hObject, eventdata, handles)

run('Level_1_5')
close('Level_1_4')

% --- Executes during object creation, after setting all properties.
function Amplitude_CreateFcn(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function axes2_CreateFcn(hObject, eventdata, handles)

% --- Recalculates and plots when mm are selected.
function radiobutton11_Callback(hObject, eventdata, handles)

if (get(handles.radiobutton11,'Value') == get(handles.radiobutton11,'Max'))
    % Radio button is selected-take approriate action
    u=25.4;
else
    % Radio button is not selected-take approriate action
    u=1;
end
```

```
%
TimeStep=0.001;
Time=2;
t=0:TimeStep:Time;
Frequency=get(handles.slider2,'Value');
w=2*pi*Frequency;
Amplitude=u*get(handles.slider1,'Value');

%The following are needed to plot -peak, zero, peak and RMS lines
one=ones(size(t));
zero=zeros(size(t));
p=Amplitude*one;

%Makes the sine wave plotted in axes1 and for the fft process
%Also plots -peak, zero, peak and RMS lines
sine=Amplitude*sin(w*t);
PP=1500;
RM=1600;
plot(handles.axes1,t(1:PP),sine(1:PP),t,p,'g',t,-p,'g',t,zero,'g'...
    ,t(1:RM),0.707*p(1:RM),'r')
text(0.7,Amplitude+0.05,' This is the Positive Peak','Parent'...
    ,handles.axes1);
text(0.7,-(Amplitude+0.05),' This is the Negative Peak','Parent'...
    ,handles.axes1);
text(1.6,Amplitude/sqrt(2),' This is the RMS Level','Parent'...
    ,handles.axes1);
set(handles.axes1,'xlim', [0 2.2],'ylim',[-u*1.2 u*1.2])

%Setting parameters for input into fft
SampleFrequency=1/TimeStep;
Length=Time/TimeStep;
%Hanning window to reduce leakage in fft
HanningLength=length(sine);
Transposesine=sine';
%This an amplitude correction factor of 2
cf=2;
Sine=cf*Transposesine.*hanning(HanningLength);
% Next power of 2 for length
NFFT = 2^(nextpow2(Length)+2);
%Conduct fft and plot in axes2 as spectrum
Y = fft(Sine,NFFT)/Length;
f = SampleFrequency/2*linspace(0,1,NFFT/2+1);
plot(handles.axes2,f,2*abs(Y(1:NFFT/2+1)),f,sqrt(2)*abs(Y(1:NFFT/2+1)),'r')
```

```
set(handles.axes2,'xlim', [0 55],'ylim',[0 u*1.2])

axes(handles.axes1);
xlabel('Time (s)')
ylabel('Amplitude')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Amplitude')

text(Frequency+1,Amplitude,'Peak','Parent',handles.axes2);
text(Frequency+1,Amplitude/sqrt(2),'RMS','Parent',handles.axes2);

% --- Recalculates and plots when inches are selected.
function radiobutton12_Callback(hObject, eventdata, handles)

if (get(handles.radiobutton11,'Value') == get(handles.radiobutton11,'Max'))
    % Radio button is selected-take approriate action
    u=25.4;
else
    % Radio button is not selected-take approriate action
    u=1;
end
%
TimeStep=0.001;
Time=2;
t=0:TimeStep:Time;
Frequency=get(handles.slider2,'Value');
w=2*pi*Frequency;
Amplitude=u*get(handles.slider1,'Value');

%The following are needed to plot -peak, zero, peak and RMS lines
one=ones(size(t));
zero=zeros(size(t));
p=Amplitude*one;

%Makes the sine wave plotted in axes1 and for the fft process
%Also plots -peak, zero, peak and RMS lines
sine=Amplitude*sin(w*t);

PP=1500;
RM=1600;
plot(handles.axes1,t(1:PP),sine(1:PP),t,p,'g',t,-p,'g',t,zero,'g'...
    ,t(1:RM),0.707*p(1:RM),'r')
```

```
text(0.7,Amplitude+0.05,' This is the Positive Peak','Parent'...
    ,handles.axes1);
text(0.7,-(Amplitude+0.05),' This is the Negative Peak','Parent'...
    ,handles.axes1);
text(1.6,Amplitude/sqrt(2),' This is the RMS Level','Parent'...
    ,handles.axes1);
set(handles.axes1,'xlim', [0 2.2],'ylim',[-u*1.2 u*1.2])

%Setting parameters for input into fft
SampleFrequency=1/TimeStep;
Length=Time/TimeStep;
%Hanning window to reduce leakage in fft
HanningLength=length(sine);
Transposesine=sine';
%This an amplitude correction factor of 2
cf=2;
Sine=cf*Transposesine.*hanning(HanningLength);
% Next power of 2 for length
NFFT = 2^(nextpow2(Length)+2);
%Conduct fft and plot in axes2 as spectrum
Y = fft(Sine,NFFT)/Length;
f = SampleFrequency/2*linspace(0,1,NFFT/2+1);
plot(handles.axes2,f,2*abs(Y(1:NFFT/2+1)),f,sqrt(2)*abs(Y(1:NFFT/2+1)),'r')
set(handles.axes2,'xlim', [0 55],'ylim',[0 u*1.2])

axes(handles.axes1);
xlabel('Time (s)')
ylabel('Amplitude')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Amplitude')

text(Frequency+1,Amplitude,'Peak','Parent',handles.axes2);
text(Frequency+1,Amplitude/sqrt(2),'RMS','Parent',handles.axes2);

% --- Executes during object creation, after setting all properties.
function axes3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
text(0.1,0.9,'Conversion of RPM to Hz','FontSize',10)
text('Interpreter','latex','String','$$f(Hz)=RPM/60$$','Position'...
    ,[.2 .75],'FontSize',10)
```

```
text(0.1,0.6,'RMS level for a sinusoid','FontSize',10)
text('Interpreter','latex','String','$$x_{rms}=A_{max}/\sqrt{2}$$'...
    ,'Position',[.2 .45],'FontSize',10)

%END Level_1_4
```

C.7 Level_1_5

```
function varargout = Level_1_5(varargin)
% LEVEL_1_5 M-file for Level_1_5.fig
%
% Last Modified by GUIDE v2.5 17-Aug-2015 07:11:15

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Level_1_5_OpeningFcn, ...
                   'gui_OutputFcn',  @Level_1_5_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before Level_1_5 is made visible.
function Level_1_5_OpeningFcn(hObject, eventdata, handles, varargin)

% Choose default command line output for Level_1_5
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

fid=fopen('Level_1_5.txt');
C = textscan(fid,'%s','Delimiter','\n');
C=C{1};
fclose(fid);
set(handles.edit1,'String',C);
```

```
FMax=get(handles.popupmenu2,'Value');
switch FMax
    case 1
        FMax=20000;
    case 2
        FMax=10000;
    case 3
        FMax=5000;
    case 4
        FMax=2000;
    case 5
        FMax=1000;
    case 6
        FMax=500;
    case 7
        FMax=200;
    case 8
        FMax=100;
    case 9
        FMax=50;
    otherwise
end

LOR=get(handles.popupmenu1,'Value');
switch LOR
    case 1
        LOR=25600;
    case 2
        LOR=12800;
    case 3
        LOR=6400;
    case 4
        LOR=3200;
    case 5
        LOR=1600;
    case 6
        LOR=800;
    case 7
        LOR=400;
    otherwise
end
```

```
%fft accounts for the common use of the factor 2.56. Matlab will use 100
% of the fft outcome giving a FMax at half the sampling frequency where
% because of alaising that can occur with digitizing a signal because of
%leakage with filtering hardware it is common to use 100%*(2/2.56)=78.125%
%
WavePoints=2.56*LOR;
Sampfreq=2.56*FMax;
TimeStep=1/Sampfreq;
t=(0:WavePoints-1)*TimeStep;

set(handles.slider2,'Max',FMax);
set(handles.slider2,'Min',1);
set(handles.slider2,'Value',0.5*FMax);

f=get(handles.slider2,'Value');
Amplitude=get(handles.slider1,'Value');
w=2*pi*f;
sine=Amplitude*sin(w*t);
plot(handles.axes1,t,sine)
set(handles.axes1,'ylim',[-1.1 1.1]);

Y = fft(sine,2.56*LOR)/(2.56*LOR);
f = FMax*linspace(0,1,LOR);
plot(handles.axes2,f,2*abs(Y(1:LOR)))
set(handles.axes2,'ylim',[0 1.2])

axes(handles.axes1);
xlabel('Time (s)')
ylabel('Amplitude')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Amplitude')

% --- Outputs from this function are returned to the command line.
function varargout = Level_1_5_OutputFcn(hObject, eventdata, handles)

% Get default command line output from handles structure
varargout{1} = handles.output;

function edit1_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on slider movement.
function slider1_Callback(hObject, eventdata, handles)

FMax=get(handles.popupmenu2,'Value');
switch FMax
    case 1
        FMax=20000;
    case 2
        FMax=10000;
    case 3
        FMax=5000;
    case 4
        FMax=2000;
    case 5
        FMax=1000;
    case 6
        FMax=500;
    case 7
        FMax=200;
    case 8
        FMax=100;
    case 9
        FMax=50;
    otherwise
end

LOR=get(handles.popupmenu1,'Value');
switch LOR
    case 1
        LOR=25600;
    case 2
        LOR=12800;
```

```
    case 3
        LOR=6400;
    case 4
        LOR=3200;
    case 5
        LOR=1600;
    case 6
        LOR=800;
    case 7
        LOR=400;
    otherwise
end

WavePoints=2.56*LOR;
Sampfreq=2.56*FMax;
TimeStep=1/Sampfreq;
t=(0:WavePoints-1)*TimeStep;

f=get(handles.slider2,'Value');
Amplitude=get(handles.slider1,'Value');
w=2*pi*f;
sine=Amplitude*sin(w*t);
plot(handles.axes1,t,sine)
set(handles.axes1,'ylim',[-1.1 1.1]);

Y = fft(sine,2.56*LOR)/(2.56*LOR);
f = FMax*linspace(0,1,LOR);
plot(handles.axes2,f,2*abs(Y(1:LOR)))
set(handles.axes2,'ylim',[0 1.2])

axes(handles.axes1);
xlabel('Time (s)')
ylabel('Amplitude')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Amplitude')

% --- Executes during object creation, after setting all properties.
function slider1_CreateFcn(hObject, eventdata, handles)

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function slider2_Callback(hObject, eventdata, handles)

FMax=get(handles.popupmenu2,'Value');
switch FMax
    case 1
        FMax=20000;
    case 2
        FMax=10000;
    case 3
        FMax=5000;
    case 4
        FMax=2000;
    case 5
        FMax=1000;
    case 6
        FMax=500;
    case 7
        FMax=200;
    case 8
        FMax=100;
    case 9
        FMax=50;
    otherwise
end

LOR=get(handles.popupmenu1,'Value');
switch LOR
    case 1
        LOR=25600;
    case 2
        LOR=12800;
    case 3
        LOR=6400;
    case 4
        LOR=3200;
    case 5
        LOR=1600;
    case 6
        LOR=800;
```

```
    case 7
        LOR=400;
    otherwise
end

WavePoints=2.56*LOR;
Sampfreq=2.56*FMax;
TimeStep=1/Sampfreq;
t=(0:WavePoints-1)*TimeStep;

set(handles.slider2,'Max',FMax);
set(handles.slider2,'Min',1);

f=get(handles.slider2,'Value');
Amplitude=get(handles.slider1,'Value');
w=2*pi*f;
sine=Amplitude*sin(w*t);
plot(handles.axes1,t,sine)
set(handles.axes1,'ylim',[-1.1 1.1]);

Y = fft(sine,2.56*LOR)/(2.56*LOR);
f = FMax*linspace(0,1,LOR);
plot(handles.axes2,f,2*abs(Y(1:LOR)))
set(handles.axes2,'ylim',[0 1.2])

axes(handles.axes1);
xlabel('Time (s)')
ylabel('Amplitude')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Amplitude')

% --- Executes during object creation, after setting all properties.
function slider2_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
```

```
% --- Executes on button press in Previous.
function Previous_Callback(hObject, eventdata, handles)

run('Level_1_4')
close('Level_1_5')

% --- Executes on button press in Next.
function Next_Callback(hObject, eventdata, handles)

run('Level_1_6')
close('Level_1_5')

% --- Executes during object creation, after setting all properties.
function Amplitude_CreateFcn(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function axes2_CreateFcn(hObject, eventdata, handles)

% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)

FMax=get(handles.popupmenu2,'Value');
switch FMax
    case 1
        FMax=20000;
    case 2
        FMax=10000;
    case 3
        FMax=5000;
    case 4
        FMax=2000;
    case 5
        FMax=1000;
    case 6
        FMax=500;
    case 7
        FMax=200;
    case 8
        FMax=100;
    case 9
        FMax=50;
    otherwise
end
```

```
LOR=get(handles.popupmenu1,'Value');
switch LOR
    case 1
        LOR=25600;
    case 2
        LOR=12800;
    case 3
        LOR=6400;
    case 4
        LOR=3200;
    case 5
        LOR=1600;
    case 6
        LOR=800;
    case 7
        LOR=400;
    otherwise
end

WavePoints=2.56*LOR;
Sampfreq=2.56*FMax;
TimeStep=1/Sampfreq;
t=(0:WavePoints-1)*TimeStep;

set(handles.slider2,'Max',FMax);
set(handles.slider2,'Min',1);

f=get(handles.slider2,'Value');
Amplitude=get(handles.slider1,'Value');
w=2*pi*f;
sine=Amplitude*sin(w*t);
plot(handles.axes1,t,sine)
set(handles.axes1,'ylim',[-1.1 1.1]);

Y = fft(sine,2.56*LOR)/(2.56*LOR);
f = FMax*linspace(0,1,LOR);
plot(handles.axes2,f,2*abs(Y(1:LOR)))
set(handles.axes2,'ylim',[0 1.2])

axes(handles.axes1);
xlabel('Time (s)')
ylabel('Amplitude')
```

```
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Amplitude')

% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in popupmenu2.
function popupmenu2_Callback(hObject, eventdata, handles)

FMax=get(handles.popupmenu2,'Value');
switch FMax
    case 1
        FMax=20000;
    case 2
        FMax=10000;
    case 3
        FMax=5000;
    case 4
        FMax=2000;
    case 5
        FMax=1000;
    case 6
        FMax=500;
    case 7
        FMax=200;
    case 8
        FMax=100;
    case 9
        FMax=50;
    otherwise
end

LOR=get(handles.popupmenu1,'Value');
switch LOR
    case 1
        LOR=25600;
```

```
    case 2
        LOR=12800;
    case 3
        LOR=6400;
    case 4
        LOR=3200;
    case 5
        LOR=1600;
    case 6
        LOR=800;
    case 7
        LOR=400;
    otherwise
end

WavePoints=2.56*LOR;
Sampfreq=2.56*FMax;
TimeStep=1/Sampfreq;
t=(0:WavePoints-1)*TimeStep;

f=get(handles.slider2,'Value');
Amplitude=get(handles.slider1,'Value');
w=2*pi*f;
sine=Amplitude*sin(w*t);
plot(handles.axes1,t,sine)
set(handles.axes1,'ylim',[-1.1 1.1]);

Y = fft(sine,2.56*LOR)/(2.56*LOR);
f = FMax*linspace(0,1,LOR);
plot(handles.axes2,f,2*abs(Y(1:LOR)))
set(handles.axes2,'ylim',[0 1.2])

axes(handles.axes1);
xlabel('Time (s)')
ylabel('Amplitude')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Amplitude')

% --- Executes during object creation, after setting all properties.
function popupmenu2_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

%END Level_1_5
```

C.8 Level_1_6

```
function varargout = Level_1_6(varargin)
% LEVEL_1_6 M-file for Level_1_6.fig
%
% Last Modified by GUIDE v2.5 17-Aug-2015 11:59:13

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Level_1_6_OpeningFcn, ...
                   'gui_OutputFcn',  @Level_1_6_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before Level_1_6 is made visible.
function Level_1_6_OpeningFcn(hObject, eventdata, handles, varargin)

% Choose default command line output for Level_1_6
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
%Display text description of concepts presented in GUI
fid=fopen('Level_1_6.txt');
C = textscan(fid,'%s','Delimiter','\n');
C=C{1};
fclose(fid);
set(handles.edit1,'String',C);
```

```
FMax=200;   %Maximum frequency
LOR=6400;   %Lines of resolution
%Get parameter values from sliders for Amplitude, frequency, mass and
%stiffness
Amplitude=get(handles.slider1,'Value');
fHz=get(handles.slider2,'Value');
m=get(handles.slider3,'Value');
k=get(handles.slider4,'Value');

WavePoints=2.56*LOR;
Sampfreq=2.56*FMax;
TimeStep=1/Sampfreq;
t=(0:WavePoints-1)*TimeStep;
%Forcing frequency and natural frequency
wf=2*pi*fHz;
wn=sqrt(k/m);
%Frequency ratio
F_R=wf/wn;
d_r=0.1;    %Damping ratio
T_R=((F_R^2)/sqrt((1-F_R^2)^2+(2*d_r*F_R)^2));
f_r=(0.01:0.01:2*F_R);
t_r=((f_r.^2)./sqrt((1-f_r.^2).^2+(2*d_r*f_r).^2));
sine=T_R*Amplitude*sin(wf*t);

set(handles.edit3,'String',fHz);
set(handles.edit4,'String',round(Amplitude,2));
set(handles.edit5,'String',round(wn/(2*pi),2));
set(handles.edit6,'String',round(T_R,2));

p=plot(handles.axes1,f_r,t_r,F_R,T_R);
p(2).Marker='O';
legend(handles.axes1,'Transmissibility Curve','System Transmissibility')
set(handles.axes1,'xlim',[0 2*F_R],'ylim',[0 6]);
axes(handles.axes1);
xlabel('wf/wn')
ylabel('Transmissibility Tr')
plot(handles.axes2,t,sine)
set(handles.axes2,'xlim',[0 1],'ylim',[-6 6]);
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Forcing Amplitude x Tr')
```

```
% --- Outputs from this function are returned to the command line.
function varargout = Level_1_6_OutputFcn(hObject, eventdata, handles)

% Get default command line output from handles structure
varargout{1} = handles.output;

function edit1_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Recalculates and plots when amplitude is adjusted
function slider1_Callback(hObject, eventdata, handles)

FMax=200;
LOR=6400;

Amplitude=get(handles.slider1,'Value');
fHz=get(handles.slider2,'Value');
m=get(handles.slider3,'Value');
k=get(handles.slider4,'Value');

WavePoints=2.56*LOR;
Sampfreq=2.56*FMax;
TimeStep=1/Sampfreq;
t=(0:WavePoints-1)*TimeStep;

wf=2*pi*fHz;
wn=sqrt(k/m);

F_R=wf/wn;
d_r=0.1;
T_R=((F_R^2)/sqrt((1-F_R^2)^2+(2*d_r*F_R)^2));
f_r=(0.01:0.01:2*F_R);
t_r=((f_r.^2)./sqrt((1-f_r.^2).^2+(2*d_r*f_r).^2));
sine=T_R*Amplitude*sin(wf*t);
```

```
set(handles.edit3,'String',fHz);
set(handles.edit4,'String',round(Amplitude,2));
set(handles.edit5,'String',round(wn/(2*pi),2));
set(handles.edit6,'String',round(T_R,2));

p=plot(handles.axes1,f_r,t_r,F_R,T_R);
p(2).Marker='O';
legend(handles.axes1,'Transmissibility Curve','System Transmissibility')
set(handles.axes1,'xlim',[0 2*F_R],'ylim',[0 6]);
axes(handles.axes1);
xlabel('wf/wn')
ylabel('Transmissibility Tr')
plot(handles.axes2,t,sine)
set(handles.axes2,'xlim',[0 1],'ylim',[-6 6]);
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Forcing Amplitude x Tr')

% --- Executes during object creation, after setting all properties.
function slider1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Recalculates and plots when frequency is adjusted
function slider2_Callback(hObject, eventdata, handles)

FMax=200;
LOR=6400;

Amplitude=get(handles.slider1,'Value');
fHz=get(handles.slider2,'Value');
m=get(handles.slider3,'Value');
k=get(handles.slider4,'Value');

WavePoints=2.56*LOR;
Sampfreq=2.56*FMax;
```

```
TimeStep=1/Sampfreq;
t=(0:WavePoints-1)*TimeStep;

wf=2*pi*fHz;
wn=sqrt(k/m);

F_R=wf/wn;
d_r=0.1;
T_R=((F_R^2)/sqrt((1-F_R^2)^2+(2*d_r*F_R)^2));
f_r=(0.01:0.01:2*F_R);
t_r=((f_r.^2)./sqrt((1-f_r.^2).^2+(2*d_r*f_r).^2));
sine=T_R*Amplitude*sin(wf*t);

set(handles.edit3,'String',fHz);
set(handles.edit4,'String',round(Amplitude,2));
set(handles.edit5,'String',round(wn/(2*pi),2));
set(handles.edit6,'String',round(T_R,2));

p=plot(handles.axes1,f_r,t_r,F_R,T_R);
p(2).Marker='O';
legend(handles.axes1,'Transmissibility Curve','System Transmissibility')
set(handles.axes1,'xlim',[0 2*F_R],'ylim',[0 6]);
axes(handles.axes1);
xlabel('wf/wn')
ylabel('Transmissibility Tr')
plot(handles.axes2,t,sine)
set(handles.axes2,'xlim',[0 1],'ylim',[-6 6]);
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Forcing Amplitude x Tr')

% --- Executes during object creation, after setting all properties.
function slider2_CreateFcn(hObject, eventdata, handles)

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
```

```
% --- Executes on button press in Previous.
function Previous_Callback(hObject, eventdata, handles)

run('Level_1_5')
close('Level_1_6')

% --- Executes on button press in Next.
function Next_Callback(hObject, eventdata, handles)

run('Level_1_7')
close('Level_1_6')

% --- Executes during object creation, after setting all properties.
function Amplitude_CreateFcn(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function axes2_CreateFcn(hObject, eventdata, handles)

% --- Displays equations for concepts covered within GUI.
function axes4_CreateFcn(hObject, eventdata, handles)

text(0.1,0.9,'Natural frequency mass/spring system','FontSize',10)
text('Interpreter','latex','String','$$\omega_{n}=\sqrt\frac{k}{m}  $$'...
    ,'Position',[0.2 0.75],'FontSize',10)

text(0.1,0.6,'Transmissibility','FontSize',10)
text('Interpreter','latex','String'...
    ,'$$Tr=\frac{\Omega^2}{\sqrt((1-\Omega)^2)^2+(2\zeta\Omega)^2)}$$'...
    ,'Position',[.2 0.45],'FontSize',10)

text(0.1,0.3,'Frequency Ratio','FontSize',10)
text('Interpreter','latex','String'...
    ,'$$\Omega=\frac{\omega_{f}}{\omega_{n}}$$','Position'...
    ,[0.6 0.3],'FontSize',10)

text(0.1,0.15,'Forcing Frequency','FontSize',10)
text('Interpreter','latex','String','$$\omega_{f}$$','Position'...
    ,[0.6 0.15],'FontSize',10)

text(0.1,0.05,'Natural Frequency','FontSize',10)
text('Interpreter','latex','String','$$\omega_{n}$$','Position'...
    ,[0.6 0.05],'FontSize',10)
```

```
% --- Recalculates and plots when system mass slider is adjusted
function slider3_Callback(hObject, eventdata, handles)

FMax=200;
LOR=6400;

Amplitude=get(handles.slider1,'Value');
fHz=get(handles.slider2,'Value');
m=get(handles.slider3,'Value');
k=get(handles.slider4,'Value');

WavePoints=2.56*LOR;
Sampfreq=2.56*FMax;
TimeStep=1/Sampfreq;
t=(0:WavePoints-1)*TimeStep;

wf=2*pi*fHz;
wn=sqrt(k/m);

F_R=wf/wn;
d_r=0.1;
T_R=((F_R^2)/sqrt((1-F_R^2)^2+(2*d_r*F_R)^2));
f_r=(0.01:0.01:2*F_R);
t_r=((f_r.^2)./sqrt((1-f_r.^2).^2+(2*d_r*f_r).^2));
sine=T_R*Amplitude*sin(wf*t);

set(handles.edit3,'String',fHz);
set(handles.edit4,'String',round(Amplitude,2));
set(handles.edit5,'String',round(wn/(2*pi),2));
set(handles.edit6,'String',round(T_R,2));

p=plot(handles.axes1,f_r,t_r,F_R,T_R);
p(2).Marker='O';
legend(handles.axes1,'Transmissibility Curve','System Transmissibility')
set(handles.axes1,'xlim',[0 2*F_R],'ylim',[0 6]);
axes(handles.axes1);
xlabel('wf/wn')
ylabel('Transmissibility Tr')
plot(handles.axes2,t,sine)
set(handles.axes2,'xlim',[0 1],'ylim',[-6 6]);
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Forcing Amplitude x Tr')
```

```
% --- Executes during object creation, after setting all properties.
function slider3_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Recalculates and plots when system stiffness slider is adjusted
function slider4_Callback(hObject, eventdata, handles)

FMax=200;
LOR=6400;

Amplitude=get(handles.slider1,'Value');
fHz=get(handles.slider2,'Value');
m=get(handles.slider3,'Value');
k=get(handles.slider4,'Value');

WavePoints=2.56*LOR;
Sampfreq=2.56*FMax;
TimeStep=1/Sampfreq;
t=(0:WavePoints-1)*TimeStep;

wf=2*pi*fHz;
wn=sqrt(k/m);

F_R=wf/wn;
d_r=0.1;
T_R=((F_R^2)/sqrt((1-F_R^2)^2+(2*d_r*F_R)^2));
f_r=(0.01:0.01:2*F_R);
t_r=((f_r.^2)./sqrt((1-f_r.^2).^2+(2*d_r*f_r).^2));
sine=T_R*Amplitude*sin(wf*t);

set(handles.edit3,'String',fHz);
set(handles.edit4,'String',round(Amplitude,2));
set(handles.edit5,'String',round(wn/(2*pi),2));
set(handles.edit6,'String',round(T_R,2));

p=plot(handles.axes1,f_r,t_r,F_R,T_R);
p(2).Marker='O';
legend(handles.axes1,'Transmissibility Curve','System Transmissibility')
```

