POLITECNICO DI TORINO

Department of Mechanical and Aerospace Engineering

Master Degree in Aerospace Engineering

Master Degree Thesis

Methods for the analysis of aeronautical gearbox experimental data

Supervisors
Prof. Daniele Botto
Ing. Luca Ronchiato

Candidate
Lorenzo Capra

July 2019

Ringraziamenti

Desidero ringraziare i miei genitori, e la mia famiglia, per tutti i sacrifici che hanno fatto per permettermi di portare avanti gli studi, e per avermi sempre sostenuto ed incoraggiato durante il periodo universitario. Un ringraziamento va ai miei compagni di corso, con i quali ho condiviso gioie e sofferenze tra i banchi universitari. In particolare ad Alberto, per gli innumerevoli caffé e passaggi offerti a Rivalta. Un ringraziamento speciale va anche ai miei amici di Aosta: Jacopo, Riccardo, Pietro, e Simone, che la nostra amicizia sopravviva al tempo e alle distanze. Volevo, inoltre, ringraziare la mia fidanzata Adriana per aver tollerato la mia compagnia da quasi tre anni a questa parte.
Un ringraziamento va anche ai colleghi tesisti e borsisti di Avio che hanno contribuito ad alleggerire le giornate al GreatLab, e all'ingegnere Luca Ronchiato, che ha sempre dimostrato disponibilitá e riconoscimento nei confronti del lavoro da me svolto.

Summary

The aim of this work is to illustrate the tools for signal analysis concerning the processing of experimental data of aeronautical gearboxes. A code has been developed as part of this work with the purpose of allowing even an inexperienced user to obtain an acceptable post processing. This is possible thanks to dedicated functions that cover the role of determining the appropriate analysis parameters for the specific experimental data and from the requested output. The code has been validated and tested on some GE Avio s.r.l. gearboxes.
In chapter 1 an overview of the mechanism generating noise and vibrations in gears is given, along with some basics of gears dynamics. In the same chapter, some information about maintenance philosophies and sensors are illustrated.
In chapter 2 the basics of signal processing are given. The various kinds of signals are illustrated, particular focus has been given to non-stationary signals since they represent gearbox experimental data category. An overlook of the Fourier transform is given, with particular interest on the practical applications of the Discrete Fourier Transform (DFT). Sampling, truncation, and windowing, which are the steps required toward applying a DFT, are explained, along with the choice of the analysis parameters (i.e. windowing period, windowing function, sampling frequency, etc.), and their effects on the resulting frequency spectrum.
Chapter 3 explains how the post processing code performs the frequency analysis. In particular, by means of the automatic parameters selection feature, the user can leave almost every analysis settings to be determined by the code itself. The most influential and difficult parameter to determine is the windowing period. To obtain this value the code takes into account the closest orders to be separated and the steepness of the speed ramp (i.e. the acceleration), thus it is equipped with a dedicated function which maps the speed profile and finds the accelerations of the reference shaft. The task of such function is particularly difficult to complete due to the extremely unpredictable nature of experimental data, and the required tailored filtering that comes along to eliminate random noise. The output produced, typically a Campbell diagram, can be referred to either speed or time, both these configurations are discussed. In particular concerning the most common configuration of a speed x-axis frequency spectrum the issue concerning the representation of unevenly spaced colormap is discussed.
Chapter 4 deals with the code function entitled to generating the order analysis Campbell diagram. The first step is to convert the tachometer voltage signal into the speed data, this requires maximum precision in order to avoid resampling errors. Afterwards the signal gets resampled synchronously to the reference shaft, and then its frequency spectrum is extracted. Even for this analysis most of the parameters can be set automatically by the code, in this case though, due to the signal being resampled synchronously with the reference shaft, there is no need to find the shaft accelerations. The windowing period is set only considering the closest orders to separate. The chapter ends with some considerations about the order colormap issue of bins duplication.
Depending on the experimental data the frequency spectrum might not be required, in

chapter 5 the resampling function and the averaging function are illustrated. The first one usually is useful whenever the signal comes from a sensor rotating with the reference shaft. An example is illustrated of a strain gauge measuring the strain at the tooth root while meshing with another gear tooth. It is also shown the kind of errors that might arise while resampling due to erroneous conversion of the tachometer signal. The last section concerns the averaging function and the output produced by it.