```
set(handles.axes1,'xlim',[0 2*F_R],'ylim',[0 6]);
axes(handles.axes1);
xlabel('wf/wn')
ylabel('Transmissibility Tr')
plot(handles.axes2,t,sine)
set(handles.axes2,'xlim',[0 1],'ylim',[-6 6]);
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Forcing Amplitude x Tr')

% --- Executes during object creation, after setting all properties.
function slider4_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

function edit3_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit4_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit5_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
```

```
function edit5_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit6_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

%END_Level_1_6
```

C.9 Level_1_7

```
function varargout = Level_1_7(varargin)
% LEVEL_1_7 M-file for Level_1_7.fig

% Last Modified by GUIDE v2.5 23-Aug-2015 22:12:58

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Level_1_7_OpeningFcn, ...
                   'gui_OutputFcn',  @Level_1_7_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before Level_1_7 is made visible.
function Level_1_7_OpeningFcn(hObject, eventdata, handles, varargin)

% Choose default command line output for Level_1_7
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

evalin('base','clear')
%Display text covering concepts presented in GUI
fid=fopen('Level_1_7.txt');
C = textscan(fid,'%s','Delimiter','\n');
C=C{1};
```

```
fclose(fid);
set(handles.edit1,'String',C);
%Image associated with concept based on defect selected in popupmenu1
axes(handles.axes4)
imshow('Unbal.jpg');

FMax=get(handles.popupmenu2,'Value');
%Maximum frequency and colection parameters based on shaft speed (fHz)
switch FMax
    case 1  %[FMax LOR fHz FTF BSF ORDF IRDF]
        FMax=10000;
        LOR=6400;
        fHz=50;
    case 2
        FMax=10000;
        LOR=6400;
        fHz=25;
    case 3
        FMax=5000;
        LOR=3200;
        fHz=16.6;
    case 4
        FMax=2000;
        LOR=3200;
        fHz=8.3;

    otherwise
end
%Menu to select fault condition for input into SDOF model
dratio=get(handles.popupmenu1,'Value');
switch dratio
    case 1  %Unbalance
        dratio=1;           %Defect as a ratio of shaft speed
        mod_switch=2;       %Modulation
        mod_ratio=0;        %Rate of modulation as a ratio of shaft speed
        mis_switch=0;       %Misalignment
        vane_switch=0;      %Pump vane pass
        cav_switch=0;       %For cavitation effect
    case 2  %Misalignment
        dratio=1;
        mod_switch=2;
        mod_ratio=0;
        mis_switch=1;
```

```
        vane_switch=0;
        cav_switch=0;
    case 3  %Looseness
        dratio=1;
        mod_switch=0;
        mod_ratio=0;
        mis_switch=0;
        vane_switch=0;
        cav_switch=0;
    case 4  %Gear damage
        dratio=17;
        mod_switch=1;
        mod_ratio=1;
        mis_switch=0;
        vane_switch=0;
        cav_switch=0;
    case 5  %Pump vane psee
        dratio=5;
        mod_switch=2;
        mod_ratio=0;
        mis_switch=0;
        vane_switch=1;
        cav_switch=0;
    case 6  %Cavitation
        dratio=1;
        mod_switch=2;
        mod_ratio=0;
        mis_switch=0;
        vane_switch=1;
        cav_switch=1;
    case 7  %Friction not yet working
        dratio=1;
        mod_switch=2;
        mod_ratio=0;
        mis_switch=0;
        vane_switch=0;
        cav_switch=0;

    otherwise
end

global Mt M mo c k T r wf fi tspan
     %Kust test for a bearing outer defect with 5.4 orders defect
```

```
%Also outer because force will be constant, ie in addition to the imbalance

M=150;                          %Machine mass
mo=500*(1/fHz^2);               %Unbalance mass
Mt=M+mo;                        %Syatem mass
k=460000000;                    %Stiffness N/m
wf=2*pi*fHz;                    %Shaft turning speed rad/s
%beta=0.05;                     %Empirical constant for structural damping
%c=2*k*beta/wf;                 %Structural Damping coefficient
c=100000;                       %Viscous damping coefficient
imp=10*fHz^2;                   %Peak pulse force N varies with speed
r=0.5;                          %radius unbalance mass rotates at
dHz=fHz*dratio;                 %Defect ration as a function of shaft speed
T=1/dHz;                        %Defect force impulse period
%wn=sqrt(k/M+mo);               %Single degree of freedom natural frequency
%c_c=2*(M+mo)*wn;               %Critical damping
%zeta=c/c_c;                    %Damping ration
Wave_L=2.56*LOR;                %Number of points in the timewaveform
Samp_freq=2.56*FMax;            %Frequency of sampling in waveform
Tstep=1/Samp_freq;              %The waveform sample period
dt=0.1;                         %Time shift of wave for reaching steady state
dt_n=round(dt/Tstep);
%Time span for fixing ODE slover iterations
tspan=(0:(Wave_L-1)+dt_n)*Tstep;
d=(0:T:max(tspan));                 % Array for defect repetition
%Single degree of freedom defect input forces
%This determines value of pulse force to go to solver
if mod_switch==0
    y1=imp*(pulstran(tspan,d,'rectpuls',0.05*T));
elseif mod_switch==1
    y1=imp*((0.5*(1+sin(mod_ratio*wf*tspan))).*...
        (pulstran(tspan,d,'rectpuls',0.05*T)));
elseif mod_switch==2
    y1=zeros(1,(max(size(tspan))));
end

%This will add a value y2 for misalignment 180 degree out of phase at 0.7
%the magnitude of unbalance force for convenience in displating the effect
if mis_switch==1
    y2=1.5*mo*r*wf^2*sin(2*wf*tspan);
else
    y2=zeros(1,(max(size(tspan))));
end
```

```
%This adds the vane pass frequency vibration if called and in this case is
%a simple model for a centrifugal pump with 5 vanes.
if vane_switch==1
    y3=0.1*mo*r*wf^2*sin(5*wf*tspan);
else
    y3=zeros(1,(max(size(tspan))));
end

%Input force for cavitation effect
if cav_switch==1

    y4 = (10*fHz^2)*randn(1,length(tspan));

else
    y4=zeros(1,(max(size(tspan))));
end
%Sum of single degree of freedom defect input forces
fi=y1+y2+y3+y4;
%Calls SDOF SOLVER
[t, x] = ode23('SDOF_SOLVER',tspan, [0 0]);
%Solutions to SDOF system
D=x(:,1);
D=D(dt_n+1:end);
V=x(:,2);
V=V(dt_n+1:end);
A=diff(V);
A(max(size(A))+1)=0;
t=t(1:end-dt_n);

plot(handles.axes1,t,V*1000);       %Velocity waveform mm/s

%fft to use 100%*(2/2.56)=78.125% of Nyquist rate
Yv = fft(V,2.56*LOR)/(2.56*LOR);
YV=2*abs(Yv(1:LOR));
fV = FMax*linspace(0,1,LOR);
plot(handles.axes2,fV,YV*1000)      %Velocity spectrum mm/s

%ACCELERATION SPECTRUM
Ya = fft(A,2.56*LOR)/(2.56*LOR);
YA=2*abs(Ya(1:LOR));
fA = FMax*linspace(0,1,LOR);
%plot(handles.axes2,fA,YA)
```

```
axes(handles.axes1);
xlabel('Time (s)')
ylabel('Velocity (mm/s) ')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Peak Velocity (mm/s)')

assignin ('base','D',D);
assignin ('base','V',V)
assignin ('base','A',A)
assignin ('base','t',t)
assignin ('base','fV',fV);
assignin ('base','YV',YV)
assignin ('base','fA',fA);
assignin ('base','YA',YA)

set(handles.edit4,'String',(fHz*dratio));
set(handles.edit5,'String',dratio);

% --- Outputs from this function are returned to the command line.
function varargout = Level_1_7_OutputFcn(hObject, eventdata, handles)

% Get default command line output from handles structure
varargout{1} = handles.output;

function edit1_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)

% --- Executes on button press in Previous.
function Previous_Callback(hObject, eventdata, handles)
```

```
run('Level_1_6')
close('Level_1_7')

% --- Executes on button press in Next.
function Next_Callback(hObject, eventdata, handles)

run('Level_1_8')
close('Level_1_7')

% --- Executes during object creation, after setting all properties.
function Amplitude_CreateFcn(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function axes2_CreateFcn(hObject, eventdata, handles)

% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
%See opening function for Level_1_7  for additional notes
FMax=get(handles.popupmenu2,'Value');

switch FMax
    case 1  %[FMax LOR fHz FTF BSF ORDF IRDF]
        FMax=10000;
        LOR=6400;
        fHz=50;
    case 2
        FMax=10000;
        LOR=6400;
        fHz=25;
    case 3
        FMax=5000;
        LOR=3200;
        fHz=16.6;
    case 4
        FMax=2000;
        LOR=3200;
        fHz=8.3;

    otherwise
end

dratio=get(handles.popupmenu1,'Value');
switch dratio
```

```
    case 1  %Unbalance
        dratio=1;
        mod_switch=2;
        mod_ratio=0;
        mis_switch=0;
        vane_switch=0;
        cav_switch=0;
        axes(handles.axes4)
        imshow('Unbal.jpg');        %Image associated with defect selected
    case 2  %Misalignment
        dratio=1;
        mod_switch=2;
        mod_ratio=0;
        mis_switch=1;
        vane_switch=0;
        cav_switch=0;
        axes(handles.axes4)
        imshow('MisAlign.jpg');     %Image associated with defect selected
    case 3  %Looseness
        dratio=1;
        mod_switch=0;
        mod_ratio=0;
        mis_switch=0;
        vane_switch=0;
        cav_switch=0;
    case 4  %Gear damage
        dratio=17;
        mod_switch=1;               %Gives modulating gear damage
        mod_ratio=1;
        mis_switch=0;
        vane_switch=0;
        cav_switch=0;
    case 5  %Pump vane pass
        dratio=5;
        mod_switch=2;
        mod_ratio=0;
        mis_switch=0;
        vane_switch=1;
        cav_switch=0;
    case 6 %Cavitation
        dratio=1;
        mod_switch=2;
        mod_ratio=0;
```

```
        mis_switch=0;
        vane_switch=1;
        cav_switch=1;
    case 7  %Friction
        dratio=1;
        mod_switch=2;
        mod_ratio=0;
        mis_switch=0;
        vane_switch=0;
        cav_switch=0;

    otherwise
end

global Mt M mo c k T r wf fi tspan

h=waitbar(0.5,'Calculating Output, Please wait...');

M=150;                      %Machine mass
mo=500*(1/fHz^2);           %Unbalance mass
Mt=M+mo;                    %System mass
k=460000000;                %Stiffness N/m
wf=2*pi*fHz;                %Shaft turning speed rad/s
c=100000;                   %Viscous damping coefficient
imp=10*fHz^2;               %Peak pulse force N varies with speed
r=0.5;                      %radius unbalance mass rotates at
dHz=fHz*dratio;
T=1/dHz;                    %Defect force impulse period
Wave_L=2.56*LOR;            %Number of points in the timewaveform
Samp_freq=2.56*FMax;        %Frequency of sampling in waveform
Tstep=1/Samp_freq;          %The waveform sample period
dt=0.1;                     %Time shift of wave for reaching steady state
dt_n=round(dt/Tstep);
%tspan created to control ODE solver iterations
tspan=(0:(Wave_L-1)+dt_n)*Tstep;
d=(0:T:max(tspan));                 % T repetition frequency

%This determines value of pulse force to go to solver
if mod_switch==0
    y1=imp*(pulstran(tspan,d,'rectpuls',0.05*T));
elseif mod_switch==1
    y1=imp*((0.5*(1+sin(mod_ratio*wf*tspan))).*...
        (pulstran(tspan,d,'rectpuls',0.05*T)));
```

```
elseif mod_switch==2
    y1=zeros(1,(max(size(tspan))));
end

%This will add a value y2 for misalignment 180 degree out of phase at 0.7
%the magnitude of unbalance force for convenience in displating the effect
if mis_switch==1
    y2=1.5*mo*r*wf^2*sin(2*wf*tspan);
else
    y2=zeros(1,(max(size(tspan))));
end

%This adds the vane pass frequency vibration if called and in this case is
%a simple model for a centrifugal pump with 5 vanes.
if vane_switch==1
    y3=0.1*mo*r*wf^2*sin(5*wf*tspan);
else
    y3=zeros(1,(max(size(tspan))));
end

if cav_switch==1

    y4 = (10*fHz^2)*randn(1,length(tspan));

else
    y4=zeros(1,(max(size(tspan))));
end
%Sum of input defect forces for ODE solver
fi=y1+y2+y3+y4;
%Calls solver
[t, x] = ode23('SDOF_SOLVER',tspan, [0 0]);

D=x(:,1);
D=D(dt_n+1:end);
V=x(:,2);
V=V(dt_n+1:end);
A=diff(V);
A(max(size(A))+1)=0;
t=t(1:end-dt_n);

plot(handles.axes1,t,V*1000);       %1000 converts m to mm

%fft to use 100%*(2/2.56)=78.125% of Nyquist rate
```

```
Yv = fft(V,2.56*LOR)/(2.56*LOR);
YV=2*abs(Yv(1:LOR));
fV = FMax*linspace(0,1,LOR);
plot(handles.axes2,fV,YV*1000)    %Velocity waveform mm/s

%ACCELERATION SPECTRUM
Ya = fft(A,2.56*LOR)/(2.56*LOR);
YA=2*abs(Ya(1:LOR));
fA = FMax*linspace(0,1,LOR);
%plot(handles.axes2,fA,YA)

axes(handles.axes1);
xlabel('Time (s)')
ylabel('Velocity (mm/s) ')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Peak Velocity (mm/s)')

assignin ('base','D',D);
assignin ('base','V',V)
assignin ('base','A',A)
assignin ('base','t',t)
assignin ('base','fV',fV);
assignin ('base','YV',YV)
assignin ('base','fA',fA);
assignin ('base','YA',YA)

set(handles.edit4,'String',(fHz*dratio));
set(handles.edit5,'String',dratio);

close(h)

% --- Executes during object creation, after setting all properties.
function popupmenu2_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
```

```
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in popupmenu2.
function popupmenu2_Callback(hObject, eventdata, handles)

FMax=get(handles.popupmenu2,'Value');

switch FMax
    case 1
        FMax=10000;
        LOR=6400;
        fHz=50;
    case 2
        FMax=10000;
        LOR=6400;
        fHz=25;
    case 3
        FMax=5000;
        LOR=3200;
        fHz=16.6;
    case 4
        FMax=2000;
        LOR=3200;
        fHz=8.3;

    otherwise
end

dratio=get(handles.popupmenu1,'Value');
switch dratio
    case 1
        dratio=1;
        mod_switch=2;
        mod_ratio=0;
        mis_switch=0;
        vane_switch=0;
        cav_switch=0;
    case 2
```

```
        dratio=1;
        mod_switch=2;
        mod_ratio=0;
        mis_switch=1;
        vane_switch=0;
        cav_switch=0;
    case 3
        dratio=1;
        mod_switch=0;
        mod_ratio=0;
        mis_switch=0;
        vane_switch=0;
        cav_switch=0;
    case 4
        dratio=17;
        mod_switch=1;
        mod_ratio=1;
        mis_switch=0;
        vane_switch=0;
        cav_switch=0;
    case 5
        dratio=5;
        mod_switch=2;
        mod_ratio=0;
        mis_switch=0;
        vane_switch=1;
        cav_switch=0;
    case 6
        dratio=1;
        mod_switch=2;
        mod_ratio=0;
        mis_switch=0;
        vane_switch=1;
        cav_switch=1;
    case 7
        dratio=1;
        mod_switch=2;
        mod_ratio=0;
        mis_switch=0;
        vane_switch=0;
        cav_switch=0;

    otherwise
```

```
end

global Mt M mo c k T r wf fi tspan

h=waitbar(0.5,'Calculating Output, Please wait...');

M=150;                          %Machine mass
mo=500*(1/fHz^2);               %Unbalance mass
Mt=M+mo;                        %System mass
k=460000000;                    %Stiffness N/m
wf=2*pi*fHz;                    %Shaft turning speed rad/s
c=100000;                       %Viscous damping coefficient
imp=10*fHz^2;                   %Peak pulse force N varies with speed
r=0.5;                          %Radius unbalance mass rotates at
dHz=fHz*dratio;
T=1/dHz;                        %Defect force impulse period
%wn=sqrt(k/M+mo)                %Single degree of freedom natural frequency
%c_c=2*(M+mo)*wn
%zeta=c/c_c
Wave_L=2.56*LOR;                %Number of points in the timewaveform
Samp_freq=2.56*FMax;            %Frequency of sampling in waveform
Tstep=1/Samp_freq;              %The waveform sample period
dt=0.1;                         %Time shift of wave for reaching steady state
dt_n=round(dt/Tstep);
%Tspan to control iteration time steps of ODE solver
tspan=(0:(Wave_L-1)+dt_n)*Tstep;
d=(0:T:max(tspan));             % T repetition frequency

%This determines value of pulse force to go to solver
if mod_switch==0
    y1=imp*(pulstran(tspan,d,'rectpuls',0.05*T));
elseif mod_switch==1
    y1=imp*((0.5*(1+sin(mod_ratio*wf*tspan))).*...
        (pulstran(tspan,d,'rectpuls',0.05*T)));
elseif mod_switch==2
    y1=zeros(1,(max(size(tspan))));
end

%This will add a value y2 for misalignment 180 degree out of phase at 0.7
%the magnitude of unbalance force for convenience in displating the effect
if mis_switch==1
    y2=1.5*mo*r*wf^2*sin(2*wf*tspan);
else
```

```
    y2=zeros(1,(max(size(tspan))));
end

%This adds the vane pass frequency vibration if called and in this case is
%a simple model for a centrifugal pump with 5 vanes.
if vane_switch==1
    y3=0.1*mo*r*wf^2*sin(5*wf*tspan);
else
    y3=zeros(1,(max(size(tspan))));
end

%Cavitation input force
if cav_switch==1

    y4 = (10*fHz^2)*randn(1,length(tspan));

else
    y4=zeros(1,(max(size(tspan))));
end
%Sum of defect input force for ODE solver
fi=y1+y2+y3+y4;
%Calls solver
[t, x] = ode23('SDOF_SOLVER',tspan, [0 0]);
%Solver results
D=x(:,1);
D=D(dt_n+1:end);
V=x(:,2);
V=V(dt_n+1:end);
A=diff(V);
A(max(size(A))+1)=0;
t=t(1:end-dt_n);

plot(handles.axes1,t,V*1000);       %Velocity waveform mm/s

%fft to use 100%*(2/2.56)=78.125% of Nyquist rate
Yv = fft(V,2.56*LOR)/(2.56*LOR);
YV=2*abs(Yv(1:LOR));
fV = FMax*linspace(0,1,LOR);
plot(handles.axes2,fV,YV*1000)      %Velocity spectrum mm/s

%ACCELERATION SPECTRUM
Ya = fft(A,2.56*LOR)/(2.56*LOR);
YA=2*abs(Ya(1:LOR));
```

```
fA = FMax*linspace(0,1,LOR);
%plot(handles.axes2,fA,YA)

axes(handles.axes1);
xlabel('Time (s)')
ylabel('Velocity (mm/s) ')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Peak Velocity (mm/s)')

assignin ('base','D',D);
assignin ('base','V',V)
assignin ('base','A',A)
assignin ('base','t',t)
assignin ('base','fV',fV);
assignin ('base','YV',YV)
assignin ('base','fA',fA);
assignin ('base','YA',YA)

set(handles.edit4,'String',(fHz*dratio));
set(handles.edit5,'String',dratio);

close(h)

% --- Executes during object creation, after setting all properties.
function axes4_CreateFcn(hObject, eventdata, handles)

function edit4_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit5_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
```

```
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
%Plots spectrum in figure separate to GUI
fV=evalin('base','fV');
YV=evalin('base','YV');
figure
plot(fV,YV*1000)
xlabel('Frequency (Hz)')
ylabel('Peak Velocity (mm/s) ')

% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
%Plots waveform in figure separate to GUI
V=evalin('base','V');
t=evalin('base','t');
figure
plot(t,V*1000)
xlabel('Time (s)')
ylabel('Velocity (mm/s) ')

%END Level_1_7
```

C.10 Level_1_8

```
function varargout = Level_1_8(varargin)
% LEVEL_1_8 M-file for Level_1_8.fig
%
% Last Modified by GUIDE v2.5 23-Aug-2015 15:03:26

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Level_1_8_OpeningFcn, ...
                   'gui_OutputFcn',  @Level_1_8_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before Level_1_8 is made visible.
function Level_1_8_OpeningFcn(hObject, eventdata, handles, varargin)

% Choose default command line output for Level_1_8
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

fid=fopen('Level_1_8.txt');
C = textscan(fid,'%s','Delimiter','\n');
C=C{1};
fclose(fid);
set(handles.edit1,'String',C);
```

```
FMax=get(handles.popupmenu2,'Value');
%FMax Maximum frequency specified for spectrum
%LOR Lines of resolution for spectrum
%fHz shaft speed
%Baring defects as a ratio of shaft speed
%FTF(Cage Defect), BSF(Ball Spin), ORDF(Outer Race), IRDF(Inner Race)
switch FMax
    case 1  %[FMax LOR fHz  FTF  BSF  ORDF IRDF]
        Info=[200 1600 1.67 0.45 4.74 11.2 13.8];       %Conveyor pulley
        axes(handles.axes4)
        imshow('Pulley.jpg');   %Image of machine associated with defect
    case 2
        Info=[1000 3200 7.05 0.441 4.074 9.711 12.289];   %Crusher shaft
        axes(handles.axes4)
        imshow('Crusher.jpg');  %Image of machine associated with defect
    case 3
        Info=[2000 6400 19.62 0.4 2.45 6.47 9.53];           %GBox shaft
        axes(handles.axes4)
        imshow('2ndShaft.jpg'); %Image of machine associated with defect
    case 4
        Info=[2000 6400 24.83 0.39 2.09 3.09 4.91];   %Motor NDE Bearing
        axes(handles.axes4)
        imshow('MtrNDE.jpg');   %Image of machine associated with defect
    case 5
        Info=[2000 6400 24.83 0.4 2.43 5.21 7.79];     %Motor DE Bearing
        axes(handles.axes4)
        imshow('MtrDE.jpg');    %Image of machine associated with defect
    otherwise
end
%Selects data collection properties and defect related properties
%based on the defect selected in the 'Non Synchronous Defects' popupmenu
LOR=get(handles.popupmenu1,'Value');
switch LOR
    case 1              %Bearing Cage Defect
        FMax=Info(1);       %Selects maximum frequency
        LOR=Info(2);        %Selects Lines of Resolution
        fHz=Info(3);        %Selects shaft speed
        dratio=Info(4);     %Selects defect ratio
        mod_switch=0;       %No modulation
        mod_ratio=0;        %No modulation
    case 2              %Bearing Ball Defect
        FMax=Info(1);       %Selects maximum frequency
```

```
        LOR=Info(2);        %Selects Lines of Resolution
        fHz=Info(3);        %Selects shaft speed
        %Multiply ball spin ratio by 2, 2 impacts inner and outer per rev
        dratio=2*Info(5);   %Selects defect ratio
        mod_switch=1;       %No modulation
        mod_ratio=Info(4);  %Modulation as a ratio of shaft speed
    case 3              %Outer Race Defect (Rotating Shaft)
        FMax=Info(1);       %Selects maximum frequency
        LOR=Info(2);        %Selects Lines of Resolution
        fHz=Info(3);        %Selects shaft speed
        dratio=Info(6);     %Selects defect ratio
        mod_switch=0;       %No modulation
        mod_ratio=0;        %No modulation
    case 4              %Inner Race Defect (Rotating Shaft)
        FMax=Info(1);       %Selects maximum frequency
        LOR=Info(2);        %Selects Lines of Resolution
        fHz=Info(3);        %Selects shaft speed
        dratio=Info(7);     %Selects defect ratio
        mod_switch=1;       %Applies modulation
        mod_ratio=1;        %Modulation as a ratio of shaft speed = 1

    otherwise
end

global Mt M mo c k T r wf fi tspan

M=150;                      %Machine mass
mo=50*(1/fHz^2);            %Unbalance mass
Mt=M+mo;                    %System mass
c=100000;                   %Damping coefficient
k=460000000;                %Stiffness N/m
imp=100*fHz;                %Peak pulse force N varies with speed
r=0.5;                      %radius unbalance mass rotates at
dHz=fHz*dratio;             %Defect ratio Hz
T=1/dHz;                    %Defect force pulse period
wf=2*pi*fHz;                %Shaft turning speed rad/s
%c_c=2*(M+mo)*wn            %Critical damping ratio
%zeta=c/c_c                 %Damping ration
%wn=sqrt(k/(M+mo));         %Single degree of freedom natural frequency
Wave_L=2.56*LOR;            %Number of points in the timewaveform
Samp_freq=2.56*FMax;        %Frequency of sampling in waveform
Tstep=1/Samp_freq;          %The waveform sample period
Time=LOR/FMax;              %Time/duration of waveform seconds
```

```
%Sets time steps to control ODE iterations
tspan=(0:Wave_L-1)*Tstep;
d=(0:T:Time);               % Array for defect repetition

%Modified sinusoid to simulate damaged bearing component entering and
%leaving the bearing load zone
y1=sin(mod_ratio*wf*tspan)-0.2165;
yscale=1/max(y1);
for n=1:length(tspan)
    if y1(n)>0
    y1(n)=y1(n)*yscale;
    else y1(n)=0;
    end
end
%Create pulse force for input into ODE solver
if mod_switch==0
    y=pulstran(tspan,d,'rectpuls',0.05*T);
else
    y=y1.*(pulstran(tspan,d,'rectpuls',0.05*T));
end
fi=imp*y;