Contents

List of Figures

List of Tables

Chapter 1

Introduction to gear vibration

Vibration analysis is a key tool for condition monitoring of rotating machinery (such as fans, motors, pumps and gearboxes). In particular it allows to identify what causes vibrations, whether it is a machine fault, such as misalignment, unbalances, etc. or resonances due to excitation forces acting within the machine resonance frequency range.
Vibration analysis can be split into two main phases: data gathering and data post processing. The first one typically consists of a test run of the machine at various operational rotational speeds. During that time period sensors, which are placed in crucial positions, measure displacements, speeds, or accelerations, and their readings are saved in the so-called raw data file. Each sensor type is particularly suited for measuring signals at different frequencies, therefore their choice criterion is something not be neglected. For example velocity sensors are better suited at measuring low to mid frequencies (10 Hz to 1500 Hz), while displacement sensors tend to measure with higher accuracy low frequencies [2].
In the second part the acquired data is processed to produce the preferred output. According to the aim of the analysis there are two main types of analysis that can be carried out within signal analysis:

1. frequency analysis

2. order analysis

both analysis cover a specific role which will be explained throughout this work, along with other tools to post process experimental data.

1.1 Vibration and noise origins in gearboxes

Before analyzing different vibration analysis techniques a brief overview on the origins of gears vibration is given.
Gearboxes are machines made up of several gears whose aim is to transfer angular velocity and torque from the input shaft to the output according to the gear ratio, which is determined by the number of teeth of the meshing gears. In aviation, gearboxes are typically used to transfer power from the engine to the accessories such as the hydraulic system and the air conditioning unit. Other aviation gearboxes are employed in turboprop engines to reduce the propeller shaft speed which otherwise would result in low propeller efficiency.
To describe the mechanisms that generate vibrations the assumption of loaded gears has to be made. In such condition, gears transfer both torque and rotational speed. If this condition is not satisfied the gears teeth move within their backlash range and

produce rattling noise, this is not a common operating condition in aviation gearboxes and therefore it is not within the focus of this work. Once this assumption is considered to be valid, it is possible to explain one of the causes of vibrations which is the small mechanical shock occurring whenever two teeth mesh. The impact generated by the contact between them transmits vibrations to the gearbox structure which are not naturally damped. The frequency at which these shocks occur depends on the number of teeth of the gears and the rotational speed of the gear, such frequency is generally referred to as the mesh frequency, which is defined as follows:

$$f_{GMF} = nf_0 \tag{1.1}$$

where f_0 is the rotating frequency of a gear and n is the number of teeth. The f_{GMF} is one of the most prevalent sources of excitation in any gearbox.
Whenever two gears mesh they exchange angular velocity and torque through teeth contact. During that period of time the contact stiffness of the two bodies (also known as mesh stiffness) varies mainly due to two phenomena: the contact point moving towards the teeth flanks and the number of meshing teeth changing. These two effects combined generate mechanical vibrations. In figure 1.1 for example, the variation of mesh stiffness is shown as function of ε_α defined as profile contact ratio, which indicates the number of meshing teeth at any given angular position of the gear. Just by looking at the two plots it is clear that two gears whose ε_α is constant, such as the one in the left of figure 1.1, are subject to less vibrations. As the gears transfer mechanical power a force acting along the action line F_t represents the force exchanged between the two gears. Such force has to be compensated by another force acting on the gear shaft to keep the gears from separating, called F_s. These two forces together generate a torque on both gears as shown in figure 1.2, the oscillation of the mesh stiffness value causes the force F_t to change over time, forcing the equilibrating force F_s to change as well, transmitting vibrations to the gearbox casing.
Generally speaking there could be some other source of excitation not originating from teeth contact such as bearings imperfections, input torque not being constant, shafts imbalances, and so on. It is important to know how to recognize vibrations induced by regular gearboxes features, such as teeth meshing, from vibrations originating from anomalies, such as teeth flanks imperfections.