%Call ODE Solver
[t, x] = ode23('SDOF_SOLVER',tspan, [0 0]);
%Solver results
D=x(:,1);
V=x(:,2);
A=diff(V);
A(max(size(A))+1)=0;

plot(handles.axes1,t,V*1000);       %1000 converts m to mm

%fft to use 100%*(2/2.56)=78.125% of Nyquist rat

Yv = fft(V,2.56*LOR)/(2.56*LOR);
YV=2*abs(Yv(1:LOR));
fV = FMax*linspace(0,1,LOR);
plot(handles.axes2,fV,YV*1000)    %Velocity spectrum mm/s

%ACCELERATION SPECTRUM
%Ya = fft(A,2.56*LOR)/(2.56*LOR);
%YA=2*abs(Ya(1:LOR));
%fA = FMax*linspace(0,1,LOR);
```

```
%plot(handles.axes2,fA,YA)

axes(handles.axes1);
xlabel('Time (s)')
ylabel('Velocity (mm/s) ')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Peak Velocity (mm/s)')

assignin ('base','D',D);
assignin ('base','V',V)
assignin ('base','A',A)
assignin ('base','tspan',tspan)
assignin ('base','fV',fV);
assignin ('base','YV',YV)

set(handles.edit3,'String',fHz);
set(handles.edit4,'String',round((fHz*dratio),2));
set(handles.edit5,'String',dratio);

% --- Outputs from this function are returned to the command line.
function varargout = Level_1_8_OutputFcn(hObject, eventdata, handles)

% Get default command line output from handles structure
varargout{1} = handles.output;

function edit1_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)

% --- Executes on button press in Previous.
function Previous_Callback(hObject, eventdata, handles)
```

```
run('Level_1_7')
close('Level_1_8')

% --- Executes on button press in Next.
function Next_Callback(hObject, eventdata, handles)

run('Level_1_9')
close('Level_1_8')

% --- Executes during object creation, after setting all properties.
function Amplitude_CreateFcn(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function axes2_CreateFcn(hObject, eventdata, handles)

% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)

FMax=get(handles.popupmenu2,'Value');
%FMax Maximum frequency specified for spectrum
%LOR Lines of resolution for spectrum
switch FMax
    case 1  %[FMax LOR fHz FTF BSF ORDF IRDF]
        Info=[200 1600 1.67 0.45 4.74 11.2 13.8];       %Conveyor Pulley
        axes(handles.axes4)
        imshow('Pulley.jpg');
    case 2
        Info=[1000 3200 7.05 0.441 4.074 9.711 12.289]; %Crusher Shaft
        axes(handles.axes4)
        imshow('Crusher.jpg');
    case 3
        Info=[2000 6400 19.62 0.4 2.45 6.47 9.53];      %GBox Shaft
        axes(handles.axes4)
        imshow('2ndShaft.jpg');
    case 4
        Info=[2000 6400 24.83 0.39 2.09 3.09 4.91];     %Motor NDE Bearing
        axes(handles.axes4)
        imshow('MtrNDE.jpg');
    case 5
        Info=[2000 6400 24.83 0.4 2.43 5.21 7.79];      %Motor DE Bearing
        axes(handles.axes4)
        imshow('MtrDE.jpg');
    otherwise
```

```
end
%See opening function for Level_1_8 for notes
LOR=get(handles.popupmenu1,'Value');
switch LOR
    case 1
        FMax=Info(1);
        LOR=Info(2);
        fHz=Info(3);
        dratio=Info(4);
        mod_switch=0;
        mod_ratio=0;
    case 2
        FMax=Info(1);
        LOR=Info(2);
        fHz=Info(3);
        dratio=2*Info(5);
        mod_switch=1;
        mod_ratio=Info(4);
    case 3
        FMax=Info(1);
        LOR=Info(2);
        fHz=Info(3);
        dratio=Info(6);
        mod_switch=0;
        mod_ratio=0;
    case 4
        FMax=Info(1);
        LOR=Info(2);
        fHz=Info(3);
        dratio=Info(7);
        mod_switch=1;
        mod_ratio=1;

    otherwise
end

global Mt M mo c k T r wf fi tspan

h=waitbar(0.5,'Calculating Output, Please wait...');

M=150;                          %Machine mass
mo=50*(1/fHz^2);                %Unbalance mass
Mt=M+mo;                        %System mass
```

```
c=100000;                   %Damping coefficient
k=460000000;                %Stiffness N/m
imp=100*fHz;                %Peak pulse force N varies with speed
r=0.5;                      %Radius unbalance mass rotates at
dHz=fHz*dratio;             %Defect ratio as a function of shaft speed
T=1/dHz;                    %Defect force pulse period
wf=2*pi*fHz;                %Shaft turning speed rad/s
%wn=sqrt(k/(M+mo))          %Single degree of freedom natural frequency
%c_c=2*(M+mo)*wn            %Critical damping factor
%zeta=c/c_c                 %Damping ration
def_deg=155;                %Defect Load Zone andle 155 degrees
def_rad=def_deg/(180/pi);   %Defect Load Zone andle radians
shift=sin(def_rad);         %Factor to shifts sinwave for load zone
Wave_L=2.56*LOR;            %Number of points in the timewaveform
Samp_freq=2.56*FMax;        %Frequency of sampling in waveform
Tstep=1/Samp_freq;          %The waveform sample period
Time=LOR/FMax;              %Time/duration of waveform seconds
%Time array to control ODE solver iterations
tspan=(0:Wave_L-1)*Tstep;
d=(0:T:Time);               % Defect repetition

%Modified sinusoid to simulate bearing load zone modulation
y1=sin(mod_ratio*wf*tspan)-sin(1/mod_ratio*shift);
yscale=1/max(y1);
for n=1:length(tspan)
    if y1(n)>0
    y1(n)=y1(n)*yscale;
    else y1(n)=0;
    end
end
%Create pusele for input to ODE slover
if mod_switch==0
    y=pulstran(tspan,d,'rectpuls',0.05*T);
else
    y=y1.*(pulstran(tspan,d,'rectpuls',0.05*T));
end

fi=imp*y;
%Call solver for SDOF system
[t, x] = ode23('SDOF_SOLVER',tspan, [0 0]);
%ODE Solver results
D=x(:,1);
V=x(:,2);
```

```
A=diff(V);
A(max(size(A))+1)=0;
plot(handles.axes1,t,V*1000);       %Velocity waveform mm/s

%fft to use 100%*(2/2.56)=78.125% of Nyquist rate
Yv = fft(V,2.56*LOR)/(2.56*LOR);
YV=2*abs(Yv(1:LOR));
fV = FMax*linspace(0,1,LOR);
plot(handles.axes2,fV,YV*1000)    %Velocity spectrum mm/s

axes(handles.axes1);
xlabel('Time (s)')
ylabel('Velocity (mm/s) ')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Peak Velocity (mm/s)')

assignin ('base','D',D);
assignin ('base','V',V)
assignin ('base','A',A)
assignin ('base','tspan',tspan)
assignin ('base','fV',fV);
assignin ('base','YV',YV)

set(handles.edit3,'String',fHz);
set(handles.edit4,'String',round((fHz*dratio),2));
set(handles.edit5,'String',dratio);

close(h)

% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in popupmenu2.
function popupmenu2_Callback(hObject, eventdata, handles)

FMax=get(handles.popupmenu2,'Value');
```

```
%FMax Maximum frequency specified for spectrum
%LOR Lines of resolution for spectrum
switch FMax
    case 1  %[FMax LOR fHz FTF BSF ORDF IRDF]
        Info=[200 1600 1.67 0.45 4.74 11.2 13.8];       %Pulley
        axes(handles.axes4)
        imshow('Pulley.jpg');
    case 2
        Info=[1000 3200 7.05 0.441 4.074 9.711 12.289]; %Crusher
        axes(handles.axes4)
        imshow('Crusher.jpg');
    case 3
        Info=[2000 6400 19.62 0.4 2.45 6.47 9.53];      %GBox Speed
        axes(handles.axes4)
        imshow('2ndShaft.jpg');
    case 4
        Info=[2000 6400 24.83 0.39 2.09 3.09 4.91];     %Motor NDE
        axes(handles.axes4)
        imshow('MtrNDE.jpg');
    case 5
        Info=[2000 6400 24.83 0.4 2.43 5.21 7.79];      %Motor DE
        axes(handles.axes4)
        imshow('MtrDE.jpg');
    otherwise
end

LOR=get(handles.popupmenu1,'Value');
switch LOR
    case 1
        FMax=Info(1);
        LOR=Info(2);
        fHz=Info(3);
        dratio=Info(4);
        mod_switch=0;
        mod_ratio=0;
    case 2
        FMax=Info(1);
        LOR=Info(2);
        fHz=Info(3);
        dratio=2*Info(5);
        mod_switch=1;
        mod_ratio=Info(4);
    case 3
```

```
        FMax=Info(1);
        LOR=Info(2);
        fHz=Info(3);
        dratio=Info(6);
        mod_switch=0;
        mod_ratio=0;
    case 4
        FMax=Info(1);
        LOR=Info(2);
        fHz=Info(3);
        dratio=Info(7);
        mod_switch=1;
        mod_ratio=1;

    otherwise
end

global Mt M mo c k T r wf fi tspan

h=waitbar(0.5,'Calculating Output, Please wait...');

M=150;                          %Machine mass
mo=50*(1/fHz^2);                %Unbalance mass
Mt=M+mo;                        %System mass
c=100000;                       %Damping coefficient
k=460000000;                    %Stiffness N/m
imp=100*fHz;                    %Peak pulse force N varies with speed
r=0.5;                          %radius unbalance mass rotates at
dHz=fHz*dratio;                 %Defect ratio as a function of shaft speed
T=1/dHz;                        %Defect force pulse period
wf=2*pi*fHz;                    %Shaft turning speed rad/s
%c_c=2*(M+mo)*wn                %Critical damping factor
%zeta=c/c_c                     %Damping ratio
%wn=sqrt(k/(M+mo))              %Single degree of freedom natural frequency
def_deg=155;                    %Defect Load Zone andle 155 degrees
def_rad=def_deg/(180/pi);       %Defect Load Zone andle radians
shift=sin(def_rad);             %Factor to simulate load zone modultaion
Wave_L=2.56*LOR;                %Number of points in the timewaveform
Samp_freq=2.56*FMax;            %Frequency of sampling in waveform
Tstep=1/Samp_freq;              %The waveform sample period
Time=LOR/FMax;                  %Time/duration of waveform seconds
%Time span to control ODE iterations
tspan=(0:Wave_L-1)*Tstep;
```

```
d=(0:T:Time);               % T repetition frequency

%Create modified sinusoid for load zone modulation
y1=sin(mod_ratio*wf*tspan)-sin(1/mod_ratio*shift);
yscale=1/max(y1);
for n=1:length(tspan)
    if y1(n)>0
    y1(n)=y1(n)*yscale;
    else y1(n)=0;
    end
end

%Create pusle force for input into ODE solver
if mod_switch==0
    y=pulstran(tspan,d,'rectpuls',0.05*T);
else
    y=y1.*(pulstran(tspan,d,'rectpuls',0.05*T));
end

fi=imp*y;
%Call ODE solver
[t, x] = ode23('SDOF_SOLVER',tspan, [0 0]);
%ODE solver results
D=x(:,1);
V=x(:,2);
A=diff(V);
A(max(size(A))+1)=0;
plot(handles.axes1,t,V*1000);       %Velocity waveform mm/s

%fft to use 100%*(2/2.56)=78.125% of Nyquist rate
Yv = fft(V,2.56*LOR)/(2.56*LOR);
YV=2*abs(Yv(1:LOR));
fV = FMax*linspace(0,1,LOR);
plot(handles.axes2,fV,YV*1000)    %Velocity spectrum mm/s

axes(handles.axes1);
xlabel('Time (s)')
ylabel('Peak Velocity (mm/s) ')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Peak Velocity (mm/s)')

assignin ('base','D',D);
```

```
assignin ('base','V',V)
assignin ('base','A',A)
assignin ('base','tspan',tspan)
assignin ('base','fV',fV);
assignin ('base','YV',YV)

set(handles.edit3,'String',fHz);
set(handles.edit4,'String',round((fHz*dratio),2));
set(handles.edit5,'String',dratio);

close(h)

% --- Executes during object creation, after setting all properties.
function popupmenu2_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Plots displayed spectrum in figure separate to GUI.
function pushbutton4_Callback(hObject, eventdata, handles)

fV=evalin('base','fV');
YV=evalin('base','YV');
figure
plot(fV,YV*1000)
xlabel('Frequency (Hz)')
ylabel('Peak Velocity (mm/s) ')

% --- Plots displayed waveform in figure separate to GUI.
function pushbutton5_Callback(hObject, eventdata, handles)

V=evalin('base','V');
tspan=evalin('base','tspan');
figure
plot(tspan,V*1000)
xlabel('Time (s)')
ylabel('Velocity (mm/s) ')


% --- Executes during object creation, after setting all properties.
```

```
function axes4_CreateFcn(hObject, eventdata, handles)

function edit3_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit4_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit5_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

%END Level_1_8
```

C.11 Level_1_9

```
function varargout = Level_1_9(varargin)
% LEVEL_1_9 M-file for Level_1_9.fig
%
% Last Modified by GUIDE v2.5 24-Aug-2015 21:23:05

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Level_1_9_OpeningFcn, ...
                   'gui_OutputFcn',  @Level_1_9_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before Level_1_9 is made visible.
function Level_1_9_OpeningFcn(hObject, eventdata, handles, varargin)

% Choose default command line output for Level_1_9
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
%Displays text associated with concept covered in GUI
fid=fopen('Level_1_9.txt');
C = textscan(fid,'%s','Delimiter','\n');
C=C{1};
fclose(fid);
set(handles.edit1,'String',C);
```

```
%ISO Table displayed in GUI
axes(handles.axes4)
imshow('ISO10816_1.jpg');
%Popupmenu assigning Velocity RMS Values to class of machine
Evaluation=get(handles.popupmenu1,'Value');
switch Evaluation
    case 1  %Class 1
        v1=0.71;
        v2=1.8;
        v3=4.5;
        axes(handles.axes5)
        imshow('4kW.jpg');
    case 2  %Class 2
        v1=1.12;
        v2=2.8;
        v3=7.1;
        axes(handles.axes5)
        imshow('186kW.jpg');
    case 3  %Class 3
        v1=1.8;
        v2=4.5;
        v3=11.2;
        axes(handles.axes5)
        imshow('2350kW.jpg');
    case 4  %Class 4
        v1=2.8;
        v2=7.1;
        v3=18;
        axes(handles.axes5)
        imshow('600MW.jpg');
    otherwise
end

global Mt M mo c k T r wf fi tspan

M=200;                      %Machine mass
mo=0.05;                    %Unbalance mass
Mt=M+mo;                    %System mass
c=100000;                   %Damping coefficient Ns/m
k=46000000;                 %Stiffness N/m
imp=100;                    %Peak pulse force N
r=0.5;                      %radius unbalance mass rotates at
fHz=25;                     %Shaft turning speed Hz
```

```
dratio=6.3;                 %Defect frequency as a ratio of shaft speed
dHz=fHz*dratio;             %Defect ratio as a function of shaft speed
T=1/dHz;                    %Defect force pulse period
wf=2*pi*fHz;                %Shaft turning speed rad/s
FMax=1000;                  %Frequency maximum
LOR=1600;                   %Lines of resolution
Wave_L=2.56*LOR;            %Number of points in the timewaveform
Samp_freq=2.56*FMax;        %Frequency of sampling in waveform
Tstep=1/Samp_freq;          %The waveform sample period
Time=LOR/FMax;              %Time/duration of waveform seconds
%Time sequence to contol ODE solver iteration
tspan=(0:Wave_L-1)*Tstep;
d=(0:T:Time);               % T repetition frequency
%Small defect pulse and solver input force fi
y=pulstran(tspan,d,'rectpuls',0.05*T);
fi=imp*y;
%Calls ODE solver for SDOF system
[t, x] = ode23('SDOF_SOLVER',tspan, [0 0]);
%ODE solver results
D=x(:,1);                       %Displacement m
V=x(:,2);                       %Velocity m/s
Vmm=V*1000;                     %m to mm conversion
A=diff(V);                      %Acceleration m/s/s
A(max(size(A))+1)=0;            %Append 0
plot(handles.axes1,t,Vmm);      %plot time waveform mm/s

V_RMS=sqrt(mean((Vmm).^2));     %RMS calculation mm/s

%fft using 100%*(2/2.56)=78.125% of Nyquist rate

Yv = fft(V,2.56*LOR)/(2.56*LOR);
YV = 2*abs(Yv(1:LOR));
YVmm=1000*YV;
fV = FMax*linspace(0,1,LOR);
%plot(handles.axces2,fV,YV)

axes(handles.axes1);
xlabel('Time (s)')
ylabel('Velocity (mm/s) ')

assignin ('base','D',D);
assignin ('base','Vmm',Vmm)
assignin ('base','A',A)
```

```
assignin ('base','tspan',tspan)
assignin ('base','fV',fV);
assignin ('base','YVmm',YVmm)

set(handles.edit3,'String',round(V_RMS,2));
set(handles.edit4,'String',(fHz*dratio));

%This loop categorises the evaluation of the machine condition based on
%the RMS output and displays 'Machine Evaluation'
if  (V_RMS <= v1);
            set(handles.edit4,'String','Good')
elseif  (V_RMS > v1 && V_RMS <= v2)
            set(handles.edit4,'String','Acceptable')
elseif  (V_RMS > v2 && V_RMS <= v3)
            set(handles.edit4,'String'...
                ,'Satisfactory for short term operation')
elseif  (V_RMS > v3)
            set(handles.edit4,'String','Damaging')
end

% --- Outputs from this function are returned to the command line.
function varargout = Level_1_9_OutputFcn(hObject, eventdata, handles)
% Get default command line output from handles structure
varargout{1} = handles.output;

function edit1_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)

% --- Executes on button press in Previous.
function Previous_Callback(hObject, eventdata, handles)

run('Level_1_8')
```

```
close('Level_1_9')

% --- Executes on button press in Next.
function Next_Callback(hObject, eventdata, handles)

run('Level_1_10')
close('Level_1_9')

% --- Executes during object creation, after setting all properties.
function Amplitude_CreateFcn(hObject, eventdata, handles)

% --- Executes on selection of 'Class of Machine' in popupmenu.
function popupmenu1_Callback(hObject, eventdata, handles)
%Resets case value, recalculates,replots and evaluates machine condition
Evaluation=get(handles.popupmenu1,'Value');
switch Evaluation
    case 1
        v1=0.71;
        v2=1.8;
        v3=4.5;
        axes(handles.axes5)
        imshow('4kW.jpg');
    case 2
        v1=1.12;
        v2=2.8;
        v3=7.1;
        axes(handles.axes5)
        imshow('186kW.jpg');
    case 3
        v1=1.8;
        v2=4.5;
        v3=11.2;
        axes(handles.axes5)
        imshow('2350kW.jpg');
    case 4
        v1=2.8;
        v2=7.1;
        v3=18;
        axes(handles.axes5)
        imshow('600MW.jpg');
    otherwise
end
```

```
global Mt M mo c k T r wf fi tspan

M=200;                          %Machine mass
mo=0.05;                        %Unbalance mass
Mt=M+mo;                        %System mass
c=100000;                       %Damping coefficient
k=46000000;                     %Stiffness N/m
imp=100;                        %Peak pulse force N
r=0.5;                          %radius unbalance mass rotates at
fHz=25;                         %Shaft turning speed Hz
dratio=6.3;                     %Defect frequency as a ratio of shaft speed
dHz=fHz*dratio;                 %Defect frequency Hz
T=1/dHz;                        %Defect force pulse period
wf=2*pi*fHz;                    %Shaft turning speed rad/s
FMax=1000;                      %Maximum frequency
LOR=1600;                       %Lines of resolution
Wave_L=2.56*LOR;                %Number of points in the timewaveform
Samp_freq=2.56*FMax;            %Frequency of sampling in waveform
Tstep=1/Samp_freq;              %The waveform sample period
Time=LOR/FMax;                  %Time/duration of waveform seconds
%Time constrain on ODE solver iterations
tspan=(0:Wave_L-1)*Tstep;
d=(0:T:Time);                   % Defect repetition
%pusle defect for input to ODE solver
y=pulstran(tspan,d,'rectpuls',0.05*T);
fi=imp*y;
%Call ODE solver for SDOF system
[t, x] = ode23('SDOF_SOLVER',tspan, [0 0]);
%ODE solver results
D=x(:,1);
V=x(:,2);
Vmm=V*1000;
A=diff(V);
A(max(size(A))+1)=0;
plot(handles.axes1,t,Vmm);
%RMS of velocity waveform mm/s
V_RMS=sqrt(mean((Vmm).^2));

%fft to use 100%*(2/2.56)=78.125% of Nyquist rate
Yv = fft(V,2.56*LOR)/(2.56*LOR);
YV = 2*abs(Yv(1:LOR));          %m/s
YVmm=1000*YV;
fV = FMax*linspace(0,1,LOR);
```

```
%plot(handles.axces2,fV,YV)

axes(handles.axes1);
xlabel('Time (s)')
ylabel('Velocity (mm/s) ')

assignin ('base','D',D);
assignin ('base','Vmm',Vmm)
assignin ('base','A',A)
assignin ('base','tspan',tspan)
assignin ('base','fV',fV);
assignin ('base','YVmm',YVmm)

%Display values RMS level in GUI
set(handles.edit3,'String',round(V_RMS,2));

%This loop categorises the evaluation of the machine condition based on
%the RMS output
if  (V_RMS <= v1);
            set(handles.edit4,'String','Good')
elseif  (V_RMS > v1 && V_RMS <= v2)
            set(handles.edit4,'String','Acceptable')
elseif  (V_RMS > v2 && V_RMS <= v3)
            set(handles.edit4,'String'...
                ,'Satisfactory for short term operation')
elseif  (V_RMS > v3)
            set(handles.edit4,'String','Damaging')
end

% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Plot spectrum on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)

fV=evalin('base','fV');
YVmm=evalin('base','YVmm');
```

```
figure
plot(fV,YVmm)
xlabel('Frequency (Hz)')
ylabel('Peak Velocity (mm/s) ')

% --- Plot waveform on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)

Vmm=evalin('base','Vmm');
tspan=evalin('base','tspan');
figure
plot(tspan,Vmm)
xlabel('Time (s)')
ylabel('Velocity (mm/s) ')

% --- Executes during object creation, after setting all properties.
function axes4_CreateFcn(hObject, eventdata, handles)

function edit3_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit4_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit5_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
```

```
function edit5_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Recalculates waveform with random amplitude for evaluation.
function pushbutton6_Callback(hObject, eventdata, handles)
%Sets RMS valuse for Machine class
%Sets m1 factor to change the range of values randn generates
Evaluation=get(handles.popupmenu1,'Value');
switch Evaluation
    case 1
        v1=0.71;
        v2=1.8;
        v3=4.5;
        m1=0.1;
    case 2
        v1=1.12;
        v2=2.8;
        v3=7.1;
        m1=0.2;
    case 3
        v1=1.8;
        v2=4.5;
        v3=11.2;
        m1=0.3;
    case 4
        v1=2.8;
        v2=7.1;
        v3=18;
        m1=0.55;
    otherwise
end

global Mt M mo c k T r wf fi tspan

M=200;                      %Machine mass kg
mo=abs(randn)*m1;           %Unbalance mass * m1 factor, case/class set
Mt=M+mo;                    %System masee
c=100000;                   %Damping coefficient
```

```
k=46000000;                 %Stiffness N/m
imp=100;                    %Peak pulse force N
r=0.5;                      %radius unbalance mass rotates at
fHz=25;                     %Shaft turning speed Hz
dratio=6.3;                 %Defect frequency as a ratio of shaft speed
dHz=fHz*dratio;             %Defect frequency Hz
T=1/dHz;                    %Defect force pulse period
wf=2*pi*fHz;                %Shaft turning speed rad/s
FMax=1000;                  %Maximum frequency
LOR=1600;                   %Lines of resolution
Wave_L=2.56*LOR;            %Number of points in the timewaveform
Samp_freq=2.56*FMax;        %Frequency of sampling in waveform
Tstep=1/Samp_freq;          %The waveform sample period
Time=LOR/FMax;              %Time/duration of waveform seconds
%Contraining ODE solver iterations
tspan=(0:Wave_L-1)*Tstep;
d=(0:T:Time);               % Defect repetition
%pulse force for input to ODE solver
y=pulstran(tspan,d,'rectpuls',0.05*T);
fi=imp*y;
%Calls ODE Solve
[t, x] = ode23('SDOF_SOLVER',tspan, [0 0]);
%ODE Solver results
D=x(:,1);
V=x(:,2);
Vmm=V*1000;
A=diff(V);
A(max(size(A))+1)=0;
plot(handles.axes1,t,Vmm);   %Velocity waveform mm/s
V_RMS=sqrt(mean((Vmm).^2));  %Velocity RMS mm/s

%fft to use 100%*(2/2.56)=78.125% of Nyquist rate
%VELOCITY SPECTRUM
Yv = fft(V,2.56*LOR)/(2.56*LOR);
YV = 2*abs(Yv(1:LOR));
YVmm=1000*YV;
fV = FMax*linspace(0,1,LOR);
%plot(handles.axces2,fV,YV)

axes(handles.axes1);
xlabel('Time (s)')
ylabel('Velocity (mm/s) ')
```

```
assignin ('base','D',D);
assignin ('base','Vmm',Vmm)
assignin ('base','A',A)
assignin ('base','tspan',tspan)
assignin ('base','fV',fV);
assignin ('base','YVmm',YVmm)

%Display RMS Value in GUI
set(handles.edit3,'String',round(V_RMS,2));

%This loop categorises the evaluation of the machine condition based on
%the RMS output
if  (V_RMS <= v1);
            set(handles.edit4,'String','Good')
elseif  (V_RMS > v1) && (V_RMS <= v2)
            set(handles.edit4,'String','Acceptable')
elseif  (V_RMS > v2) && (V_RMS <= v3)
            set(handles.edit4,'String'...
                ,'Satisfactory for short term operation')
elseif  (V_RMS > v3)
            set(handles.edit4,'String','Damaging')
end

%END Level_1_9
```

C.12 Level_1_10

```
function varargout = Level_1_10(varargin)
% LEVEL_1_10 M-file for Level_1_10.fig
%
% Last Modified by GUIDE v2.5 25-Aug-2015 14:19:25

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Level_1_10_OpeningFcn, ...
                   'gui_OutputFcn',  @Level_1_10_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before Level_1_10 is made visible.
function Level_1_10_OpeningFcn(hObject, eventdata, handles, varargin)

% Choose default command line output for Level_1_10
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
%Displays text associated with concept covered in GUI
fid=fopen('Level_1_10.txt');
C = textscan(fid,'%s','Delimiter','\n');
C=C{1};
fclose(fid);
set(handles.edit1,'String',C);
```

```
%Shows image of ISO assesment table
axes(handles.axes4)
imshow('ISO10816_1.jpg');
%Sets RMS Values based on machine clasee in ISO10816
%Shows image for machine of slected Calss
Evaluation=get(handles.popupmenu1,'Value');
switch Evaluation
    case 1
        v1=0.71;
        v2=1.8;
        v3=4.5;
        axes(handles.axes5)
        imshow('4kW.jpg');
    case 2
        v1=1.12;
        v2=2.8;
        v3=7.1;
        axes(handles.axes5)
        imshow('186kW.jpg');
    case 3
        v1=1.8;
        v2=4.5;
        v3=11.2;
        axes(handles.axes5)
        imshow('2350kW.jpg');
    case 4
        v1=2.8;
        v2=7.1;
        v3=18;
        axes(handles.axes5)
        imshow('600MW.jpg');
    otherwise
end

global Mt M mo c k T r wf fi tspan

M=200;                      %Machine mass
mo=0.05;                    %Unbalance mass
Mt=M+mo;                    %System mass
c=100000;                   %Damping coefficient
k=46000000;                 %Stiffness N/m
imp=100;                    %Peak pulse force N
r=0.5;                      %radius unbalance mass rotates at
```

```
fHz=25;                         %Shaft turning speed Hz
dratio=6.3;                     %Defect frequency as ratio of shaft speed
dHz=fHz*dratio;                 %Defect frequency Hz
T=1/dHz;                        %Defect force pulse period
wf=2*pi*fHz;                    %Shaft turning speed rad/s
FMax=1000;                      %Maximum frequecy
LOR=1600;                       %Lines of resolution
Wave_L=2.56*LOR;                %Number of points in the timewaveform
Samp_freq=2.56*FMax;            %Frequency of sampling in waveform
Tstep=1/Samp_freq;              %The waveform sample period
Time=LOR/FMax;                  %Time/duration of waveform seconds
%Time constraint on iterations for ODE solver
tspan=(0:Wave_L-1)*Tstep;
d=(0:T:Time);                   % T repetition frequency
%Defect pulse for input to ODE solver
y=pulstran(tspan,d,'rectpuls',0.05*T);
fi=imp*y;
%Calls ODE solver for SDOF system
[t, x] = ode23('SDOF_SOLVER',tspan, [0 0]);
%ODE solver results
D=x(:,1);                       %Displacement
V=x(:,2);                       %Velocity m/s
Vmm=V*1000;                     %Velovity mm/s
A=diff(V);                      %Acceleration m/s/s
A(max(size(A))+1)=0;            %Append 0
plot(handles.axes1,t,Vmm);  %Plot velocity timewaveform mm/s
V_RMS=sqrt(mean((Vmm).^2)); %Calculate waveform RMS level mm/s

%fft to use 100%*(2/2.56)=78.125% of Nyquist rate
%VELOCITY SPECTRUM
Yv = fft(V,2.56*LOR)/(2.56*LOR);
YV = 2*abs(Yv(1:LOR));
YVmm=1000*YV;
fV = FMax*linspace(0,1,LOR);
%plot(handles.axces2,fV,YV)

axes(handles.axes1);
xlabel('Time (s)')
ylabel('Velocity (mm/s) ')

assignin ('base','D',D);
assignin ('base','Vmm',Vmm)
assignin ('base','A',A)
```

```
assignin ('base','tspan',tspan)
assignin ('base','fV',fV);
assignin ('base','YVmm',YVmm)
%Display RMS Level on GUI
set(handles.edit3,'String',round(V_RMS,2));
%Display machine condition based on ISO10816 table
%Not visible to user
set(handles.edit4,'String','Good');

% --- Outputs from this function are returned to the command line.
function varargout = Level_1_10_OutputFcn(hObject, eventdata, handles)

% Get default command line output from handles structure
varargout{1} = handles.output;


function edit1_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)

% --- Executes on button press in Previous.
function Previous_Callback(hObject, eventdata, handles)

run('Level_1_9')
close('Level_1_10')

% --- Executes on button press in Next.
function Next_Callback(hObject, eventdata, handles)

run('End_Level_1')
close('Level_1_10')

% --- Executes during object creation, after setting all properties.
```

```
function Amplitude_CreateFcn(hObject, eventdata, handles)

% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
%Rcalculates, plots based on machine class. Resets evaluation tabs
Evaluation=get(handles.popupmenu1,'Value');
switch Evaluation
    case 1
        v1=0.71;
        v2=1.8;
        v3=4.5;
        set(handles.popupmenu1,'Value',1)
        axes(handles.axes5)
        imshow('4kW.jpg');
    case 2
        v1=1.12;
        v2=2.8;
        v3=7.1;
        set(handles.popupmenu1,'Value',2)
        axes(handles.axes5)
        imshow('186kW.jpg');
    case 3
        v1=1.8;
        v2=4.5;
        v3=11.2;
        set(handles.popupmenu1,'Value',3)
        axes(handles.axes5)
        imshow('2350kW.jpg');
    case 4
        v1=2.8;
        v2=7.1;
        v3=18;
        set(handles.popupmenu1,'Value',4)
        axes(handles.axes5)
        imshow('600MW.jpg');
    otherwise
end

global Mt M mo c k T r wf fi tspan

M=200;                          %Machine mass
mo=0.05;                        %Unbalance mass
Mt=M+mo;                        %Machine mass
```

```
c=100000;                   %Damping coefficient
k=46000000;                 %Stiffness N/m
imp=100;                    %Peak pulse force N
r=0.5;                      %radius unbalance mass rotates at
fHz=25;                     %Shaft turning speed Hz
dratio=6.3;                 %Defect frequency as ratio of shaft speed
dHz=fHz*dratio;             %Defect frequency Hz
T=1/dHz;                    %Defect force pulse period
wf=2*pi*fHz;                %Shaft turning speed rad/s
FMax=1000;                  %Maximum frequency
LOR=1600;                   %Lines of resolution
Wave_L=2.56*LOR;            %Number of points in the timewaveform
Samp_freq=2.56*FMax;        %Frequency of sampling in waveform
Tstep=1/Samp_freq;          %The waveform sample period
Time=LOR/FMax;              %Time/duration of waveform seconds
%Time array to constrain ODE solver itreations
tspan=(0:Wave_L-1)*Tstep;
d=(0:T:Time);               % Defect repetition
%Assemble defect pulse train for input to ODE solver
y=pulstran(tspan,d,'rectpuls',0.05*T);
fi=imp*y;
%Calls ODE Solver for SDOF system
[t, x] = ode23('SDOF_SOLVER',tspan, [0 0]);
%ODE Solver results
D=x(:,1);
V=x(:,2);                       %Velocity m/s
Vmm=V*1000;
A=diff(V);
A(max(size(A))+1)=0;
plot(handles.axes1,t,Vmm);   %Plots velocity waveform mm/s
V_RMS=sqrt(mean((Vmm).^2));  %Calculates velocity RMS level mm/s

%fft to use 100%*(2/2.56)=78.125% of Nyquist rate
%VELOCITY SPECTRUM
Yv = fft(V,2.56*LOR)/(2.56*LOR);
YV = 2*abs(Yv(1:LOR));          %m/s
YVmm=1000*YV;                   %mm/s
fV = FMax*linspace(0,1,LOR);
%plot(handles.axces2,fV,YV)

axes(handles.axes1);
xlabel('Time (s)')
ylabel('Velocity (mm/s) ')
```

```
assignin ('base','D',D);
assignin ('base','Vmm',Vmm)
assignin ('base','A',A)
assignin ('base','tspan',tspan)
assignin ('base','fV',fV);
assignin ('base','YVmm',YVmm)
%Displays Velocity RMS level in GUI
set(handles.edit3,'String',round(V_RMS,2));
%Reset message and textbox colour
set(handles.text11,'String'...
    ,'Evaluate Machine Vibration Select A, B, C or D '...
    ,'BackgroundColor',[0.94 0.94 0.94]);

%This loop categorises the evaluation of the machine condition based
%on the RMS output not visible to user
if  (V_RMS <= v1);
            set(handles.edit4,'String','Good')
elseif  (V_RMS > v1 && V_RMS <= v2)
            set(handles.edit4,'String','Acceptable')
elseif  (V_RMS > v2 && V_RMS <= v3)
            set(handles.edit4,'String','Satisfactory')
elseif  (V_RMS > v3)
            set(handles.edit4,'String','Damaging')
end

% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Plots associated velocity spectrum on button press
function pushbutton4_Callback(hObject, eventdata, handles)

fV=evalin('base','fV');
YVmm=evalin('base','YVmm');
figure
plot(fV,YVmm)
xlabel('Frequency (Hz)')
```

```
ylabel('Peak Velocity (mm/s) ')

% --- Plots velocity timewaveform on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)

Vmm=evalin('base','Vmm');
tspan=evalin('base','tspan');
figure
plot(tspan,Vmm)
xlabel('Time (s)')
ylabel('Velocity (mm/s) ')

% --- Executes during object creation, after setting all properties.
function axes4_CreateFcn(hObject, eventdata, handles)

function edit3_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit4_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit5_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
%
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- 'Chang Machine' on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
%Generates a new waveform with a randomly generated amplitude
%m1 factor used in generating amplitude based on machine class
Evaluation=get(handles.popupmenu1,'Value');
switch Evaluation
    case 1
        v1=0.71;
        v2=1.8;
        v3=4.5;
        m1=0.1;
    case 2
        v1=1.12;
        v2=2.8;
        v3=7.1;
        m1=0.2;
    case 3
        v1=1.8;
        v2=4.5;
        v3=11.2;
        m1=0.3;
    case 4
        v1=2.8;
        v2=7.1;
        v3=18;
        m1=0.55;
    otherwise
end

global Mt M mo c k T r wf fi tspan

M=200;                      %Machine mass
mo=abs(randn)*m1;           %Unbalance mass randomly generated number
Mt=M+mo;                    %System mass
c=100000;                   %Damping coefficient
k=46000000;                 %Stiffness N/m
imp=100;                    %Peak pulse force N
r=0.5;                      %radius unbalance mass rotates at
```

```
fHz=25;                         %Shaft turning speed Hz
dratio=6.3;                     %Defect frequency as a ratio of shaft speed
dHz=fHz*dratio;                 %Defect frequency Hz
T=1/dHz;                        %Defect force pulse period
wf=2*pi*fHz;                    %Shaft turning speed rad/s
FMax=1000;                      %Maximum Frequency
LOR=1600;                       %Lines of resolution
Wave_L=2.56*LOR;                %Number of points in the timewaveform
Samp_freq=2.56*FMax;            %Frequency of sampling in waveform
Tstep=1/Samp_freq;              %The waveform sample period
Time=LOR/FMax;                  %Time/duration of waveform seconds
%Time sequence for constraint of ODE solver iterations
tspan=(0:Wave_L-1)*Tstep;
d=(0:T:Time);                   %Defect repetition
%Creates pulse force train for input to ODE solver
y=pulstran(tspan,d,'rectpuls',0.05*T);
fi=imp*y;
%Calls ODE solver for SDOF system
[t, x] = ode23('SDOF_SOLVER',tspan, [0 0]);
%ODE Solver results
D=x(:,1);
V=x(:,2);                       %Velocity m/s
Vmm=V*1000;                     %Velocity mm/s
A=diff(V);
A(max(size(A))+1)=0;
plot(handles.axes1,t,Vmm);  %Plot velocity waveform mm/s
V_RMS=sqrt(mean((Vmm).^2)); %Calculate Velocity RMS Level mm/s

%fft to use 100%*(2/2.56)=78.125% of Nyquist rate
%VELOCITY SPECTRUM
Yv = fft(V,2.56*LOR)/(2.56*LOR);
YV = 2*abs(Yv(1:LOR));      %m/s
YVmm=1000*YV;
fV = FMax*linspace(0,1,LOR);
%plot(handles.axces2,fV,YV)

axes(handles.axes1);
xlabel('Time (s)')
ylabel('Velocity (mm/s) ')

assignin ('base','D',D);
assignin ('base','Vmm',Vmm)
assignin ('base','A',A)
```

```
assignin ('base','tspan',tspan)
assignin ('base','fV',fV);
assignin ('base','YVmm',YVmm)
%Dispaly RMS level in GUI
set(handles.edit3,'String',round(V_RMS,2));
%Reset text box message and background colour to origional
set(handles.text11,'String'...
    ,'Evaluate Machine Vibration Select A, B, C or D '...
    ,'BackgroundColor',[0.94 0.94 0.94]);

%This loop categorises the evaluation of the machine condition based
%on the RMS output not visible to user
if  (V_RMS <= v1);
            set(handles.edit4,'String','Good')
elseif  (V_RMS > v1) && (V_RMS <= v2)
            set(handles.edit4,'String','Acceptable')
elseif  (V_RMS > v2) && (V_RMS <= v3)
            set(handles.edit4,'String','Satisfactory')
elseif  (V_RMS > v3)
            set(handles.edit4,'String','Damaging')
end


% --- Executes on button press Machine Evaluation 'A'
function pushbutton7_Callback(hObject, eventdata, handles)
%This loop evaluates the users answer of 'A' as Correct or Incorrect

PBA=cellstr(get(handles.edit4,'String'));
disp(PBA)
%edit4=PBA{get(handles.popupmenu1,'Value')};
%disp(edit4)
if  strcmp(PBA,'Good')
    set(handles.text11,'String','CORRECT','BackgroundColor','green')
else
    set(handles.text11,'String','INCORRECT','BackgroundColor','red')
end

% --- Executes on button press Machine Evaluation 'B'
function pushbutton8_Callback(hObject, eventdata, handles)
%This loop evaluates the users answer of 'B' as Correct or Incorrect

PBA=cellstr(get(handles.edit4,'String'));
disp(PBA)
```

```
%edit4=PBA{get(handles.popupmenu1,'Value')};
%disp(edit4)
if  strcmp(PBA,'Acceptable')
    set(handles.text11,'String','CORRECT','BackgroundColor','green')
else
    set(handles.text11,'String','INCORRECT','BackgroundColor','red')
end

% --- Executes on button press Machine Evaluation 'C'
function pushbutton9_Callback(hObject, eventdata, handles)
%This loop evaluates the users answer of 'B' as Correct or Incorrect

PBA=cellstr(get(handles.edit4,'String'));
disp(PBA)
%edit4=PBA{get(handles.popupmenu1,'Value')};
%disp(edit4)
if  strcmp(PBA,'Satisfactory')
    set(handles.text11,'String','CORRECT','BackgroundColor','green')
else
    set(handles.text11,'String','INCORRECT','BackgroundColor','red')
end

% --- Executes on button press Machine Evaluation 'D'
function pushbutton10_Callback(hObject, eventdata, handles)
%This loop evaluates the users answer of 'D' as Correct or Incorrect

PBA=cellstr(get(handles.edit4,'String'));
disp(PBA)
%edit4=PBA{get(handles.popupmenu1,'Value')};
%disp(edit4)
if  strcmp(PBA,'Damaging')
    set(handles.text11,'String','CORRECT','BackgroundColor','green')
else
    set(handles.text11,'String','INCORRECT','BackgroundColor','red')
end

%END Level_1_10
```

C.13 End_Level_1

```
function varargout = End_Level_1(varargin)
% END_LEVEL_1 M-file for End_Level_1.fig
% Last Modified by GUIDE v2.5 26-Aug-2015 07:44:28
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @End_Level_1_OpeningFcn, ...
                   'gui_OutputFcn',  @End_Level_1_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End_Level_1 initialization code - DO NOT EDIT

% --- Executes just before End_Level_1 is made visible.
function End_Level_1_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);

% --- Outputs from this function are returned to the command line.
function varargout = End_Level_1_OutputFcn(hObject, eventdata, handles)
% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)

run('Level_1_10')
close('End_Level_1')
```

```
% --- Executes during object creation, after setting all properties.
function pushbutton1_CreateFcn(hObject, eventdata, handles)

% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)

run('Open')
close('End_Level_1')

% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)

close('End_Level_1')
```

C.14 Level_2_1

```
function varargout = Level_2_1(varargin)
% LEVEL_2_1 M-file for Level_2_1.fig
% Last Modified by GUIDE v2.5 01-Oct-2015 10:31:28
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Level_2_1_OpeningFcn, ...
                   'gui_OutputFcn',  @Level_2_1_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before Level_2_1 is made visible.
function Level_2_1_OpeningFcn(hObject, eventdata, handles, varargin)
% Choose default command line output for Level_2_1
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
%Displays text associated with concept in GUI
fid=fopen('Level_2_1.txt');
C = textscan(fid,'%s','Delimiter','\n');
C=C{1};
fclose(fid);
set(handles.edit1,'String',C);
%Time array
t=0:0.01:11;
f=get(handles.slider2,'Value');     %Frequency from slider value Hz
w=2*pi*f;                           %Frequency radians
```

```
d=get(handles.slider1,'Value');     %Displacement amplitude from slider
D=d*sin(w*t);                       %Displacememt wave
V=diff(D);                          %Differentiate displacemet
V(length(D))=0;                     %Append zero
A=diff(V);                          %Differentiate Velocity
A(length(D))=0;                     %Append zero
%Plot as displacement, velocity and acceleration in GUI
plot(handles.axes1,t,D,t,V*10,t,A*100)
xlabel('Time (s)')
ylabel('Amplitude')
legend({'Displacement','Velocity x10','Acceleration x100'});
set(handles.axes1,'xlim', [0 10])
set(handles.axes1,'ylim',[-1.2 1.2])
grid on
%Export to workspace
assignin ('base','D',D);
assignin ('base','V',V)
assignin ('base','A',A)
assignin ('base','t',t)

% --- Outputs from this function are returned to the command line.
function varargout = Level_2_1_OutputFcn(hObject, eventdata, handles)
% Get default command line output from handles structure
varargout{1} = handles.output;

function edit1_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on change in Amplitude slider movement.
function slider1_Callback(hObject, eventdata, handles)
%Time array
t=0:0.01:11;
f=get(handles.slider2,'Value');     %Frequency from slider value
w=2*pi*f;                           %Frequency radians
d=get(handles.slider1,'Value');     %Displacement amplitude from slider
D=d*sin(w*t);                       %Displacememt wave
```

```
V=diff(D);                          %Differentiate displacement
V(length(D))=0;                     %Append zero
A=diff(V);                          %Differentiate Velocity
A(length(D))=0;                     %Append zero
%Plot as displacement, velocity and acceleration in GUI
plot(handles.axes1,t,D,t,V*10,t,A*100)
xlabel('Time (s)')
ylabel('Amplitude')
legend({'Displacement','Velocity x10','Acceleration x100'});
set(handles.axes1,'xlim', [0 10])
set(handles.axes1,'ylim',[-1.2 1.2])
grid on
%Export to workspace
assignin ('base','D',D);
assignin ('base','V',V)
assignin ('base','A',A)
assignin ('base','t',t)

% --- Executes during object creation, after setting all properties.
function slider1_CreateFcn(hObject, eventdata, handles)
% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on change in Frequency slider movement.
function slider2_Callback(hObject, eventdata, handles)
%Time array
t=0:0.01:11;
f=get(handles.slider2,'Value');     %Frequency from slider value Hz
w=2*pi*f;                           %Frequency radians
d=get(handles.slider1,'Value');     %Displacement amplitude from slider
D=d*sin(w*t);                       %Displacememt wave
V=diff(D);                          %Differentiate displacemet
V(length(D))=0;                     %Append zero
A=diff(V);                          %Differentiate Velocity
A(length(D))=0;                     %Append zero
%Plot as displacement, velocity and acceleration in GUI
plot(handles.axes1,t,D,t,V*10,t,A*100)
xlabel('Time (s)')
ylabel('Amplitude')
legend({'Displacement','Velocity x10','Acceleration x100'});
```

```
set(handles.axes1,'xlim', [0 10])
set(handles.axes1,'ylim',[-1.2 1.2])
grid on
%Export to workspace
assignin ('base','D',D);
assignin ('base','V',V)
assignin ('base','A',A)
assignin ('base','t',t)

% --- Executes during object creation, after setting all properties.
function slider2_CreateFcn(hObject, eventdata, handles)
% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)

% --- Executes on button press in Previous.
function Previous_Callback(hObject, eventdata, handles)
run('Open')
close('Level_2_1')

% --- Executes on button press in Next.
function Next_Callback(hObject, eventdata, handles)
run('Level_2_2')
close('Level_2_1')

% --- Executes during object creation, after setting all properties.
function Amplitude_CreateFcn(hObject, eventdata, handles)

% --- Displays Intgration and Differentiation Equations in GUI
% during object creation, after setting all properties.
function axes2_CreateFcn(hObject, eventdata, handles)
%Uses latex interpreter
'callback';
S1= '$$\int_{b}^{a} f(t) dt = lim_{n' ;
S2= '\to \infty}\sum_{i=1}^{n} f(t_i) \Delta t $$' ;
String1=[S1 S2];
text(0.1,0.95,'Integration','FontSize',10)
text('Interpreter','latex','String',String1...
```

```
    ,'Position',[.1 .75],'FontSize',10)
text(0.1,0.55,'Differentiation','FontSize',10)
S3= '$$ \frac{dy}{dt} = lim_{\Delta t \to 0} \frac' ;
S4= '{f(t + \Delta t) - f(t)} {\Delta t} $$' ;
String2=[S3 S4];
text('Interpreter','latex','String', String2...
    ,'Position',[.1 .35],'FontSize',10)

% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
%Time sequence for animation
tstep=0.01;
t=0:tstep:10;
%Recalculate plot based on slider values
f=get(handles.slider2,'Value');
w=2*pi*f;
d=get(handles.slider1,'Value');
D=d*sin(w*t);
V=diff(D);
V(length(D))=0;
A=diff(V);
A(length(D))=0;
set(handles.pushbutton3,'enable','off');
a=get(handles.pushbutton4,'position');
b=get(handles.pushbutton5,'position');
c=get(handles.pushbutton7,'position');
%Loop runs animation of buttons 4,5,7
pause on
for n=1:max(size(t))    %Calculates next position
    y1=a(2)+D(n)*0.18;
    y2=b(2)+V(n)*0.18*10;
    y3=c(2)+A(n)*0.18*100;
    pause(tstep)        %Move button to next position
    set(handles.pushbutton4,'position',[a(1) y1 a(3) a(4)])
    set(handles.pushbutton5,'position',[b(1) y2 b(3) b(4)])
    set(handles.pushbutton7,'position',[c(1) y3 c(3) c(4)])
    if n>=max(size(t))  %Set buttons back to start position
    set(handles.pushbutton4,'position',[a(1) 0.2944 a(3) a(4)])
    set(handles.pushbutton5,'position',[b(1) 0.2944 b(3) b(4)])
    set(handles.pushbutton7,'position',[c(1) 0.2944 c(3) c(4)])
    set(handles.pushbutton3,'enable','on');
    end
end
```

```
% --- Plot currently displayed waveform in separate Matlab figure
%on button press of Plot Waveform.
function pushbutton6_Callback(hObject, eventdata, handles)
%Extract information from workspace and plot
D=evalin('base','D');
V=evalin('base','V');
A=evalin('base','A');
t=evalin('base','t');
figure
plot(t,D,t,V*10,t,A*100)
xlim([0 10]);
xlabel('Time (s)')
ylabel('Amplitude')
legend({'Displacement','Velocity x10','Acceleration x100'});
grid on

%END Level_2_1
```

C.15 Level_2_2

```
function varargout = Level_2_2(varargin)
% LEVEL_2_2 M-file for Level_2_2.fig
% Last Modified by GUIDE v2.5 12-Sep-2015 16:44:31
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Level_2_2_OpeningFcn, ...
                   'gui_OutputFcn',  @Level_2_2_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before Level_2_2 is made visible.
function Level_2_2_OpeningFcn(hObject, eventdata, handles, varargin)
% Choose default command line output for Level_2_2
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
evalin('base','clear')
%Displays text associated with GUI concepts
fid=fopen('Level_2_2.txt');
C = textscan(fid,'%s','Delimiter','\n');
C=C{1};
fclose(fid);
set(handles.edit1,'String',C);
%Data creation parameters
FMax=200;
LOR=800;
```