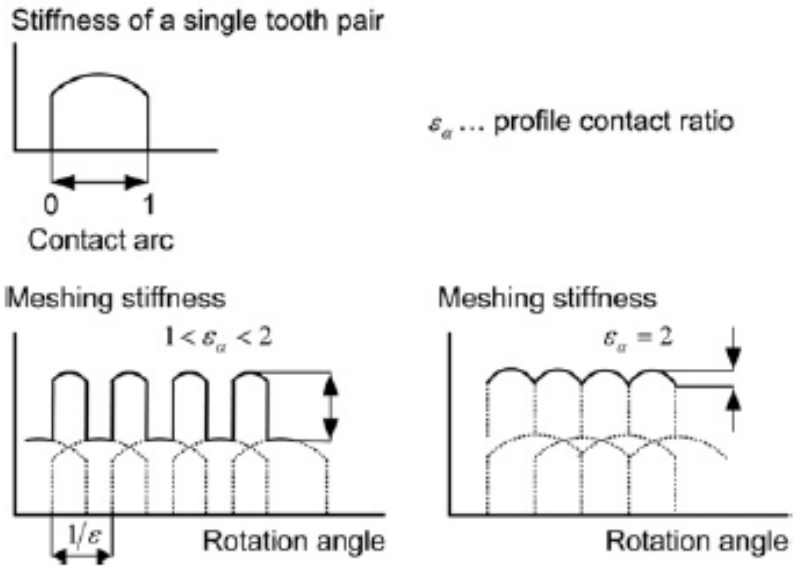


Figure 1.1: Mesh stiffness

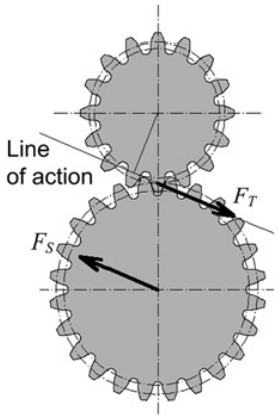


Figure 1.2: Forces acting on meshing gears

1.1.1 Fundamentals of gear dynamics

In order to better understand every function of the code a brief introduction to gear dynamics is required.
Every mechanical system can be described as a set of elements with a finite number of degrees of freedom (d.o.f). In reality the d.o.f is infinite, hence the analytic solution of the equation of motion yields an approximation, rather than an exact representation of reality. However, usually the two solutions are not too distant, therefore, the model with a discrete amount of d.o.f. can be employed, and its results can be trusted to be an accurate enough representation of the real case.
In figure 1.3 the scheme of a two d.o.f. system is depicted. In this particular case there is no damping between the two masses since the only link between them is a spring, which represents an elastic force, this kind of system is called conservative.

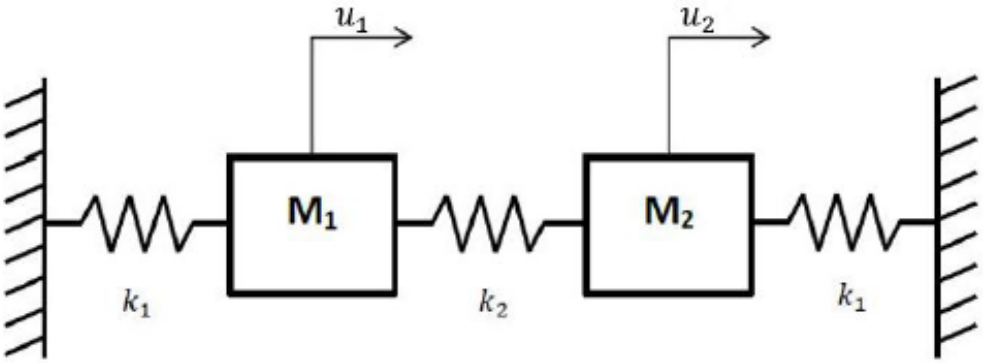


Figure 1.3: Scheme of a 2 degrees of freedom system

This assumption is generally valid for systems representing gears.
The two masses equations of motion are:

$$\begin{cases} m_1\ddot{u}_1 + k_1u_1 - k_2(u_2 - u_1) = F_1 \\ m_2\ddot{u}_2 + k_1u_2 - k_2(u_2 - u_1) = F_2 \end{cases} \tag{1.2}$$

which can be written in matrix form as:

$$\mathbf{M\ddot{u}} + \mathbf{Ku} = \mathbf{F} \tag{1.3}$$

In case damping is to be considered the previous equation becomes:

$$\mathbf{M\ddot{u}} + \mathbf{B\dot{u}} + \mathbf{Ku} = \mathbf{F} \tag{1.4}$$

where $\mathbf{M}$ is the symmetric mass matrix:

$$\mathbf{M} = \begin{bmatrix} m_1 & 0 \\ 0 & m_2 \end{bmatrix}$$

while $\mathbf{K}$ is the symmetric stiffness matrix:

$$\mathbf{K} = \begin{bmatrix} k_1 + k_2 & -k_2 \\ -k_2 & k_1 + k_2 \end{bmatrix}$$

$\mathbf{F}$ is the force matrix:

$$\mathbf{F} = \begin{bmatrix} F_1 \\ F_2 \end{bmatrix}$$

and, if needed, $\mathbf{B}$ is the damping matrix, having the same form of the stiffness matrix.
Considering equation 1.3, in case $\mathbf{F} = 0$, the solution obtained solving the equation would represent finding the free response of the system, which identifies the frequencies at which the system naturally tends to vibrate. The solution in term of displacement $u(t)$ can be written in the following form:

$$\mathbf{u} = \mathbf{A}\cos(\omega t + \phi) \tag{1.5}$$

where $\mathbf{A}$ is a non null constant vector, ω is the frequency of the vibration, and ϕ is the phase. By substituting equation 1.5 into equation 1.3, the following eigenproblem results:

$$(\mathbf{K} - \omega^2\mathbf{M})\mathbf{A} = \mathbf{0} \tag{1.6}$$

in order for the vector $\mathbf{A}$ to be different from the trivial solution (which would be the null vector), the following relation has to hold:

$$\det(\mathbf{K} - \omega^2\mathbf{M}) = 0 \tag{1.7}$$

in the simple example of a system with two d.o.f. taken into consideration, the number of ω solving equation 1.7 is two, in general for systems with n d.o.f. the number of ω is n. The frequencies obtained through this relation are the natural frequencies of the system. Additional information on this subject can be found in literature, for example [1].
Knowing the natural frequencies of a system is pivotal for any dynamic analysis. As a matter of fact thanks to this information it is possible to derive the deformation shapes, also known as mode shapes, and subsequently to calculate the dynamic response of the system to a time variable excitation force. In particular, if the excitation forces act on the system at frequencies close to a natural frequency, the system undergoes a resonance. A resonance is a dangerous phenomenon which causes the amplitude of the vibration to grow in an unstable fashion over time. Generally speaking it is always a good practice to avoid a system to have its operative range within the range of a resonance frequency.
As mentioned previously, the knowledge of the natural frequencies allows for definition of the mode shapes of the body being excited by external forces. Considering two gears meshing with each other, as they begin to vibrate, not every region is deformed in every time instant, in general there can be some regions being not deformed at any given time. Since gears are symmetric bodies, the deformation has the same feature. Due to the symmetry of the deformation, regions with zero displacement usually lay on the same line, which is referred to as nodal diameter. Nodal diameters thus are symmetry planes of the deformation.