```
WavePoints=2.56*LOR;
Sampfreq=2.56*FMax;
TimeStep=1/Sampfreq;
t=(0:WavePoints-1)*TimeStep;
%Randomly generate wave frequencies and amplitudes
f1=200*rand;
f2=200*rand;
f3=200*rand;
f4=200*rand;
f5=200*rand;
f6=200*rand;
Amplitude1=5*rand;
Amplitude2=5*rand;
Amplitude3=5*rand;
Amplitude4=5*rand;
Amplitude5=5*rand;
Amplitude6=5*rand;
w1=2*pi*f1;
w2=2*pi*f2;
w3=2*pi*f3;
w4=2*pi*f4;
w5=2*pi*f5;
w6=2*pi*f6;
wave1=Amplitude1*sin(w1*t);
wave2=Amplitude2*sin(w2*t);
wave3=Amplitude3*sin(w3*t);
wave4=Amplitude4*sin(w4*t);
wave5=Amplitude5*sin(w5*t);
wave6=Amplitude6*sin(w6*t);
%Compile and plot wave
wave=wave1+wave2+wave3+wave4+wave5+wave6;
plot(handles.axes1,t,wave)  %,t,wave1,t,wave2)
set(handles.axes1,'ylim',[1.1*min(wave) 1.1*max(wave)]);
set(handles.axes1,'xlim',[0 0.5])
%fft useing 100%*(2/2.56)=78.125% of the Nyquist rate
Y = fft(wave,2.56*LOR)/(2.56*LOR);
f = FMax*linspace(0,1,LOR);
plot(handles.axes2,f,2*abs(Y(1:LOR)))
set(handles.axes2,'ylim',[0 5])

axes(handles.axes1);
xlabel('Time (s)')
ylabel('Amplitude')
```

```
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Amplitude')

assignin ('base','wave',wave)
assignin ('base','wave1',wave1)
assignin ('base','wave2',wave2)
assignin ('base','t',t)

% --- Outputs from this function are returned to the command line.
function varargout = Level_2_2_OutputFcn(hObject, eventdata, handles)
% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)

% --- Executes on button press in Previous.
function Previous_Callback(hObject, eventdata, handles)
run('Level_2_1')
close('Level_2_2')

% --- Executes on button press in Next.
function Next_Callback(hObject, eventdata, handles)
run('Level_2_3')
close('Level_2_2')

% --- Executes during object creation, after setting all properties.
function Amplitude_CreateFcn(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function axes2_CreateFcn(hObject, eventdata, handles)

% --- Generates and plot new waveform on press of Change Waveform
function pushbutton3_Callback(hObject, eventdata, handles)
```

```
%Data sampling parameters
FMax=200;
LOR=800;
WavePoints=2.56*LOR;
Sampfreq=2.56*FMax;
TimeStep=1/Sampfreq;
t=(0:WavePoints-1)*TimeStep;
%Randomly generate enerate new waveform
f1=200*rand;
f2=200*rand;
f3=200*rand;
f4=200*rand;
f5=200*rand;
f6=200*rand;
Amplitude1=5*rand;
Amplitude2=5*rand;
Amplitude3=5*rand;
Amplitude4=5*rand;
Amplitude5=5*rand;
Amplitude6=5*rand;
w1=2*pi*f1;
w2=2*pi*f2;
w3=2*pi*f3;
w4=2*pi*f4;
w5=2*pi*f5;
w6=2*pi*f6;
wave1=Amplitude1*sin(w1*t);
wave2=Amplitude2*sin(w2*t);
wave3=Amplitude3*sin(w3*t);
wave4=Amplitude4*sin(w4*t);
wave5=Amplitude5*sin(w5*t);
wave6=Amplitude6*sin(w6*t);
%Compile waveform and plot
wave=wave1+wave2+wave3+wave4+wave5+wave6;
plot(handles.axes1,t,wave)  %,t,wave1,t,wave2)
set(handles.axes1,'ylim',[1.1*min(wave) 1.1*max(wave)]);
set(handles.axes1,'xlim',[0 0.5])
%fft using 100%*(2/2.56)=78.125% of the Nyquist rate
Y = fft(wave,2.56*LOR)/(2.56*LOR);
f = FMax*linspace(0,1,LOR);
plot(handles.axes2,f,2*abs(Y(1:LOR)))
set(handles.axes2,'ylim',[0 5])
```

```
axes(handles.axes1);
xlabel('Time (s)')
ylabel('Amplitude')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Amplitude')

assignin ('base','wave',wave)
assignin ('base','wave1',wave1)
assignin ('base','wave2',wave2)
assignin ('base','t',t)

%END Level_2_2
```

C.16 Level_2_3

```
function varargout = Level_2_3(varargin)
% LEVEL_2_3 M-file for Level_2_3.fig
%
% Last Modified by GUIDE v2.5 21-Oct-2015 09:07:20

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Level_2_3_OpeningFcn, ...
                   'gui_OutputFcn',  @Level_2_3_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before Level_2_3 is made visible.
function Level_2_3_OpeningFcn(hObject, eventdata, handles, varargin)
% Choose default command line output for Level_2_3
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
%Clear workspace
evalin('base','clear')
%Display text associated with the GUI concepts
fid=fopen('Level_2_3.txt');
C = textscan(fid,'%s','Delimiter','\n');
C=C{1};
fclose(fid);
set(handles.edit1,'String',C);
```

```
%Set data generation parameter
FMax=200;
LOR=800;
WavePoints=2.56*LOR;
Sampfreq=2.56*FMax;
TimeStep=1/Sampfreq;
t=(0:WavePoints-1)*TimeStep;
%Randomly generate frequencies and assign to workspace
f1=194*rand;
f2=194*rand;
f3=194*rand;
f4=194*rand;
f5=194*rand;
f6=194*rand;
f7=f6+5;
f8=abs(f6-5);
assignin('base','f1',f1);
assignin('base','f2',f2);
assignin('base','f3',f3);
assignin('base','f4',f4);
assignin('base','f5',f5);
assignin('base','f6',f6);
assignin('base','f7',f7);
assignin('base','f8',f8);
%Randomly generate amplitudes and assign to workspace
Amp1=5*rand;
Amplitude1=abs(Amp1+0.5*randn);
Amp2=5*rand;
Amplitude2=abs(Amp2+0.5*randn);
Amp3=5*rand;
Amplitude3=abs(Amp3+0.5*randn);
Amp4=5*rand;
Amplitude4=abs(Amp4+0.5*randn);
Amp5=5*rand;
Amplitude5=abs(Amp5+0.5*randn);
Amp6=5*rand;
Amplitude6=abs(Amp6+0.5*randn);
Amp7=0.7*Amp6;
Amplitude7=Amp7;
Amp8=Amp7;
Amplitude8=Amp7;
assignin('base','Amp1',Amp1);
assignin('base','Amp2',Amp2);
```

```
assignin('base','Amp3',Amp3);
assignin('base','Amp4',Amp4);
assignin('base','Amp5',Amp5);
assignin('base','Amp6',Amp6);
assignin('base','Amp7',Amp7);
assignin('base','Amp8',Amp8);
%Compile waves
w1=2*pi*f1;
w2=2*pi*f2;
w3=2*pi*f3;
w4=2*pi*f4;
w5=2*pi*f5;
w6=2*pi*f6;
w7=2*pi*f7;
w8=2*pi*f8;
wave1=Amplitude1*sin(w1*t);
wave2=Amplitude2*sin(w2*t);
wave3=Amplitude3*sin(w3*t);
wave4=Amplitude4*sin(w4*t);
wave5=Amplitude5*sin(w5*t);
wave6=Amplitude6*sin(w6*t);
wave7=Amplitude7*sin(w7*t);
wave8=Amplitude8*sin(w8*t);
wave=wave1+wave2+wave3+wave4+wave5+wave6+wave7+wave8;
%Add random noise
noise=20*rand;
wave=wave+noise*randn(size(wave));

plot(handles.axes1,t,wave)
set(handles.axes1,'ylim',[1.1*min(wave) 1.1*max(wave)]);

%fft using 100%*(2/2.56)=78.125% of Nyquist rate
Y = fft(wave,2.56*LOR)/(2.56*LOR);
f = FMax*linspace(0,1,LOR);
plot(handles.axes2,f,2*abs(Y(1:LOR)))
set(handles.axes2,'ylim',[0 5])

axes(handles.axes1);
xlabel('Time (s)')
ylabel('Amplitude')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Amplitude')
```

```
%Send information to workspace
assignin ('base','wave',wave)
assignin ('base','LOR',LOR)
assignin ('base','Sampfreq',Sampfreq)
assignin ('base','FMax',FMax)
assignin ('base','t',t)
assignin ('base','f',f)
assignin ('base','noise',noise)
%Lower and upper ends of frequency band selection
Fpass1=get(handles.slider3,'Value');
set(handles.edit3,'String',Fpass1);
Fpass2=get(handles.slider4,'Value');
set(handles.edit4,'String',Fpass2);
%Stop the envelope function from being used
set(handles.radiobutton2,'enable','off');
%Calculate, display, workspace, RMS, Crest Factor, Kurtosis Values
RMS=round(rms(wave),2);
CF=round(peak2rms(wave),2);
K=round(kurtosis(wave),2);
set(handles.edit5,'String',RMS)
set(handles.edit6,'String',CF)
set(handles.edit7,'String',K)
assignin('base','RMSAv',0)
assignin('base','CFAv',0)
assignin('base','KAv',0)
%Display the number of averages as per slider bar value
sl5=get(handles.slider5,'Value');
set(handles.edit8,'String',sl5)
%Enable pause for displaying average
pause on

% --- Outputs from this function are returned to the command line.
function varargout = Level_2_3_OutputFcn(hObject, eventdata, handles)
% Get default command line output from handles structure
varargout{1} = handles.output;

function edit1_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end

% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)

% --- Executes on button press in Previous.
function Previous_Callback(hObject, eventdata, handles)
pause off
run('Level_2_2')
close('Level_2_3')

% --- Executes on button press in Next.
function Next_Callback(hObject, eventdata, handles)
pause off
h=waitbar(0.5,'Opening Next Level, Please wait...');
run('Level_2_4')
close('Level_2_3')
close(h)

% --- Executes during object creation, after setting all properties.
function Amplitude_CreateFcn(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function axes2_CreateFcn(hObject, eventdata, handles)

% --- Generate new waveform on button press Change Waveform
function pushbutton3_Callback(hObject, eventdata, handles)
%Allow filter band button to be used
set(handles.radiobutton1,'enable','on')
%Do not allow the envelope button to be used
set(handles.radiobutton2,'enable','off')
%Set data generation parameters
FMax=200;
LOR=800;
WavePoints=2.56*LOR;
Sampfreq=2.56*FMax;
TimeStep=1/Sampfreq;
t=(0:WavePoints-1)*TimeStep;
%Randomly generate frequencies and send to workspace
f1=194*rand;
f2=194*rand;
f3=194*rand;
```

```
f4=194*rand;
f5=194*rand;
f6=194*rand;
f7=f6+5;
f8=abs(f6-5);
assignin('base','f1',f1);
assignin('base','f2',f2);
assignin('base','f3',f3);
assignin('base','f4',f4);
assignin('base','f5',f5);
assignin('base','f6',f6);
assignin('base','f7',f7);
assignin('base','f8',f8);
%Randomly generate amplitudes and send to workspace
Amp1=5*rand;
Amplitude1=abs(Amp1+0.5*randn);
Amp2=5*rand;
Amplitude2=abs(Amp2+0.5*randn);
Amp3=5*rand;
Amplitude3=abs(Amp3+0.5*randn);
Amp4=5*rand;
Amplitude4=abs(Amp4+0.5*randn);
Amp5=5*rand;
Amplitude5=abs(Amp5+0.5*randn);
Amp6=5*rand;
Amplitude6=abs(Amp6+0.5*randn);
Amp7=0.7*Amp6;
Amplitude7=Amp7;
Amp8=Amp7;
Amplitude8=Amp7;
assignin('base','Amp1',Amp1);
assignin('base','Amp2',Amp2);
assignin('base','Amp3',Amp3);
assignin('base','Amp4',Amp4);
assignin('base','Amp5',Amp5);
assignin('base','Amp6',Amp6);
assignin('base','Amp7',Amp7);
assignin('base','Amp8',Amp8);
%Compile waveform
w1=2*pi*f1;
w2=2*pi*f2;
w3=2*pi*f3;
w4=2*pi*f4;
```

```
w5=2*pi*f5;
w6=2*pi*f6;
w7=2*pi*f7;
w8=2*pi*f8;
wave1=Amplitude1*sin(w1*t);
wave2=Amplitude2*sin(w2*t);
wave3=Amplitude3*sin(w3*t);
wave4=Amplitude4*sin(w4*t);
wave5=Amplitude5*sin(w5*t);
wave6=Amplitude6*sin(w6*t);
wave7=Amplitude7*sin(w7*t);
wave8=Amplitude8*sin(w8*t);
wave=wave1+wave2+wave3+wave4+wave5+wave6+wave7+wave8;
%Add random noise to waveform
noise=20*rand;
wave=wave+noise*randn(size(wave));

plot(handles.axes1,t,wave)
set(handles.axes1,'ylim',[1.1*min(wave) 1.1*max(wave)]);
%fft using 100%*(2/2.56)=78.125% of Nyquist rate
Y = fft(wave,2.56*LOR)/(2.56*LOR);
f = FMax*linspace(0,1,LOR);
plot(handles.axes2,f,2*abs(Y(1:LOR)))
%set(handles.axes2,'ylim',[0 5])

axes(handles.axes1);
xlabel('Time (s)')
ylabel('Amplitude')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Amplitude')
%Send information to workspace
assignin ('base','wave',wave)
assignin ('base','LOR',LOR)
assignin ('base','Sampfreq',Sampfreq)
assignin ('base','FMax',FMax)
assignin ('base','t',t)
assignin ('base','f',f)
assignin('base','noise',noise)
%Calculate, display, workspace, RMS, Crest Factor, Kurtosis Values
RMS=round(rms(wave),2);
CF=round(peak2rms(wave),2);
K=round(kurtosis(wave),2);
```

```
set(handles.edit5,'String',RMS)
set(handles.edit6,'String',CF)
set(handles.edit7,'String',K)
assignin('base','RMSAv',0)
assignin('base','CFAv',0)
assignin('base','KAv',0)

% --- Filters displayed waveform on button press for Filter Band
function radiobutton1_Callback(hObject, eventdata, handles)
%Lower and upper ends of frequency band selection
Fpass1=round(get(handles.slider3,'Value'));
Fpass2=round(get(handles.slider4,'Value'));
%Extract time, the wave and sample parameter from the workspace
wave=evalin('base','wave');
FMax=evalin('base','FMax');
Fs=evalin('base','Sampfreq');   % Sampling Frequency
LOR=evalin('base','LOR');
t=evalin('base','t');

rb=get(handles.radiobutton1,'Value');
if rb==1;
    %Stop user from generating a new waveform while using filter
    set(handles.pushbutton3,'enable','off');
    %Stop user from using the average function while using filter
    set(handles.radiobutton3,'enable','off');
% Frequency values are in Hz.

Fstop1 = Fpass1-3;              % First Stopband Frequency
Fstop2 = Fpass2+3;              % Second Stopband Frequency
Dstop1 = 0.0001;                % First Stopband Attenuation
Dpass  = 0.057501127785;        % Passband Ripple
Dstop2 = 0.0001;                % Second Stopband Attenuation
dens   = 20;                    % Density Factor

% Calculate the order from the parameters using FIRPMORD.(MATLAB CODE)
[N, Fo, Ao, W] = firpmord([Fstop1 Fpass1 Fpass2 Fstop2]/(Fs/2), [0 1 ...
                          0], [Dstop1 Dpass Dstop2]);
% Calculate the coefficients using the FIRPM function.
b  = firpm(N, Fo, Ao, W, {dens});
Hd = dfilt.dffir(b);                                    %(MATLAB CODE)

%grpdelay(Hd,length(wave),Fs)   % plot group delay
D = round(mean(grpdelay(Hd))); % filter delay in sample   (MATLAB CODE)
```

```
wavefilt = filter(Hd,[wave,zeros(1,D)]);% Append D zeros to the input data
wavefilt = wavefilt(D+1:end);          % Shift data to compensate for delay
%Send filtre and filtered waveform to workspace
assignin ('base','Hd',Hd)
assignin ('base','wavefilt',wavefilt)
%fft using 100%*(2/2.56)=78.125% of Nyquist rate
Y = fft(wavefilt,2.56*LOR)/(2.56*LOR);
f = FMax*linspace(0,1,LOR);
plot(handles.axes2,f,2*abs(Y(1:LOR)))
set(handles.axes2,'ylim',[0 5])
%Hilbert transform applied to filtered waveform as envelope
envelope=abs(hilbert(wavefilt));
%fft of the Hilbert transform envelope
envfft = fft(envelope,2.56*LOR)/(2.56*LOR);
envspect=2*abs(envfft(1:LOR));
plot(handles.axes1,t,wavefilt)%Plot filtered waveform
%send envelope, fft of filtered waveform and envelope to workspace
assignin('base','Y',Y)
assignin('base','envelope',envelope)
assignin('base','envspect',envspect)
assignin('base','f',f)
%Allow user to use the envelope radiobutton
set(handles.radiobutton2,'enable','on');
%Conditions if Filter Band is turned off (Value=0)
else
    set(handles.pushbutton3,'enable','on');
    set(handles.radiobutton3,'enable','on');
    set(handles.radiobutton2,'value',0);
    set(handles.radiobutton2,'enable','off');
    plot(handles.axes1,t,wave)
    Y = fft(wave,2.56*LOR)/(2.56*LOR);
    f = FMax*linspace(0,1,LOR);
    plot(handles.axes2,f,2*abs(Y(1:LOR)))
end

% --- Executes during object creation, after setting all properties.
function radiobutton1_CreateFcn(hObject, eventdata, handles)

function edit3_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit4_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Displays lower band pass frequency on slider movement.
function slider3_Callback(hObject, eventdata, handles)

Fpass1=round(get(handles.slider3,'Value'));
set(handles.edit3,'String',Fpass1);
Fpass2=round(get(handles.slider4,'Value'));
set(handles.edit4,'String',Fpass2);

% --- Executes during object creation, after setting all properties.
function slider3_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Displays upper band pass frequency on slider movement.
function slider4_Callback(hObject, eventdata, handles)

Fpass1=round(get(handles.slider3,'Value'));
set(handles.edit3,'String',Fpass1);
Fpass2=round(get(handles.slider4,'Value'));
set(handles.edit4,'String',Fpass2);

% --- Executes during object creation, after setting all properties.
function slider4_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Plots the Hilbert transform and it fft on button press Envelope
function radiobutton2_Callback(hObject, eventdata, handles)
%Get value for if logic
rb=get(handles.radiobutton1,'Value');
rb2=get(handles.radiobutton2,'Value');
%Extract required information from workspace
Y=evalin('base','Y');
wavefilt=evalin('base','wavefilt');
envelope=evalin('base','envelope');
envspect=evalin('base','envspect');
f=evalin('base','f');
t=evalin('base','t');
LOR=evalin('base','LOR');

if rb==1 && rb2==1; %Plots Hilbert Transform Information
    set(handles.pushbutton3,'enable','off');
    plot(handles.axes1,t,wavefilt,t,envelope,'LineWidth',0.8)
    plot(handles.axes2,f,2*abs(Y(1:LOR)),f,2*envspect)
else                %Returns to filtered polts
    plot(handles.axes1,t,wavefilt)
    plot(handles.axes2,f,2*abs(Y(1:LOR)))
end

function edit5_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit6_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end

function edit7_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit7_CreateFcn(hObject, eventdata, handles)
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit8_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit8_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Sets number of waveform to be averaged on slider movement.
function slider5_Callback(hObject, eventdata, handles)

sl5=round(get(handles.slider5,'Value'));
set(handles.edit8,'String',sl5)

% --- Executes during object creation, after setting all properties.
function slider5_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Performs the averageing process on button press Averaging
function radiobutton3_Callback(hObject, eventdata, handles)
%Number of waveforms to be averaged
sl5=round(get(handles.slider5,'Value'));
assignin('base','sl5',sl5)
%Stop other functions while conducting this
```

```
set(handles.radiobutton1,'enable','off')
set(handles.radiobutton2,'enable','off')
set(handles.radiobutton3,'enable','off')
%Use amplitudes and frequencies from workspace
Amp1=evalin('base','Amp1');
Amp2=evalin('base','Amp2');
Amp3=evalin('base','Amp3');
Amp4=evalin('base','Amp4');
Amp5=evalin('base','Amp5');
Amp6=evalin('base','Amp6');
Amp7=evalin('base','Amp7');
Amp8=evalin('base','Amp8');
f1=evalin('base','f1');
f2=evalin('base','f2');
f3=evalin('base','f3');
f4=evalin('base','f4');
f5=evalin('base','f5');
f6=evalin('base','f6');
f7=evalin('base','f7');
f8=evalin('base','f8');
LOR=evalin('base','LOR');
t=evalin('base','t');
f=evalin('base','f');
%This will create a similar amount of noise in all average samples
noise=evalin('base','noise');
%Start the average process at zero to add first waveform
Y=zeros(1,800);
assignin('base','Av',Y)
%Loop assembles new waveform with frequencies and amplitudes and noise
%taken from the workspace to generate a similar waveform with variation as
%would be seen in operation of a machine.
for n=1:sl5;
    pause(1.5)  %Allows display of each wavefom used in average
    Amplitude1=abs(Amp1+0.5*randn);
    Amplitude2=abs(Amp2+0.5*randn);
    Amplitude3=abs(Amp3+0.5*randn);
    Amplitude4=abs(Amp4+0.5*randn);
    Amplitude5=abs(Amp5+0.5*randn);
    Amplitude6=abs(Amp6+0.5*randn);
    Amplitude7=0.7*Amplitude6;
    Amplitude8=Amplitude7;
    w1=2*pi*f1;
    w2=2*pi*f2;
```

```
w3=2*pi*f3;
w4=2*pi*f4;
w5=2*pi*f5;
w6=2*pi*f6;
w7=2*pi*f7;
w8=2*pi*f8;
wave1=Amplitude1*sin(w1*t);
wave2=Amplitude2*sin(w2*t);
wave3=Amplitude3*sin(w3*t);
wave4=Amplitude4*sin(w4*t);
wave5=Amplitude5*sin(w5*t);
wave6=Amplitude6*sin(w6*t);
wave7=Amplitude7*sin(w7*t);
wave8=Amplitude8*sin(w8*t);
wave=wave1+wave2+wave3+wave4+wave5+wave6+wave7+wave8;
wave=wave+noise*randn(size(wave));
%Fft using 78.125% of Nyquist rate and plots current waveform in GUI
%for 1.5 seconds
Y = fft(wave,2.56*LOR)/(2.56*LOR);
Y = 2*abs(Y(1:LOR));
plot(handles.axes1,t,wave)
plot(handles.axes2,f,Y)
axes(handles.axes1);
xlabel('Time (s)')
ylabel('Amplitude')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Amplitude')
%Takes Av from the workspace (initially zeros) and adds the wave
%generated in this loop and sends the sum to the workspace for use in
%the next iteration of the loop.
Av=evalin('base','Av');
Av=(Av+Y);
assignin('base','Av',Av)
%Used to obtain average values for RMS, CF and K, displays current
%waveform value in GUI
RMS=round(rms(wave),2);
CF=round(peak2rms(wave),2);
K=round(kurtosis(wave),2);
set(handles.edit5,'String',RMS)
set(handles.edit6,'String',CF)
set(handles.edit7,'String',K)
RMSAv=evalin('base','RMSAv');
```

```
    RMSAv=(RMSAv+RMS);
    assignin('base','RMSAv',RMSAv)
    CFAv=evalin('base','CFAv');
    CFAv=(CFAv+CF);
    assignin('base','CFAv',CFAv)
    KAv=evalin('base','KAv');
    KAv=(KAv+K);
    assignin('base','KAv',KAv)
    %When number of averages reaches that desired by user this allows 1.5
    %second pause to display last wave/fft going into avearge and uses the
    %summed Av value divided by number of ffts in average and plots
    if n==sl5
        pause(1.5)
    plot(handles.axes2,f,(Av/sl5))
    axes(handles.axes2);
    xlabel('Frequency (Hz)')
    ylabel('Amplitude')
    legend('Averaged Signal');
    %Displays average values
    RMSAv=evalin('base','RMSAv');
    CFAv=evalin('base','CFAv');
    KAv=evalin('base','KAv');
    set(handles.edit5,'String',round(RMSAv/(sl5),2))
    set(handles.edit6,'String',round(CFAv/(sl5),2))
    set(handles.edit7,'String',round(KAv/(sl5),2))
    else
    end
end
%Resets values so the loop can be used again
set(handles.radiobutton3,'enable','on')
set(handles.radiobutton3,'Value',0)
assignin('base','RMSAv',0)
assignin('base','CFAv',0)
assignin('base','KAv',0)

%END Level_2_3
```

C.17 Level_2_4

```
function varargout = Level_2_4(varargin)
% LEVEL_2_4 M-file for Level_2_4.fig
%
% Last Modified by GUIDE v2.5 21-Oct-2015 09:06:54

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Level_2_4_OpeningFcn, ...
                   'gui_OutputFcn',  @Level_2_4_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Level_2_4 is made visible.
function Level_2_4_OpeningFcn(hObject, eventdata, handles, varargin)

% Choose default command line output for Level_2_4
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);

global M C K I wf mo r fi tspan
%Displays text associated with GUI concepts
fid=fopen('Level_2_4.txt');
C = textscan(fid,'%s','Delimiter','\n');
C=C{1};
```

```
fclose(fid);
set(handles.edit1,'String',C);

FMax=get(handles.popupmenu2,'Value');
switch FMax
    case 1  %[FMax LOR fHz  FTF  BSF  ORDF IRDF]
        Info=[2000 12800 1.67 0.45 4.74 11.2 13.8];     %Conveyor pulley
        axes(handles.axes4)
        imshow('Pulley.jpg');
    case 2
        Info=[10000 25600 7.05 0.441 4.074 9.711 12.289];%Crusher shaft
        axes(handles.axes4)
        imshow('Crusher.jpg');
    case 3
        Info=[10000 25600 19.62 0.4 2.45 6.47 9.53];        %GBox shaft
        axes(handles.axes4)
        imshow('2ndShaft.jpg');
    case 4
        Info=[10000 25600 24.83 0.39 2.09 3.09 4.91];        %Motor NDE
        axes(handles.axes4)
        imshow('MtrNDE.jpg');
    case 5
        Info=[10000 25600 24.83 0.4 2.43 5.21 7.79];          %Motor DE
        axes(handles.axes4)
        imshow('MtrDE.jpg');
    otherwise
end

LOR=get(handles.popupmenu1,'Value');
switch LOR
    case 1
        FMax=Info(1);
        LOR=Info(2);
        fHz=Info(3);
        dratio=Info(4);
        mod_switch=0;
        mod_ratio=0;
    case 2
        FMax=Info(1);
        LOR=Info(2);
        fHz=Info(3);
        dratio=2*Info(5);
        mod_switch=1;
```

```
        mod_ratio=Info(4);
    case 3
        FMax=Info(1);
        LOR=Info(2);
        fHz=Info(3);
        dratio=Info(6);
        mod_switch=0;
        mod_ratio=0;
    case 4
        FMax=Info(1);
        LOR=Info(2);
        fHz=Info(3);
        dratio=Info(7);
        mod_switch=1;
        mod_ratio=1;
    otherwise
end
%Two degree of freedom syatem parameters
m1=100;                     %Component 1 mass
m2=600;                     %Component 2 mass
mo=500*(1/fHz^2);           %Unbalance mass at component 1
c1=900000;                  %Damping coefficient 1 Ns/m
c2=2500000;                 %Damping coefficient 2 Ns/m
k1=40000000000;             %Stiffness N/m
k2=80000000000;             %Stiffness N/m
g=9.81;                     %Gravity
imp=100000*fHz;             %Peak pulse force N varies with speed
r=0.5;                      %radius unbalance mass rotates at
dHz=fHz*dratio;             %Defect frequency Hz
T=1/dHz;                    %Defect force pulse period
wf=2*pi*fHz;                %Shaft turning speed rad/s
cf=mo*r*wf^2;               %Centrifugal force
%Matrices for inertia M,damping C, stiffness K and an identity matrix I.
M=[m1+mo,0;0,m2];
C=[c1,-c1;-c1,c1+c2];
K=[k1,-k1;-k1,k1+k2];
I=[1,0;0,1];
Wave_L=2.56*LOR;            %Number of points in the timewaveform
Samp_freq=2.56*FMax;        %Frequency of sampling in waveform
Tstep=1/Samp_freq;          %The waveform sample period
Time=LOR/FMax;              %Time/duration of waveform seconds
%Time contraint for ODE solver
tspan=(0:Wave_L-1)*Tstep;
```

```
d=(0:T:Time);               % T repetition frequency
%Modified sinusoid for load zone modulation
y1=sin(mod_ratio*wf*tspan)-0.2165;
yscale=1/max(y1);
for n=1:length(tspan)
    if y1(n)>0
    y1(n)=y1(n)*yscale;
    else y1(n)=0;
    end
end
%Construct pulse train for use in ODE solution
if mod_switch==0
    y=pulstran(tspan,d,'rectpuls',0.02*T);
else
    y=y1.*(pulstran(tspan,d,'rectpuls',0.02*T));
end
fi=imp*y;
%Call ODE solver for two degree of freedom system
[t, x] = ode23('TDOF_SOLVER',tspan, [0 0 0 0]);
%Results
D=x(:,1);
V=x(:,2);
A=diff(V);
A(max(size(A))+1)=0;
A=A/g;
plot(handles.axes1,t,V*1000);       %1000 converts m to mm
%fft using 100%*(2/2.56)=78.125% of Nyquist rate
%VELOCITY SPECTRUM and plotting
Yv = fft(V,2.56*LOR)/(2.56*LOR);
YV=2*abs(Yv(1:LOR));
f = FMax*linspace(0,1,LOR);
plot(handles.axes2,f,YV*1000)    %1000 converts m to mm
%ACCELERATION SPECTRUM and plotting
Ya = fft(A,2.56*LOR)/(2.56*LOR);
YA=2*abs(Ya(1:LOR));        %m/s^2
YA=YA/g;
axes(handles.axes1);
xlabel('Time (s)')
ylabel('Velocity (mm/s) ')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Peak Velocity (mm/s)')
%Send information to workspace
```

```
assignin ('base','D',D);
assignin ('base','V',V)
assignin ('base','A',A)
assignin ('base','tspan',tspan)
assignin ('base','f',f);
assignin ('base','YV',YV)
assignin ('base','YA',YA)
assignin ('base','LOR',LOR)
assignin ('base','Samp_freq',Samp_freq)
assignin('base','FMax',FMax)
%Display values on GUI
set(handles.edit3,'String',fHz);
set(handles.edit4,'String',round((fHz*dratio),2));
set(handles.edit5,'String',dratio);
Fpass1=round(get(handles.slider3,'Value'));
set(handles.edit6,'String',Fpass1);
Fpass2=round(get(handles.slider4,'Value'));
set(handles.edit7,'String',Fpass2);
%Set functions as off so they cannot be used at this stage
set(handles.radiobutton4,'enable','off');
set(handles.radiobutton5,'enable','off');
set(handles.radiobutton6,'enable','off');

% --- Outputs from this function are returned to the command line.
function varargout = Level_2_4_OutputFcn(hObject, eventdata, handles)
% Get default command line output from handles structure
varargout{1} = handles.output;


function edit1_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)

% --- Executes on button press in Previous.
```

```
function Previous_Callback(hObject, eventdata, handles)

run('Level_2_3')
close('Level_2_4')

% --- Executes on button press in Next.
function Next_Callback(hObject, eventdata, handles)

run('Level_2_5')
close('Level_2_4')

% --- Executes during object creation, after setting all properties.
function Amplitude_CreateFcn(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function axes2_CreateFcn(hObject, eventdata, handles)

% --- Executes on selecting a Non Synchronous Defect in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
%Enable use of Velocity and Acceleration radiobuttons
set(handles.radiobutton1,'enable','on');
set(handles.radiobutton3,'enable','on');
%Reset Filter Band and Envelope radiobuttons
set(handles.radiobutton4,'Value',0);
set(handles.radiobutton5,'Value',0);
%Select current Machine Component displayed in GUI
FMax=get(handles.popupmenu2,'Value');
switch FMax
    case 1  %[FMax LOR fHz  FTF  BSF  ORDF IRDF]
        Info=[2000 12800 1.67 0.45 4.74 11.2 13.8];
        axes(handles.axes4)
        imshow('Pulley.jpg');
    case 2
        Info=[10000 25600 7.05 0.441 4.074 9.711 12.289];
        axes(handles.axes4)
        imshow('Crusher.jpg');
    case 3
        Info=[10000 25600 19.62 0.4 2.45 6.47 9.53];
        axes(handles.axes4)
        imshow('2ndShaft.jpg');
    case 4
        Info=[10000 25600 24.83 0.39 2.09 3.09 4.91];       %
        axes(handles.axes4)
```

```
        imshow('MtrNDE.jpg');
    case 5
        Info=[10000 25600 24.83 0.4 2.43 5.21 7.79];
        axes(handles.axes4)
        imshow('MtrDE.jpg');
    otherwise
end
%Select information based on Non Synchronous Defect chosen in popupmenu
LOR=get(handles.popupmenu1,'Value');
switch LOR
    case 1
        FMax=Info(1);
        LOR=Info(2);
        fHz=Info(3);
        dratio=Info(4);
        mod_switch=0;
        mod_ratio=0;
    case 2
        FMax=Info(1);
        LOR=Info(2);
        fHz=Info(3);
        dratio=2*Info(5);
        mod_switch=1;
        mod_ratio=Info(4);
    case 3
        FMax=Info(1);
        LOR=Info(2);
        fHz=Info(3);
        dratio=Info(6);
        mod_switch=0;
        mod_ratio=0;
    case 4
        FMax=Info(1);
        LOR=Info(2);
        fHz=Info(3);
        dratio=Info(7);
        mod_switch=1;
        mod_ratio=1;
    otherwise
end

global M C K I wf mo r fi tspan
h=waitbar(0.5,'Calculating Output, Please wait...');
```

```
m1=100;                     %Component 1 mass
m2=500;                     %Component 2 mass
mo=500*(1/fHz^2);           %Unbalance mass at component 1
c1=900000;                  %Damping coefficient 1 Ns/m
c2=2500000;                 %Damping coefficient 2 Ns/m
k1=40000000000;             %Stiffness 1 N/m
k2=80000000000;             %Stiffness 2 N/m
g=9.81;                     %Gravity
imp=100000*fHz;             %Peak pulse force N varies with speed
r=0.5;                      %radius unbalance mass rotates at
dHz=fHz*dratio;             %Defect ratio Hz
T=1/dHz;                    %Defect force pulse period
wf=2*pi*fHz;                %Shaft turning speed rad/s
cf=mo*r*wf^2;               %Centrifugal force
%Matrices for inertia M,damping C, stiffness K and an identity matrix I.
M=[m1+mo,0;0,m2];
C=[c1,-c1;-c1,c1+c2];
K=[k1,-k1;-k1,k1+k2];
I=[1,0;0,1];
%Modified sinusoid for load zone modulation
def_deg=155;                %Defect Load Zone andle 155 degrees
def_rad=def_deg/(180/pi);   %Defect Load Zone andle radians
shift=sin(def_rad);         %Factor to shift sinwave
%Data collection/generation parameters
Wave_L=2.56*LOR;            %Number of points in the timewaveform
Samp_freq=2.56*FMax;        %Frequency of sampling in waveform
Tstep=1/Samp_freq;          %The waveform sample period
Time=LOR/FMax;              %Time/duration of waveform seconds
%Time constraint on ODE solver
tspan=(0:Wave_L-1)*Tstep;
d=(0:T:Time);               % T repetition frequency
%Construct window for load zone modulation
y1=sin(mod_ratio*wf*tspan)-sin(1/mod_ratio*shift);
yscale=1/max(y1);
for n=1:length(tspan)
    if y1(n)>0
    y1(n)=y1(n)*yscale;
    else y1(n)=0;
    end
end
%Construct pulse train for input to ODE solver
if mod_switch==0
    y=pulstran(tspan,d,'rectpuls',0.02*T);
```

```
else
    y=y1.*(pulstran(tspan,d,'rectpuls',0.02*T));
end
fi=imp*y;
%Call ODE solver
[t, x] = ode23('TDOF_SOLVER',tspan, [0 0 0 0]);
%Simulation Results
D=x(:,1);
V=x(:,2);
A=diff(V);
A(max(size(A))+1)=0;
A=A/g;
%fft using 100%*(2/2.56)=78.125% of Nyquist rate
%VELOCITY SPECTRUM
Yv = fft(V,2.56*LOR)/(2.56*LOR);
YV=2*abs(Yv(1:LOR));
f = FMax*linspace(0,1,LOR);
%ACCELERATION SPECTRUM
Ya = fft(A,2.56*LOR)/(2.56*LOR);
YA=2*abs(Ya(1:LOR));   %m/s^2
YA=YA/g;    %G's
%Send information to workspace
assignin ('base','D',D);
assignin ('base','V',V)
assignin ('base','A',A)
assignin ('base','tspan',tspan)
assignin ('base','f',f);
assignin ('base','YV',YV)
assignin ('base','YA',YA)
assignin ('base','LOR',LOR)
assignin ('base','Samp_freq',Samp_freq)
assignin('base','FMax',FMax)
%Plots velocity and sets conditions of Filter Band, Envelope and Play
%Sound as unusable if Velocity is selected
rb1=get(handles.radiobutton1,'Value');
 if rb1==1
     set(handles.radiobutton4,'enable','off');
     set(handles.radiobutton4,'Value',0);
     set(handles.radiobutton5,'enable','off');
     set(handles.radiobutton5,'Value',0);
     set(handles.radiobutton6,'enable','off');
     plot(handles.axes1,tspan,V*1000)
     plot(handles.axes2,f,YV*1000)
```

```
     axes(handles.axes1);
     xlabel('Time (s)')
     ylabel('Velocity (mm/s) ')
     axes(handles.axes2);
     xlabel('Frequency (Hz)')
     ylabel('Peak Velocity (mm/s)')
 else
     %Plots acceleration and sets conditions of Filter Band, Envelope
     %and Play Sound as unusable if Velocity is not selected
     set(handles.radiobutton4,'Value',0);
     set(handles.radiobutton5,'Value',0);
     set(handles.radiobutton6,'enable','on');
     plot(handles.axes1,tspan,A)
     plot(handles.axes2,f,YA)
     axes(handles.axes1);
     xlabel('Time (s)')
     ylabel('Acceleration (Gs) ')
     axes(handles.axes2);
     xlabel('Frequency (Hz)')
     ylabel('Peak Acceleration (Gs)')
 end

set(handles.edit3,'String',fHz);
set(handles.edit4,'String',round((fHz*dratio),2));
set(handles.edit5,'String',dratio);
Fpass1=round(get(handles.slider3,'Value'));
set(handles.edit6,'String',Fpass1);
Fpass2=round(get(handles.slider4,'Value'));
set(handles.edit7,'String',Fpass2);

close(h)    %Close waitbar

% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on Select Machine Component change in popupmenu2.
function popupmenu2_Callback(hObject, eventdata, handles)
%Enable use of Velocity and Acceleration radiobuttons
```

```
set(handles.radiobutton1,'enable','on');
set(handles.radiobutton3,'enable','on');
%Reset Filter Band and Envelope radiobuttons
set(handles.radiobutton4,'Value',0);
set(handles.radiobutton5,'Value',0);
%Select current Machine Component and Defect displayed in GUI
FMax=get(handles.popupmenu2,'Value');
%FMax Maximum frequency specified for spectrum
%LOR Lines of resolution for spectrum
switch FMax
   case 1  %[FMax LOR fHz  FTF  BSF  ORDF IRDF]
        Info=[2000 12800 1.67 0.45 4.74 11.2 13.8];
        axes(handles.axes4)
        imshow('Pulley.jpg');
    case 2
        Info=[10000 25600 7.05 0.441 4.074 9.711 12.289];
        axes(handles.axes4)
        imshow('Crusher.jpg');
    case 3
        Info=[10000 12800 19.62 0.4 2.45 6.47 9.53];
        axes(handles.axes4)
        imshow('2ndShaft.jpg');
    case 4
        Info=[10000 25600 24.83 0.39 2.09 3.09 4.91];
        axes(handles.axes4)
        imshow('MtrNDE.jpg');
    case 5
        Info=[10000 25600 24.83 0.4 2.43 5.21 7.79];
        axes(handles.axes4)
        imshow('MtrDE.jpg');
    otherwise
end
%Select information based on Non Synchronous Defect chosen in
LOR=get(handles.popupmenu1,'Value');
switch LOR
    case 1
        FMax=Info(1);
        LOR=Info(2);
        fHz=Info(3);
        dratio=Info(4);
        mod_switch=0;
        mod_ratio=0;
    case 2
```

```
        FMax=Info(1);
        LOR=Info(2);
        fHz=Info(3);
        dratio=2*Info(5);
        mod_switch=1;
        mod_ratio=Info(4);
    case 3
        FMax=Info(1);
        LOR=Info(2);
        fHz=Info(3);
        dratio=Info(6);
        mod_switch=0;
        mod_ratio=0;
    case 4
        FMax=Info(1);
        LOR=Info(2);
        fHz=Info(3);
        dratio=Info(7);
        mod_switch=1;
        mod_ratio=1;
    otherwise
end

global M C K I wf mo r fi tspan
h=waitbar(0.5,'Calculating Output, Please wait...');

m1=100;                     %Component 1 mass
m2=500;                     %Component 2 mass
mo=500*(1/fHz^2);           %Unbalance mass at component 1
c1=900000;                  %Damping coefficient 1 Ns/m
c2=2500000;                 %Damping coefficient 2 Ns/m
k1=40000000000;             %Stiffness N/m
k2=80000000000;             %Stiffness N/m
g=9.81;                     %Gravity
imp=100000*fHz;             %Peak pulse force N varies with speed
r=0.5;                      %radius unbalance mass rotates at
dHz=fHz*dratio;             %Defect frequency Hz
T=1/dHz;                    %Defect force pulse period
wf=2*pi*fHz;                %Shaft turning speed rad/s
cf=mo*r*wf^2;               %Centrifugal force
%Matrices for inertia M,damping C, stiffness K and an identity matrix I.
M=[m1+mo,0;0,m2];
C=[c1,-c1;-c1,c1+c2];
```

```
K=[k1,-k1;-k1,k1+k2];
I=[1,0;0,1];
%Modified sinusoid for load zone modulation
def_deg=155;                %Defect Load Zone andle 155 degrees
def_rad=def_deg/(180/pi);   %Defect Load Zone andle radians
shift=sin(def_rad);         %Factor to shift sinwave
%Data collection/generation parameters
Wave_L=2.56*LOR;            %Number of points in the timewaveform
Samp_freq=2.56*FMax;        %Frequency of sampling in waveform
Tstep=1/Samp_freq;          %The waveform sample period
Time=LOR/FMax;              %Time/duration of waveform seconds
%Time constraint on ODE solver
tspan=(0:Wave_L-1)*Tstep;
d=(0:T:Time);               % Defect repetition
%Construct window for load zone modulation
y1=sin(mod_ratio*wf*tspan)-sin(1/mod_ratio*shift);
yscale=1/max(y1);
for n=1:length(tspan)
    if y1(n)>0
    y1(n)=y1(n)*yscale;
    else y1(n)=0;
    end
end
%Construct pulse train for input to ODE solver
if mod_switch==0
    y=pulstran(tspan,d,'rectpuls',0.02*T);
else
    y=y1.*(pulstran(tspan,d,'rectpuls',0.02*T));
end
fi=imp*y;
%Call ODE Solver
[t, x] = ode23('TDOF_SOLVER',tspan, [0 0 0 0]);
%Simulation results
D=x(:,1);
V=x(:,2);
A=diff(V);
A(max(size(A))+1)=0;
A=A/g;
%fft using 100%*(2/2.56)=78.125% of Nyquist rate
%VELOCITY SPECTRUM
Yv = fft(V,2.56*LOR)/(2.56*LOR);
YV=2*abs(Yv(1:LOR));
f = FMax*linspace(0,1,LOR);
```

```
%ACCELERATION SPECTRUM
Ya = fft(A,2.56*LOR)/(2.56*LOR);
YA=2*abs(Ya(1:LOR));
YA=YA/g;    %G's
%Send information to workspace
assignin ('base','D',D);
assignin ('base','V',V)
assignin ('base','A',A)
assignin ('base','tspan',tspan)
assignin ('base','f',f);
assignin ('base','YV',YV)
assignin ('base','YA',YA)
assignin ('base','LOR',LOR)
assignin ('base','Samp_freq',Samp_freq)
assignin('base','FMax',FMax)
%Plots velocity if Velocity is selected
rb1=get(handles.radiobutton1,'Value');
 if rb1==1
     plot(handles.axes1,tspan,V*1000)
     plot(handles.axes2,f,YV*1000)
     axes(handles.axes1);
     xlabel('Time (s)')
     ylabel('Velocity (mm/s) ')
     axes(handles.axes2);
     xlabel('Frequency (Hz)')
     ylabel('Peak Velocity (mm/s)')
 else
     %Plots Acceleration if Velocity is not selected
     plot(handles.axes1,tspan,A)
     plot(handles.axes2,f,YA)
     axes(handles.axes1);
     xlabel('Time (s)')
     ylabel('Acceleration (Gs) ')
     axes(handles.axes2);
     xlabel('Frequency (Hz)')
     ylabel('Peak Acceleration (Gs)')
 end

set(handles.edit3,'String',fHz);
set(handles.edit4,'String',round((fHz*dratio),2));
set(handles.edit5,'String',dratio);
Fpass1=round(get(handles.slider3,'Value'));
set(handles.edit6,'String',Fpass1);
```

```
Fpass2=round(get(handles.slider4,'Value'));
set(handles.edit7,'String',Fpass2);

close(h)    %Close waitbar

% --- Executes during object creation, after setting all properties.
function popupmenu2_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Plots current spectrum in GUI on pushbutton Plot Spectrum.
function pushbutton4_Callback(hObject, eventdata, handles)
%Get radiobutton information on what is currently plotted
rb1=get(handles.radiobutton1,'Value');
rb3=get(handles.radiobutton3,'Value');
rb4=get(handles.radiobutton4,'Value');
rb5=get(handles.radiobutton5,'Value');
%Get information from workspace
f=evalin('base','f');
YV=evalin('base','YV');
YA=evalin('base','YA');
LOR=evalin('base','LOR');
%Sort radiobutton information and plot approriate spectrum in a
%separate Matlab figure
if rb1==1
    figure
    plot(f,YV*1000)
    xlabel('Frequency (Hz)')
    ylabel('Peak Velocity (mm/s)')
    xlim([0 2000])
    title('Velocity Spectrum')
elseif rb3==1 && rb4==0 && rb5==0
    figure
    plot(f,YA)
    xlabel('Frequency (Hz) ')
    ylabel('Peak Acceleration (Gs) ')
    title('Accelration Spectrum')
elseif rb3==1 && rb4==1 && rb5==0
    Y_filt=evalin('base','Y_filt');
```

```
    figure
    plot(f,2*abs(Y_filt(1:LOR)))
    xlabel('Frequency (Hz) ')
    ylabel('Peak Acceleration (Gs) ')
    title('Spectrum of Filtered Acceleration Signal')
elseif rb3==1 && rb4==1 && rb5==1
    figure
    envspect=evalin('base','envspect');
    plot(f,envspect,'color',[0.8500    0.3250    0.0980])
    xlabel('Frequency (Hz) ')
    ylabel('Peak Acceleration (Gs) ')
    xlim([0 1000])
    title('Spectrum of Hilbert Tranform Envelope - Demodulated Signal')
end


% --- Plots current waveform in GUI on pushbutton Plot Waveform.
function pushbutton5_Callback(hObject, eventdata, handles)
%Get radiobutton information on what is currently plotted
rb1=get(handles.radiobutton1,'Value');
rb3=get(handles.radiobutton3,'Value');
rb4=get(handles.radiobutton4,'Value');
rb5=get(handles.radiobutton5,'Value');
%Get information from workspace
tspan=evalin('base','tspan');
V=evalin('base','V');
A=evalin('base','A');
%Sort radiobutton information and plot approriate waveform in a
%separate Matlab figure
if rb1==1
    figure
    plot(tspan,V*1000)
    xlabel('Time (s)')
    ylabel('Velocity (mm/s) ')
    title('Velocity Time Waveform')
elseif rb3==1 && rb4==0 && rb5==0
    figure
    plot(tspan,A)
    xlabel('Time(s)')
    ylabel('Acceleration (Gs)')
    title('Unfiltered Acceleration Signal')
elseif rb3==1 && rb4==1 && rb5==0
    A_filt=evalin('base','A_filt');
```

```
    figure
    plot(tspan,A_filt)
    xlabel('Time(s)')
    ylabel('Acceleration (Gs)')
    title('Filtered Acceleration Signal')
elseif rb3==1 && rb4==1 && rb5==1
    A_filt=evalin('base','A_filt');
    envelope=evalin('base','envelope');
    figure
    plot(tspan,A_filt,tspan,envelope)
    xlabel('Time(s)')
    ylabel('Acceleration (Gs)')
    title('Hilbert Tranform Envelope on Filtered Acceleration Signal')
end

% --- Executes during object creation, after setting all properties.
function axes4_CreateFcn(hObject, eventdata, handles)

function edit3_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit4_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit5_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Plots Current solution as Velocity in GUI on execution of
%Units, Velocity radiobutton.
function radiobutton1_Callback(hObject, eventdata, handles)
%Get information from workspace on current solution
tspan=evalin('base','tspan');
V=evalin('base','V');
f = evalin('base','f');
YV=evalin('base','YV');
%Plots and sets other conditions
rb1=get(handles.radiobutton1,'Value');
 if rb1==1
     set(handles.radiobutton4,'enable','off');
     set(handles.radiobutton4,'Value',0);
     set(handles.radiobutton5,'enable','off');
     set(handles.radiobutton5,'Value',0);
     set(handles.radiobutton6,'enable','off');
     plot(handles.axes1,tspan,V*1000)
     plot(handles.axes2,f,YV*1000)
     axes(handles.axes1);
     xlabel('Time (s)')
     ylabel('Velocity (mm/s) ')
     axes(handles.axes2);
     xlabel('Frequency (Hz)')
     ylabel('Peak Velocity (mm/s)')
 else
 end

% --- % --- Plots Current solution as Acceleration in GUI on execution
%of Units, Acceleration radiobutton.
function radiobutton3_Callback(hObject, eventdata, handles)
%Get information from workspace on current solution
tspan=evalin('base','tspan');
A=evalin('base','A');
f = evalin('base','f');
YA=evalin('base','YA');
%Plots and sets other conditions
rb3=get(handles.radiobutton3,'Value');
 if rb3==1
```

```
     set(handles.radiobutton4,'enable','on');
     set(handles.radiobutton6,'enable','on');
     plot(handles.axes1,tspan,A)
     plot(handles.axes2,f,YA)
     axes(handles.axes1);
     xlabel('Time (s)')
     ylabel('Acceleration (Gs) ')
     axes(handles.axes2);
     xlabel('Frequency (Hz)')
     ylabel('Peak Acceleration (Gs)')
 else
 end

% --- Applies selected band filter to current waveform on execution of
% Filter Band press of radiobutton.
function radiobutton4_Callback(hObject, eventdata, handles)
%Get lower and upper values for band filter
Fpass1=round(get(handles.slider3,'Value'));
Fpass2=round(get(handles.slider4,'Value'));
%Get information on currently displayed solution
A=evalin('base','A');
FMax=evalin('base','FMax');
Fs=evalin('base','Samp_freq');
LOR=evalin('base','LOR');
tspan=evalin('base','tspan');
f=evalin('base','f');
YA=evalin('base','YA');
%Button operations
rb4=get(handles.radiobutton4,'Value');
if rb4==1;
    %Turn some functions off so they cannot be used during this process
        h=waitbar(0.5,'Calculating Output, Please wait...');
        set(handles.radiobutton1,'enable','off');
        set(handles.radiobutton3,'enable','off');
        set(handles.radiobutton5,'enable','on');
        set(handles.radiobutton6,'enable','off');
%Code for filter generated by Matlab - Fstops modified
Fstop1 = Fpass1-0.02*Fpass2;            % First Stopband Frequency
Fstop2 = Fpass2+0.02*Fpass2;            % Second Stopband Frequency
Dstop1 = 0.0001;          % First Stopband Attenuation
Dpass  = 0.057501127785;  % Passband Ripple
Dstop2 = 0.0001;          % Second Stopband Attenuation
dens   = 20;              % Density Factor
```

```
% Calculate the order from the parameters using FIRPMORD.
[N, Fo, Ao, W] = firpmord([Fstop1 Fpass1 Fpass2 Fstop2]/(Fs/2), [0 1 ...
                          0], [Dstop1 Dpass Dstop2]);
% Calculate the coefficients using the FIRPM function.
b  = firpm(N, Fo, Ao, W, {dens});
Hd = dfilt.dffir(b);
%Compensate for filter delay
D = round(mean(grpdelay(Hd)));      % filter delay in sample
A_filt = filter(Hd,[A',zeros(1,D)]);% Append D zeros to the input data
A_filt = A_filt(D+1:end);           % Shift data to compensate for delay
assignin ('base','Hd',Hd)
%Filtered waveform to workspace
assignin ('base','A_filt',A_filt)
%fft of filtered waveform and plotted
Y_filt = fft(A_filt,2.56*LOR)/(2.56*LOR);
f = FMax*linspace(0,1,LOR);
plot(handles.axes2,f,2*abs(Y_filt(1:LOR)))
%Hilbert transform of filtered waveform as envelope
envelope=abs(hilbert(A_filt));
%fft of Hilbert transform envelope
envfft = fft(envelope,2.56*LOR)/(2.56*LOR);
envspect=2*abs(envfft(1:LOR));
%Plot filtered wavefrom
plot(handles.axes1,tspan,A_filt)

axes(handles.axes1);
xlabel('Time (s)')
ylabel('Acceleration (Gs) ')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Peak Acceleration (Gs)')
%Information to workspace
assignin('base','Y_filt',Y_filt)
assignin('base','envelope',envelope)
assignin('base','envspect',envspect)

close(h)    %close waitbar

else
    %Reset button functions and plot unfiltered acceleration
    set(handles.radiobutton1,'enable','on');
    set(handles.radiobutton3,'enable','on');
    set(handles.radiobutton5,'enable','off','value',0);
```

```
    set(handles.radiobutton6,'enable','on');
    plot(handles.axes1,tspan,A)
    plot(handles.axes2,f,YA)
    axes(handles.axes1);
    xlabel('Time (s)')
    ylabel('Acceleration (Gs) ')
    axes(handles.axes2);
    xlabel('Frequency (Hz)')
    ylabel('Peak Acceleration (Gs)')
end

function edit6_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit7_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit7_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Display lower Filter Band frequency on slider movement.
function slider3_Callback(hObject, eventdata, handles)
Fpass1=round(get(handles.slider3,'Value'));
set(handles.edit6,'String',Fpass1);
Fpass2=round(get(handles.slider4,'Value'));
set(handles.edit7,'String',Fpass2);

% --- Executes during object creation, after setting all properties.
function slider3_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Display upper Filter Band frequency on slider movement.
function slider4_Callback(hObject, eventdata, handles)
Fpass1=round(get(handles.slider3,'Value'));
set(handles.edit6,'String',Fpass1);
Fpass2=round(get(handles.slider4,'Value'));
set(handles.edit7,'String',Fpass2);

% --- Executes during object creation, after setting all properties.
function slider4_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Plots in the GUI the envelope on filtered waveform and the frequency
%spectrum of the Hilbert transform envelope on press of Envelope
function radiobutton5_Callback(hObject, eventdata, handles)
%Filter Band and Envelope radiobutton values
rb4=get(handles.radiobutton4,'Value');
rb5=get(handles.radiobutton5,'Value');
%Required information form workspace
Y_filt=evalin('base','Y_filt');
A_filt=evalin('base','A_filt');
envelope=evalin('base','envelope');
envspect=evalin('base','envspect');
f=evalin('base','f');
tspan=evalin('base','tspan');
LOR=evalin('base','LOR');
FMax=evalin('base','FMax');
YA=evalin('base','YA');
%Performs the following if both Filter Band and Envelope are slected
if rb4==1 && rb5==1;
%Plots the envelope on the filtered waveform in the GUI
plot(handles.axes1,tspan,A_filt,tspan,envelope,'LineWidth',0.8)
%Plots the spectrum of the Hilbert transform envelope in the GUI
plot(handles.axes2,f,envspect,'color',[0.8500    0.3250    0.0980])
set(handles.axes2,'xlim',[0 FMax/15])

axes(handles.axes1);
xlabel('Time (s)')
```

```
ylabel('Acceleration (Gs) ')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Peak Acceleration (Gs)')

else
    %Returns GUI plots to filtered waveform and spectrum
    plot(handles.axes1,tspan,A_filt)
    plot(handles.axes2,f,2*abs(Y_filt(1:LOR)))
    axes(handles.axes1);

    xlabel('Time (s)')
    ylabel('Acceleration (Gs) ')
    axes(handles.axes2);
    xlabel('Frequency (Hz)')
    ylabel('Peak Acceleration (Gs)')
end

% --- Plays the sound associated with the current ODE solution
% on button press of Play Sound
function radiobutton6_Callback(hObject, eventdata, handles)
%Get information on user selections in GUI
rb3=get(handles.radiobutton3,'Value');
rb4=get(handles.radiobutton4,'Value');
rb5=get(handles.radiobutton5,'Value');
%Information required to play the sound from workspace
A=evalin('base','A');
Samp_freq=evalin('base','Samp_freq');
tspan=evalin('base','tspan');
%Only play sound if acceleration is slected
% and if filter and envelopeare not selected
if rb3==1 && rb4==0 && rb5==0
     set(handles.radiobutton6,'enable','off')
     sound(A*1000000,Samp_freq) %Amplify sound
     pause(max(tspan))
     set(handles.radiobutton6,'Value',0)
     set(handles.radiobutton6,'enable','on')
else
end

%ENDE Level_2_4
```

C.18 Level_2_5

```
function varargout = Level_2_5(varargin)
% LEVEL_2_5 M-file for Level_2_5.fig
% Last Modified by GUIDE v2.5 21-Oct-2015 09:06:06
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Level_2_5_OpeningFcn, ...
                   'gui_OutputFcn',  @Level_2_5_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before Level_2_5 is made visible.
function Level_2_5_OpeningFcn(hObject, eventdata, handles, varargin)
% Choose default command line output for Level_2_5
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
%Clear workspace
evalin('base','clear')
%Displays text associated with GUI concepts
fid=fopen('Level_2_5.txt');
C = textscan(fid,'%s','Delimiter','\n');
C=C{1};
fclose(fid);
set(handles.edit1,'String',C);
%Data generation/collection properties to workspace
FMax=200;
```

```
assignin('base','FMax',FMax)
LOR=200;
assignin('base','LOR',LOR)
WavePoints=2.56*LOR;
Sampfreq=2.56*FMax;
TimeStep=1/Sampfreq;
t=(0:WavePoints-1)*TimeStep;
assignin('base','t',t)
%Generate random waveform frequencies and amplitudes
f1=195*rand;
f2=195*rand;
f3=195*rand;
f4=195*rand;
f5=195*rand;
f6=195*rand;
f7=f1+2;
f8=f2+2;
f9=f8+2;
Amplitude1=5*rand;
Amplitude2=5*rand;
Amplitude3=5*rand;
Amplitude4=5*rand;
Amplitude5=5*rand;
Amplitude6=5*rand;
Amplitude7=5*rand;
Amplitude8=5*rand;
Amplitude9=5*rand;
w1=2*pi*f1;
w2=2*pi*f2;
w3=2*pi*f3;
w4=2*pi*f4;
w5=2*pi*f5;
w6=2*pi*f6;
w7=2*pi*f7;
w8=2*pi*f8;
w9=2*pi*f9;
wave1=Amplitude1*sin(w1*t);
wave2=Amplitude2*sin(w2*t);
wave3=Amplitude3*sin(w3*t);
wave4=Amplitude4*sin(w4*t);
wave5=Amplitude5*sin(w5*t);
wave6=Amplitude6*sin(w6*t);
wave7=Amplitude7*sin(w7*t);
```

```
wave8=Amplitude8*sin(w8*t);
wave9=Amplitude9*sin(w9*t);
wave=wave1+wave2+wave3+wave4+wave5+wave6+wave7+wave8+wave9;
%Generate windows and export to workspace
wave=wave';
one=ones(1,length(wave));
rect=one'.*rectwin(length(wave));
hann=one'.*hanning(length(wave));
hamm=one'.*hamming(length(wave));
blackharr=one'.*blackmanharris(length(wave));
assignin('base','rect',rect)
assignin('base','hann',hann)
assignin('base','hamm',hamm)
assignin('base','blackharr',blackharr)
%Plot waveform
plot(handles.axes1,t,wave)  %,t,wave1,t,wave2)
set(handles.axes1,'ylim',[1.1*min(wave) 1.1*max(wave)]);
%fft using 100%*(2/2.56)=78.125% of Nyquist rate and plot spectrum
Y = fft(wave,2.56*LOR)/(2.56*LOR);
f = FMax*linspace(0,1,LOR);
plot(handles.axes2,f,2*abs(Y(1:LOR)))
set(handles.axes2,'ylim',[0 7])
%Plot axis labels
axes(handles.axes1);
xlabel('Time (s)')
ylabel('Amplitude')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Amplitude')
%Information to workspace
assignin ('base','wave',wave)
%assignin ('base','wave1',wave1)
%assignin ('base','wave2',wave2)
assignin ('base','t',t)

% --- Outputs from this function are returned to the command line.
function varargout = Level_2_5_OutputFcn(hObject, eventdata, handles)
% Get default command line output from handles structure
varargout{1} = handles.output;

function edit1_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
```

```
function edit1_CreateFcn(hObject, eventdata, handles)
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)

% --- Executes on button press in Previous.
function Previous_Callback(hObject, eventdata, handles)
h=waitbar(0.5,'Opening Previous Level, Please wait...');
run('Level_2_4')
close('Level_2_5')
close(h)

% --- Executes on button press in Next.
function Next_Callback(hObject, eventdata, handles)
%User requires waitbar while Level_2_6 opens
h=waitbar(0.5,'Opening Next Level, Please wait...');
run('Level_2_6')
close('Level_2_5')
close(h)

% --- Executes during object creation, after setting all properties.
function Amplitude_CreateFcn(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function axes2_CreateFcn(hObject, eventdata, handles)

% --- Generates ne waveform at press of Change Waveform button .
function pushbutton3_Callback(hObject, eventdata, handles)
%Returns to rectangular/no window
set(handles.radiobutton1,'Value',1);
set(handles.radiobutton2,'Value',0);
set(handles.radiobutton3,'Value',0);
set(handles.radiobutton4,'Value',0);
%Data generation parameter for binary size wave file
FMax=200;
LOR=200;
WavePoints=2.56*LOR;
Sampfreq=2.56*FMax;
```

```
TimeStep=1/Sampfreq;
t=(0:WavePoints-1)*TimeStep;
%Randomly generate frequencies, amplitudes, assemble wave and plot
f1=195*rand;
f2=195*rand;
f3=195*rand;
f4=195*rand;
f5=195*rand;
f6=195*rand;
f7=f1+2;
f8=f2+2;
f9=f8+2;
Amplitude1=5*rand;
Amplitude2=5*rand;
Amplitude3=5*rand;
Amplitude4=5*rand;
Amplitude5=5*rand;
Amplitude6=5*rand;
Amplitude7=5*rand;
Amplitude8=5*rand;
Amplitude9=5*rand;
w1=2*pi*f1;
w2=2*pi*f2;
w3=2*pi*f3;
w4=2*pi*f4;
w5=2*pi*f5;
w6=2*pi*f6;
w7=2*pi*f7;
w8=2*pi*f8;
w9=2*pi*f9;
wave1=Amplitude1*sin(w1*t);
wave2=Amplitude2*sin(w2*t);
wave3=Amplitude3*sin(w3*t);
wave4=Amplitude4*sin(w4*t);
wave5=Amplitude5*sin(w5*t);
wave6=Amplitude6*sin(w6*t);
wave7=Amplitude7*sin(w7*t);
wave8=Amplitude8*sin(w8*t);
wave9=Amplitude9*sin(w9*t);
wave=wave1+wave2+wave3+wave4+wave5+wave6+wave7+wave8+wave9;
plot(handles.axes1,t,wave)
set(handles.axes1,'ylim',[1.1*min(wave) 1.1*max(wave)]);
%fft on binary size wave using 78.125% of Nyquist rate
```

```
Y = fft(wave,2.56*LOR)/(2.56*LOR);
f = FMax*linspace(0,1,LOR);
plot(handles.axes2,f,2*abs(Y(1:LOR)))
set(handles.axes2,'ylim',[0 7])
%Plot axis labels
axes(handles.axes1);
xlabel('Time (s)')
ylabel('Amplitude')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Amplitude')
%Transpose new wave to work with windoing and send to workspace
wave=wave';
assignin ('base','wave',wave)

% --- Applies rectangular window to waveform on GUI radiobutton press
function radiobutton1_Callback(hObject, eventdata, handles)
%Retrieve information from workspace
rect=evalin('base','rect');
wave=evalin('base','wave');
t=evalin('base','t');
LOR=evalin('base','LOR');
FMax=evalin('base','FMax');
%Modify waveform with time window and plot
wave=wave.*rect;
plot(handles.axes1,t,wave)
set(handles.axes1,'ylim',[1.1*min(wave) 1.1*max(wave)]);
%Conduct fft on windowed waveform
Y = fft(wave,2.56*LOR)/(2.56*LOR);
f = FMax*linspace(0,1,LOR);
plot(handles.axes2,f,2*abs(Y(1:LOR)))
set(handles.axes2,'ylim',[0 7])
axes(handles.axes1);
xlabel('Time (s)')
ylabel('Amplitude')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Amplitude')

% --- Applies Hanning window to waveform on GUI radiobutton press
function radiobutton2_Callback(hObject, eventdata, handles)
%Retrieve information from workspace
hann=evalin('base','hann');
```

```
wave=evalin('base','wave');
t=evalin('base','t');
LOR=evalin('base','LOR');
FMax=evalin('base','FMax');
%Modify waveform with Hanning time window and plot
wave=wave.*hann;
plot(handles.axes1,t,wave)
set(handles.axes1,'ylim',[1.1*min(wave) 1.1*max(wave)]);
%Amplitude correction factor of 2 for the Hanning window
cf=2;
%fft of Hanning modified waveform
Y = fft(wave,2.56*LOR)/(2.56*LOR);
f = FMax*linspace(0,1,LOR);
plot(handles.axes2,f,cf*2*abs(Y(1:LOR)))
set(handles.axes2,'ylim',[0 7])
axes(handles.axes1);
xlabel('Time (s)')
ylabel('Amplitude')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Amplitude')

% --- Applies Hamming window to waveform on GUI radiobutton press
function radiobutton3_Callback(hObject, eventdata, handles)
%Retrieve information from workspace
hamm=evalin('base','hamm');
wave=evalin('base','wave');
t=evalin('base','t');
LOR=evalin('base','LOR');
FMax=evalin('base','FMax');
%Modify waveform with Hamming time window and plot
wave=wave.*hamm;
plot(handles.axes1,t,wave)
set(handles.axes1,'ylim',[1.1*min(wave) 1.1*max(wave)]);
%Amplitude correction factor of 1.85 for the Hamming window
cf=1.85;
%fft of Hanning modified waveform
Y = fft(wave,2.56*LOR)/(2.56*LOR);
f = FMax*linspace(0,1,LOR);
plot(handles.axes2,f,cf*2*abs(Y(1:LOR)))
set(handles.axes2,'ylim',[0 7])
axes(handles.axes1);
xlabel('Time (s)')
```

```
ylabel('Amplitude')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Amplitude')

% --- Applies Blackman Harris window to waveform on GUI radiobutton press
function radiobutton4_Callback(hObject, eventdata, handles)
%Retrieve information from workspace
blackharr=evalin('base','blackharr');
wave=evalin('base','wave');
t=evalin('base','t');
LOR=evalin('base','LOR');
FMax=evalin('base','FMax');
%Modify waveform with Blackman Harris time window and plot
wave=wave.*blackharr;
plot(handles.axes1,t,wave)
set(handles.axes1,'ylim',[1.1*min(wave) 1.1*max(wave)]);
%Amplitude correction factor (cf) for the Hamming window
cf=1/0.42323;
%fft of Blackman Harris modified waveform
Y = fft(wave,2.56*LOR)/(2.56*LOR);
f = FMax*linspace(0,1,LOR);
plot(handles.axes2,f,cf*2*abs(Y(1:LOR)))
set(handles.axes2,'ylim',[0 7])
axes(handles.axes1);
xlabel('Time (s)')
ylabel('Amplitude')
axes(handles.axes2);
xlabel('Frequency (Hz)')
ylabel('Amplitude')

%END Level_2_5
```

C.19 Level_2_6

```
 function varargout = Level_2_6(varargin)
% LEVEL_2_6 M-file for Level_2_6.fig
% Last Modified by GUIDE v2.5 19-Oct-2015 11:07:23
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Level_2_6_OpeningFcn, ...
                   'gui_OutputFcn',  @Level_2_6_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before Level_2_6 is made visible.
function Level_2_6_OpeningFcn(hObject, eventdata, handles, varargin)
% Choose default command line output for Level_2_6
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
%Clear workspace
evalin('base','clear')
global Mt M mo c k T r wf fi tspan
%Display text associated with the GUI concepts
fid=fopen('Level_2_6.txt');
C = textscan(fid,'%s','Delimiter','\n');
C=C{1};
fclose(fid);
set(handles.edit1,'String',C);
%Parameters for generating data
```

```
FMax=2000;
LOR=6400;
fHz=get(handles.slider1,'Value');    %Shaft speed
c=get(handles.slider2,'Value');      %Damping factor
M=get(handles.slider3,'Value');      %Machine mass
k=get(handles.slider4,'Value');      %System stiffness
mo=get(handles.slider5,'Value');     %Unbalance mass
%Display Values in GUI
set(handles.Speed,'String',fHz)
set(handles.Damping,'String',c)
set(handles.Mass,'String',M)
set(handles.Stiff,'String',k)
set(handles.Unbal,'String',mo)
dratio=1;
Mt=M+mo;                             %System mass
wf=2*pi*fHz;                         %Shaft speed rad/s
r=0.5;                               %Radius unbalance mass rotates at
dHz=fHz*dratio;
T=1/dHz;                             %Defect force pulse period
wn=sqrt(k/M+mo);                     %SDOF natural frequency
wr=wf/wn;                            %Frequency Ratio
c_c=2*(M+mo)*wn;                     %Critical Damping Value
zeta=c/c_c;                          %Damping Ratio
%Display values in GUI
set(handles.wn,'String',round(wn/(2*pi)))
set(handles.wr,'String',round(wr,2))
set(handles.CritDamp,'String',c_c)
set(handles.DampRat,'String',round(zeta,2))
Wave_L=2.56*LOR;            %Number of points in the timewaveform
Samp_freq=2.56*FMax;        %Frequency of sampling in waveform
Tstep=1/Samp_freq;          %The waveform sample period
dt=1;                       %Time shift (s) for reaching steady state
dt_n=round(dt/Tstep);
%Time constraint for ODE solver
tspan=(0:(Wave_L-1)+dt_n)*Tstep;
%Centrifugal force
fc=mo*r*wf^2*sin(wf*tspan);
fc=fc(dt_n+1:end);
assignin('base','fc',fc)
%There is no pulse force fi going to the SDOF Solver in this GUI
fi=zeros(1,length(tspan));
%Call ODE solver
[t, x] = ode23('SDOF_SOLVER',tspan, [0 0]);
```

```
%Simulation results and plotting, 1000 converts m to mm
D=x(:,1);
D=D(dt_n+1:end);
V=x(:,2);
V=V(dt_n+1:end);
A=diff(V);
A(max(size(A))+1)=0;
t=t(1:end-dt_n);
plot(handles.axes1,t,D*1000,t,fc*(max(D)*1000/max(fc)));
plot(handles.axes2,t,V*1000,t,fc*(max(V)*1000/max(fc)));
axes(handles.axes1);
lim=5*1/fHz;
xlim([0 lim])
xlabel('Time (s)')
ylabel('Displacement (mm) ')
legend('Displacement Response','Phase of Unbalance Mass')
axes(handles.axes2);
xlim([0 lim])
xlabel('Frequency (Hz)')
ylabel('Velocity (mm/s)')
legend('Velocity Response','Phase of Unbalance Mass')
%Information on current in GUI to workspace for external figure plotting
assignin ('base','D',D);
assignin ('base','V',V)
assignin ('base','A',A)
assignin('base','t',t)

% --- Outputs from this function are returned to the command line.
function varargout = Level_2_6_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;

function edit1_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
```

```
% --- Executes on button press in Previous.
function Previous_Callback(hObject, eventdata, handles)

run('Level_2_5')
close('Level_2_6')

% --- Executes on button press in Next.
function Next_Callback(hObject, eventdata, handles)

run('Open')
close('Level_2_6')

% --- Executes during object creation, after setting all properties.
function Amplitude_CreateFcn(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function axes2_CreateFcn(hObject, eventdata, handles)

function wn_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function wn_CreateFcn(hObject, eventdata, handles)
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Damping_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function Damping_CreateFcn(hObject, eventdata, handles)
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Plots displacement and phase for top pushbutton Plot Waveform
function pushbutton4_Callback(hObject, eventdata, handles)
%Plots using information on current from workspace
D=evalin('base','D');
```

```
t=evalin('base','t');
fc=evalin('base','fc');
figure
plot(t,D*1000,t,fc*(max(D)*1000/max(fc)));
xlabel('Time (s)')
ylabel('Amplitude (mm) ')
legend('Displacement Response','Phase of Unbalance Mass')

% --- Plots velocity and phase for top pushbutton Plot Waveform
function pushbutton5_Callback(hObject, eventdata, handles)
%Plots using information on current from workspace
V=evalin('base','V');
t=evalin('base','t');
fc=evalin('base','fc');
figure
plot(t,V*1000,t,fc*(max(V)*1000/max(fc)));
xlabel('Time (s)')
ylabel('Velocity (mm/s) ')
legend('Velocity Response','Phase of Unbalance Mass')

% Calculates system response to change of Machine Speed slider movement.
function slider1_Callback(hObject, eventdata, handles)

global Mt M mo c k T r wf fi tspan

FMax=2000;
LOR=6400;
fHz=round(get(handles.slider1,'Value'));
c=round(get(handles.slider2,'Value'),-1);
M=round(get(handles.slider3,'Value'));
k=round(get(handles.slider4,'Value'),-5);
mo=round(get(handles.slider5,'Value'),1);
set(handles.Speed,'String',fHz)
set(handles.Damping,'String',c)
set(handles.Mass,'String',M)
set(handles.Stiff,'String',k)
set(handles.Unbal,'String',mo)
dratio=1;
Mt=M+mo;                    %System mass
wf=2*pi*fHz;                %Shaft turning speed rad/s
r=0.5;                      %radius unbalance mass rotates at
dHz=fHz*dratio;
T=1/dHz;                    %Defect force pulse period
```

```
wn=sqrt(k/M+mo);            %Single degree of freedom natural frequency
wr=wf/wn;                   %Frequency Ratio
c_c=2*(M+mo)*wn;            %Critical Damping Value
zeta=c/c_c;                 %Damping Ratio
%Display values in GUI
set(handles.wn,'String',round(wn/(2*pi)))
set(handles.wr,'String',round(wr,2))
set(handles.CritDamp,'String',c_c)
set(handles.DampRat,'String',round(zeta,2))
Wave_L=2.56*LOR;            %Number of points in the timewaveform
Samp_freq=2.56*FMax;        %Frequency of sampling in waveform
Tstep=1/Samp_freq;          %The waveform sample period
dt=1;                       %Time shift of wave for reaching steady state
dt_n=round(dt/Tstep);
%Time constraint on ODE solver
tspan=(0:(Wave_L-1)+dt_n)*Tstep;
%Centrifugal force
fc=mo*r*wf^2*sin(wf*tspan);
fc=fc(dt_n+1:end);
assignin('base','fc',fc)
%There is no pulse force fi going to the SDOF Solver in this GUI
fi=zeros(1,length(tspan));
%Call ODE solver
[t, x] = ode23('SDOF_SOLVER',tspan, [0 0]);
%Results and plotting
D=x(:,1);
D=D(dt_n+1:end);
V=x(:,2);
V=V(dt_n+1:end);
A=diff(V);
A(max(size(A))+1)=0;
t=t(1:end-dt_n);
plot(handles.axes1,t,D*1000,t,fc*(max(D)*1000/max(fc)));
plot(handles.axes2,t,V*1000,t,fc*(max(V)*1000/max(fc)));
axes(handles.axes1);
lim=5*1/fHz;
xlim([0 lim])
xlabel('Time (s)')
ylabel('Displacement (mm) ')
legend('Displacement Response','Phase of Unbalance Mass')
axes(handles.axes2);
xlim([0 lim])
xlabel('Frequency (Hz)')
```

```
ylabel('Velocity (mm/s)')
legend('Velocity Response','Phase of Unbalance Mass')
%Current information to workspace
assignin ('base','D',D);
assignin ('base','V',V)
assignin ('base','A',A)
assignin('base','t',t)

% --- Executes during object creation, after setting all properties.
function slider1_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% Calculates system response to change of System Damping slider movement.
function slider2_Callback(hObject, eventdata, handles)

global Mt M mo c k T r wf fi tspan

FMax=2000;
LOR=6400;
fHz=round(get(handles.slider1,'Value'));
c=round(get(handles.slider2,'Value'),-1);
M=round(get(handles.slider3,'Value'));
k=round(get(handles.slider4,'Value'),-5);
mo=round(get(handles.slider5,'Value'),1);
set(handles.Speed,'String',fHz)
set(handles.Damping,'String',c)
set(handles.Mass,'String',M)
set(handles.Stiff,'String',k)
set(handles.Unbal,'String',mo)
dratio=1;
Mt=M+mo;                    %System mass
wf=2*pi*fHz;                %Shaft turning speed rad/s
r=0.5;                      %radius unbalance mass rotates at
dHz=fHz*dratio;
T=1/dHz;                    %Defect force pulse period
wn=sqrt(k/M+mo);            %Single degree of freedom natural frequency
wr=wf/wn;                   %Frequency Ratio
c_c=2*(M+mo)*wn;            %Critical Damping Value
zeta=c/c_c;                 %Damping Ratio
set(handles.wn,'String',round(wn/(2*pi)))
```

```
set(handles.wr,'String',round(wr,2))
set(handles.CritDamp,'String',c_c)
set(handles.DampRat,'String',round(zeta,2))
Wave_L=2.56*LOR;            %Number of points in the timewaveform
Samp_freq=2.56*FMax;        %Frequency of sampling in waveform
Tstep=1/Samp_freq;          %The waveform sample period
dt=1;                       %Time shift of wave for reaching steady state
dt_n=round(dt/Tstep);
%Time constraint on ODE solver
tspan=(0:(Wave_L-1)+dt_n)*Tstep;
fc=mo*r*wf^2*sin(wf*tspan); %Centrifugal force
fc=fc(dt_n+1:end);
assignin('base','fc',fc)
%There is no pulse force fi going to the SDOF Solver in this GUI
fi=zeros(1,length(tspan));
%Call ODE solver
[t, x] = ode23('SDOF_SOLVER',tspan, [0 0]);
%Results and plotting
D=x(:,1);
D=D(dt_n+1:end);
V=x(:,2);
V=V(dt_n+1:end);
A=diff(V);
A(max(size(A))+1)=0;
t=t(1:end-dt_n);
plot(handles.axes1,t,D*1000,t,fc*(max(D)*1000/max(fc)));
plot(handles.axes2,t,V*1000,t,fc*(max(V)*1000/max(fc)));
axes(handles.axes1);
lim=5*1/fHz;
xlim([0 lim])
xlabel('Time (s)')
ylabel('Displacement (mm) ')
legend('Displacement Response','Phase of Unbalance Mass')
axes(handles.axes2);
xlim([0 lim])
xlabel('Frequency (Hz)')
ylabel('Velocity (mm/s)')
legend('Velocity Response','Phase of Unbalance Mass')
%Curremt information to workspace
assignin ('base','D',D);
assignin ('base','V',V)
assignin ('base','A',A)
assignin('base','t',t)
```

```
% --- Executes during object creation, after setting all properties.
function slider2_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% Calculates system response to change of System Mass slider movement.
function slider3_Callback(hObject, eventdata, handles)

global Mt M mo c k T r wf fi tspan

FMax=2000;
LOR=6400;
fHz=round(get(handles.slider1,'Value'));
c=round(get(handles.slider2,'Value'),-1);
M=round(get(handles.slider3,'Value'));
k=round(get(handles.slider4,'Value'),-5);
mo=round(get(handles.slider5,'Value'),1);
set(handles.Speed,'String',fHz)
set(handles.Damping,'String',c)
set(handles.Mass,'String',M)
set(handles.Stiff,'String',k)
set(handles.Unbal,'String',mo)
dratio=1;
Mt=M+mo;                    %System mass
wf=2*pi*fHz;                %Shaft turning speed rad/s
r=0.5;                      %radius unbalance mass rotates at
dHz=fHz*dratio;
T=1/dHz;                    %Defect force pulse period
wn=sqrt(k/M+mo);            %Single degree of freedom natural frequency
wr=wf/wn;                   %Frequency Ratio
c_c=2*(M+mo)*wn;            %Critical Damping Value
zeta=c/c_c;                 %Damping Ratio
set(handles.wn,'String',round(wn/(2*pi)))
set(handles.wr,'String',round(wr,2))
set(handles.CritDamp,'String',c_c)
set(handles.DampRat,'String',round(zeta,2))
Wave_L=2.56*LOR;            %Number of points in the timewaveform
Samp_freq=2.56*FMax;        %Frequency of sampling in waveform
Tstep=1/Samp_freq;          %The waveform sample period
dt=1;                       %Time shift of wave for reaching steady state
```

```
dt_n=round(dt/Tstep);
%Time step constraint on ODE solver
tspan=(0:(Wave_L-1)+dt_n)*Tstep;
fc=mo*r*wf^2*sin(wf*tspan);
fc=fc(dt_n+1:end);
assignin('base','fc',fc)
%There is no pulse force fi going to the SDOF Solver in this GUI
fi=zeros(1,length(tspan));
%Call ODE solver
[t, x] = ode23('SDOF_SOLVER',tspan, [0 0]);
%Results and plotting
D=x(:,1);
D=D(dt_n+1:end);
V=x(:,2);
V=V(dt_n+1:end);
A=diff(V);
A(max(size(A))+1)=0;
t=t(1:end-dt_n);
plot(handles.axes1,t,D*1000,t,fc*(max(D)*1000/max(fc)));
plot(handles.axes2,t,V*1000,t,fc*(max(V)*1000/max(fc)));
axes(handles.axes1);
lim=5*1/fHz;
xlim([0 lim])
xlabel('Time (s)')
ylabel('Displacement (mm) ')
legend('Displacement Response','Phase of Unbalance Mass')
axes(handles.axes2);
xlim([0 lim])
xlabel('Frequency (Hz)')
ylabel('Velocity (mm/s)')
legend('Velocity Response','Phase of Unbalance Mass')
%current information to the workspace
assignin ('base','D',D);
assignin ('base','V',V)
assignin ('base','A',A)
assignin('base','t',t)

% --- Executes during object creation, after setting all properties.
function slider3_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
```

```
% Calculates system response to change of Stiffness slider movement.
function slider4_Callback(hObject, eventdata, handles)

global Mt M mo c k T r wf fi tspan

FMax=2000;
LOR=6400;
fHz=round(get(handles.slider1,'Value'));
c=round(get(handles.slider2,'Value'),-1);
M=round(get(handles.slider3,'Value'));
k=round(get(handles.slider4,'Value'),-5);
mo=round(get(handles.slider5,'Value'),1);
set(handles.Speed,'String',fHz)
set(handles.Damping,'String',c)
set(handles.Mass,'String',M)
set(handles.Stiff,'String',k)
set(handles.Unbal,'String',mo)
dratio=1;
Mt=M+mo;                    %System mass
wf=2*pi*fHz;                %Shaft turning speed rad/s
r=0.5;                      %radius unbalance mass rotates at
dHz=fHz*dratio;
T=1/dHz;                    %Defect force pulse period
wn=sqrt(k/M+mo);            %Single degree of freedom natural frequency
wr=wf/wn;                   %Frequency Ratio
c_c=2*(M+mo)*wn;            %Critical Damping Value
zeta=c/c_c;                 %Damping Ratio
set(handles.wn,'String',round(wn/(2*pi)))
set(handles.wr,'String',round(wr,2))
set(handles.CritDamp,'String',c_c)
set(handles.DampRat,'String',round(zeta,2))
Wave_L=2.56*LOR;            %Number of points in the timewaveform
Samp_freq=2.56*FMax;        %Frequency of sampling in waveform
Tstep=1/Samp_freq;          %The waveform sample period
dt=1;                       %Time shift of wave for reaching steady state
dt_n=round(dt/Tstep);
%Time constraint on time waveform
tspan=(0:(Wave_L-1)+dt_n)*Tstep;
%Centrifigal force
fc=mo*r*wf^2*sin(wf*tspan);
fc=fc(dt_n+1:end);
assignin('base','fc',fc)
```

```
%There is no pulse force fi going to the SDOF Solver in this GUI
fi=zeros(1,length(tspan));
%Call ODE solver
[t, x] = ode23('SDOF_SOLVER',tspan, [0 0]);
%Results and plotting
D=x(:,1);
D=D(dt_n+1:end);
V=x(:,2);
V=V(dt_n+1:end);
A=diff(V);
A(max(size(A))+1)=0;
t=t(1:end-dt_n);
plot(handles.axes1,t,D*1000,t,fc*(max(D)*1000/max(fc)));
plot(handles.axes2,t,V*1000,t,fc*(max(V)*1000/max(fc)));
axes(handles.axes1);
lim=5*1/fHz;
xlim([0 lim])
xlabel('Time (s)')
ylabel('Displacement (mm) ')
legend('Displacement Response','Phase of Unbalance Mass')
axes(handles.axes2);
xlim([0 lim])
xlabel('Frequency (Hz)')
ylabel('Velocity (mm/s)')
legend('Velocity Response','Phase of Unbalance Mass')
%Current information to workspace
assignin ('base','D',D);
assignin ('base','V',V)
assignin ('base','A',A)
assignin('base','t',t)

% --- Executes during object creation, after setting all properties.
function slider4_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% Calculates system response to change of Unbalance Mass slider movement.
function slider5_Callback(hObject, eventdata, handles)

global Mt M mo c k T r wf fi tspan
```

```
FMax=2000;
LOR=6400;
fHz=round(get(handles.slider1,'Value'));
c=round(get(handles.slider2,'Value'),-1);
M=round(get(handles.slider3,'Value'));
k=round(get(handles.slider4,'Value'),-5);
mo=round(get(handles.slider5,'Value'),1);
set(handles.Speed,'String',fHz)
set(handles.Damping,'String',c)
set(handles.Mass,'String',M)
set(handles.Stiff,'String',k)
set(handles.Unbal,'String',mo)
dratio=1;
Mt=M+mo;                    %System mass
wf=2*pi*fHz;                %Shaft turning speed rad/s
r=0.5;                      %radius unbalance mass rotates at
dHz=fHz*dratio;
T=1/dHz;                    %Defect force pulse period
wn=sqrt(k/M+mo);            %Single degree of freedom natural frequency
wr=wf/wn;                   %Frequency Ratio
c_c=2*(M+mo)*wn;            %Critical Damping Value
zeta=c/c_c;                 %Damping Ratio
set(handles.wn,'String',round(wn/(2*pi)))
set(handles.wr,'String',round(wr,2))
set(handles.CritDamp,'String',c_c)
set(handles.DampRat,'String',round(zeta,2))
Wave_L=2.56*LOR;            %Number of points in the timewaveform
Samp_freq=2.56*FMax;        %Frequency of sampling in waveform
Tstep=1/Samp_freq;          %The waveform sample period
dt=1;                       %Time shift of wave for reaching steady state
dt_n=round(dt/Tstep);
%Time constraint for ODE solver
tspan=(0:(Wave_L-1)+dt_n)*Tstep;
%Centrifugal force
fc=mo*r*wf^2*sin(wf*tspan);
fc=fc(dt_n+1:end);
assignin('base','fc',fc)
%There is no pulse force fi going to the SDOF Solver in this GUI
fi=zeros(1,length(tspan));
%Call ODE Solver
[t, x] = ode23('SDOF_SOLVER',tspan, [0 0]);
%Results and plotting
D=x(:,1);
```

```
D=D(dt_n+1:end);
V=x(:,2);
V=V(dt_n+1:end);
A=diff(V);
A(max(size(A))+1)=0;
t=t(1:end-dt_n);
plot(handles.axes1,t,D*1000,t,fc*(max(D)*1000/max(fc)));
plot(handles.axes2,t,V*1000,t,fc*(max(V)*1000/max(fc)));
axes(handles.axes1);
lim=5*1/fHz;
xlim([0 lim])
xlabel('Time (s)')
ylabel('Displacement (mm) ')
legend('Displacement Response','Phase of Unbalance Mass')
axes(handles.axes2);
xlim([0 lim])
xlabel('Frequency (Hz)')
ylabel('Velocity (mm/s)')
legend('Velocity Response','Phase of Unbalance Mass')
%Current information to workspace
assignin ('base','D',D);
assignin ('base','V',V)
assignin ('base','A',A)
assignin('base','t',t)

% --- Executes during object creation, after setting all properties.
function slider5_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

function Mass_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function Mass_CreateFcn(hObject, eventdata, handles)
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Stiff_Callback(hObject, eventdata, handles)
```

```
%

% --- Executes during object creation, after setting all properties.
function Stiff_CreateFcn(hObject, eventdata, handles)
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Unbal_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function Unbal_CreateFcn(hObject, eventdata, handles)
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function CritDamp_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function CritDamp_CreateFcn(hObject, eventdata, handles)
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function DampRat_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function DampRat_CreateFcn(hObject, eventdata, handles)
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function wr_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
```

```
function wr_CreateFcn(hObject, eventdata, handles)
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Speed_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function Speed_CreateFcn(hObject, eventdata, handles)
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')...
        , get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

%END Level_2_6
```

C.20 Single Degree of Freedom Solver Function

```
function xdot = SDOF_SOLVER(t,x)
%function file for a single degree of freedom system with a mass, spring
%and damper
%Position is the first output variable and velocity is the second

global Mt mo c k r wf fi tspan
A = [0, 1;-k/Mt, -c/Mt];
B = [0; 1/Mt];

%Retrieves impulse force fi at each time step from calling code,
%calculate centrifugal force and sum as f for input force

f=fi(round(t/(tspan(2))+1))+(mo*r*wf^2*sin(wf*t));

xdot = A*x + B*f;

%END
```

C.21 Two Degrees of Freedom Solver Function

```
function xdot = TDOF_SOLVER(t,x)
%function file for a two degree of freedom system with a 2 masses,
%2 springs and 2 dampers
%OUTPUTS - Position is the first variable and the second is velocity for
%the first mass, third variable is position and fourth the velocity for
%the second mass

global M C K I wf mo r fi tspan

%Assemble state matrix A and B matrix for system

A = [ [0,0;0,0], I; -(inv(M))*K, -(inv(M))*C ];
B = [ [0,0;0,0]; inv(M)];

%Retrieves impulse force fi at each time step from calling code,
%calculate centrifugal force and sum as f for input force at mass 1
%Force matrix

f1=fi(round(t/(tspan(2))+1))+(mo*r*wf^2*sin(wf*t));
f2=0;
F=[f1;f2];

xdot = A*x + B*F;

%END
```

Appendix D

Risk Analysis

Risk for this project

For undertakings of this project I expect to do all the work myself and any risk associated to affect me. As this work is going to be research based and software based most of it is going to be time sitting in front of a computer. Other tasks include using books, printing and driving a vehicle to buy or use resources. Driving to Toowoomba.

Project Hazard Risk Evaluation (Prior to Controls):

1. Repetitive Strain Injury from typing and using a mouse. Significant/possible chance of occurring as exposure will be frequent/daily. Consequence may be repetitive strain/muscle or nerve strain.
2. Eye Strain from hours sitting in front of a computer. Significant/possible chance of occurring as exposure will be frequent/daily. Consequence may be eye strain, headache, fatigue.
3. Hours sitting in front of a computer. Significant/possible chance of occurring as exposure will be frequent/daily. Consequence may be neck/back/shoulder muscle strain, headache, fatigue.
4. 240V electrical appliances, computers, charger, printers. Significant/possible chance of occurring as exposure will be frequent/daily. Consequence may include electrocution , fire or death.
5. Fatigue needs to be managed. Significant/possible chance of occurring due to work load. Exposure to this risk may be weekly. Consequence may include being tired, irritable, have difficulty concentrating, drowsiness, a need for oversleeping.
6. Driving - Significant/possible chance of occurring as it is not unheard of. Exposure to this risk may twice a month for project related activity. Consequence may include serious injury or death from a road accident.

Controls for reducing the likelihood of project hazards being risks.

1. Repetitive Strain Injury - Have arms at a comfortable distance and arms in a natural relaxed position

2. Eye Strain - Have work computer screen at a comfortable distance so that screen text is not too small. Have sufficient light for reading, use a desk lamp as needed. Adjust screen brightness depending on ambient light levels. Minimize screen glare.

3. Hours sitting in front of a computer - Minimize poor posture. Take regular breaks. Have a clean workspace and ample desk areas to have paperwork and books easily at hand. A small book stand on desk for storage. Comfortable reach distances.

4. 240V electrical appliances - Ensure 240V wall sockets and leads are in good condition. Only use 240V in a dry environment. Visually inspect 240V equipment for damage. Work office printers must have current inspection tags.

5. Fatigue - Try and maintain normal working hours, not staying up too late. Plan work hours. Get sufficient sleep 7-8 hours a night. Don't let sleep debt accumulate. Take breaks to rest eat and hydrate. be aware of adverse effects of alcohol, drugs. Be aware fatigue symptoms. Maintain nutrition and fitness.

6. Driving - Drive to conditions. Minimize need to project related travel by planning activities to coincide with other journeys. Slow down in poor visibility, wet and dark conditions. Obey traffic rules. Take a break at least every second hour.

Some Condition Monitoring Risks

There are risks associated with the condition monitoring technique described in this work. Risks associated with vibration monitoring techniques cover inspection of machines while they are in a dynamic state, possibly working in a production phase, as well as some of the hazard that might be found when the machine is in a static condition. The risks cover exposure to rotating components, process fluids such as chemicals, oils and greases, and product from a conveyor or leaking slurry, exposure to moving/mobile machinery, vibration, motion sickness, noise, dust, hot surfaces, hot working environment from being inside working machine compartments, wet environments, mud, slippery railings, poor equipment condition, electrocution, failure of equipment while testing, working in isolated areas with limited communication.

Similarly with particle monitoring associated there are risk while collecting and handling the oil/grease samples and there are risks associated with the testing processes. The collection side of the task requires either collecting samples from equipment taken off line

and switched off/isolated and from equipment that is running. A technician must either be trained how to isolate energy in a system and release residual energies that might be harmful, otherwise they need a trained and competent person to do it for them. Fluid injection injury is very dangerous and requires surgical removal by opening up the body and scraping/washing it out and can be lethal.

Appendix E

Resource Analysis

Resource Analysis

The basic resources required for this project are a computer and some software. To build the virtual training package software used will be Microsoft Excel 2013 or MatlabSuite for students R2015a.

A copy of Microsoft 2013 is available for use if the training program is built with this.

Matlab for this project can be purchased online for $109 and a cost of $32 if additional add ons are required. This can be purchased and downloaded from <http://au.mathworks.com/pricing-licensing/index.html?intendeduse=student>. This can be loaded onto multiple computers if needed or if the primary computer fails. The software can be purchased and used in this case under the agreement that it is used by the student in a degree granting university.

A computer with Windows 8.1 and Excel 2013 is available for use if the primary working computer suffers a failure.

An external hard drive is available to be set up and auto save work in progress. Additional backing up of the project will occur after a working session and be removed from the work station in the event a fire destroys the work computer and back up drive.

A car is available to acquire resources if they are available in the local area.

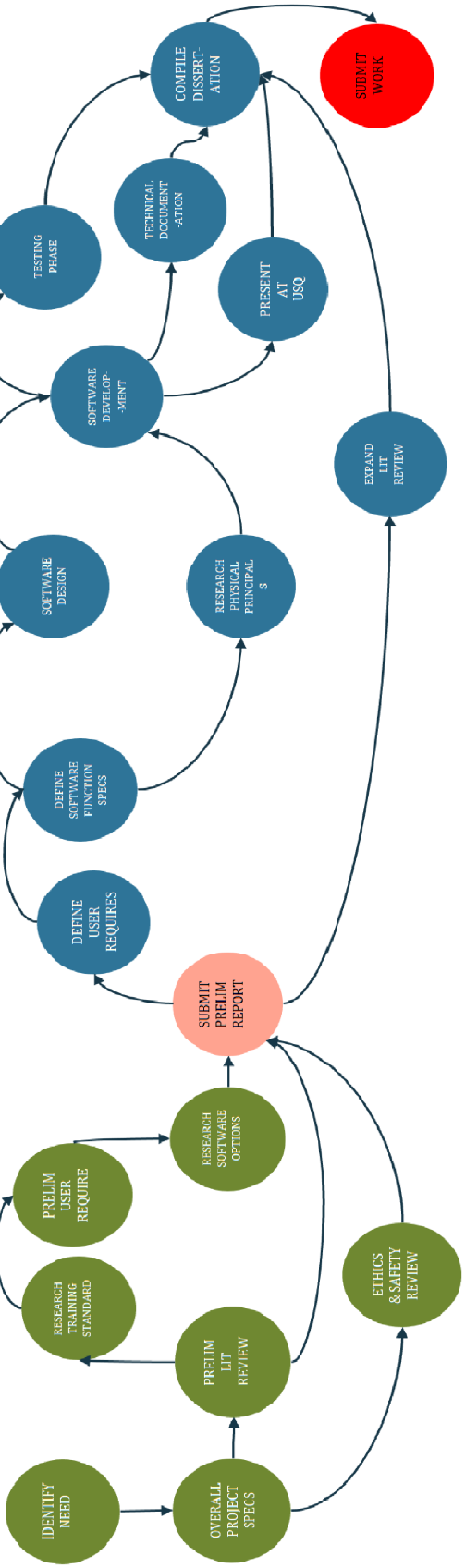
Loss of electricity and the internet for any extended period could be the biggest threat affecting the use of computer, sourcing information and affect communication with USQ. If this was to occur an alternative site to work from would be required. A generator could be acquired if absolutely necessary.

Funding this project is within the allowable budget.

Appendix F

Project Planning

F.1 Project Flowchart



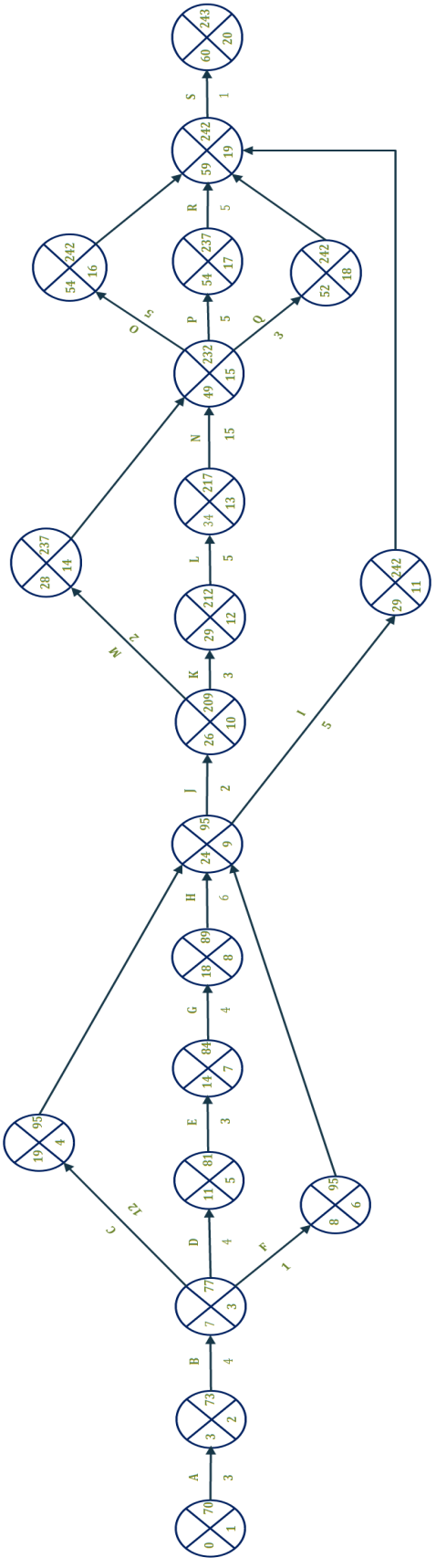
F.2 Project Activity List

Events 1-9 have a 95 day limit Event 9 is Preliminary Report Submission Day 243 is Dissertation Submission If events could be completed concurrently the Critical Path in this network would be A-B-C-D-E-G-H-J-K-L-N-P-R-S In essence as this project is completed by only one person all network paths are essentially critical

Table F.1: This table list project events, activities, durations and start finish times.

Events	Activities	Code	Duration	Requisite	Start Time(Days)		End Time(Days)		Start Time(Weeks)		End Time(Weeks)	
					Earliest	Latest	Earliest	Latest	Earliest	Latest	Earliest	Latest
1-2	Identify Need	A	3	-	0	70	3	73	0	10	0.4	10.4
2-3	Project Specification	B	4	A	3	73	7	77	0.4	10.4	1	11
3-4	Prelim Literature Review	C	12	B	7	77	19	95	1	11	2.7	13.6
3-5	Research Training Standards	D	4	B	7	77	11	81	1	11	1.6	11.6
5-7	Prelim User Requirements	E	3	D	11	81	14	84	1.6	11.6	2	12
3-6	Ethics and Safety Review	F	1	B	7	77	8	95	1	11	1.1	13.6
7-8	Research Software Options	G	4	E	14	84	18	89	2	12	2.6	12.7
8-9	Submit Preliminary Report	H	6	C,F,G	18	89	24	95	2.6	12.7	3.4	13.6
9-11	Expand literature review	I	5	H	24	95	29	242	3.4	13.6	4.1	34.6
9-10	User Requirements	J	2	H	24	95	26	209	3.4	13.6	3.7	29.9
10-12	Software Funtional Specs	K	3	J	26	209	29	212	3.7	29.9	4.1	30.3
12-13	Research Physical Concepts	L	5	K	29	212	34	217	4.1	30.3	4.9	31
10-14	Software design	M	2	K	26	209	28	237	3.7	29.9	4	33.9
13-15	Software development	N	15	L,M	34	217	49	232	4.9	31	7	33.1
15-16	USQ presentation	O	5	N	49	232	54	242	7	33.1	7.7	34.6
15-17	Testing software	P	5	N	49	232	54	237	7	33.1	7.7	33.9
15-18	Technical documentation	Q	3	N	49	232	52	242	7	33.1	7.4	34.6
17-19	Compile Dissertation	R	5	I,O,P,Q	24	237	59	242	7.7	33.9	8.4	34.6
19-20	Submit Dissertation	S	1	R	59	242	60	243	8.4	34.6	8.6	34.7

F.3 Project Network Diagram



F.4 Gantt Chart