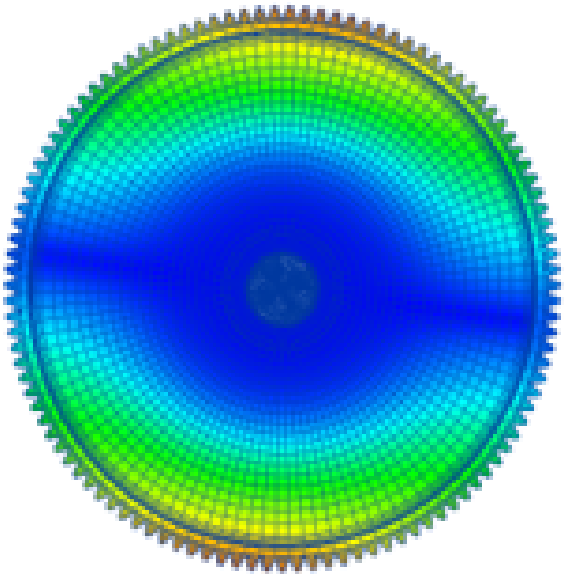


Figure 1.4: Gear 1D axial mode shape

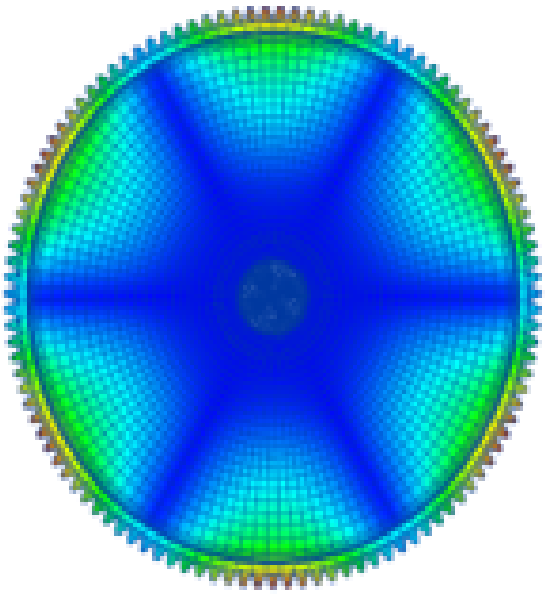


Figure 1.5: Gear 3D axial mode shape

Mode shapes are defined through the number of nodal diameters. From figures 1.4 and 1.5, the regions in blue have zero displacement, it is evident that the first one is a mode shape with one nodal diameter (1D), while the latter shows three nodal diameters and is therefore called a 3D mode shape.
As mentioned in the previous section, gears vibrate due to variation of teeth contact conditions. Despite not being the only existing source of excitation, meshing teeth is usually the most relevant for gearboxes. The frequency at which teeth mesh with each other is described by the gear mesh frequency (1.1), which is dependent on gears angular speed. Whenever the gear mesh frequency reaches a natural frequency a resonance might occur. In reality, due to gears deformation, the natural frequency shows a dependency with rotational speed, if observed from an inertial reference frame. In particular, the effects of two harmonics is especially noticeable, depending on the signs of these two harmonics the natural frequencies are influenced. The relation describing the natural frequencies is the following:

$$f = f_0 \pm N_d \Omega \tag{1.8}$$

which is valid for modes displaying axial displacement. In equation 1.8, f_0 is the original natural frequency value, N_d is the nodal diameter of the mode shape of the natural frequency, and Ω is the rotational speed. A graphical representation of equation 1.8 is shown in figure 1.6. The two harmonics can have the same sign, and therefore add up to form the so called forward resonance frequency (simply called forward), in which the mode shape rotates in the same direction as the gear generating the highest frequency of the mode.
Alternatively, the two harmonics can have different signs, thus subtracting, and form the backward resonance frequency (bwd), in which the mode shape and the gear rotate in opposite directions giving birth to the lowest frequency of the mode.
Campbell diagrams are a useful tool for allowing to picture at which speeds potential resonances occur, and how to avoid them. The most important output of the code that will be described is definitely the frequency versus speed Campbell diagram, alongside the Autoslice function which is particularly useful for detecting modes and orders crossings.
In case the displacements are radial, the slopes of the forward and backward are not simply equal to the nodal diameter of the mode shape and a correction has to modify the slope to consider that effect. The correction is particularly useful to evaluate radial mode

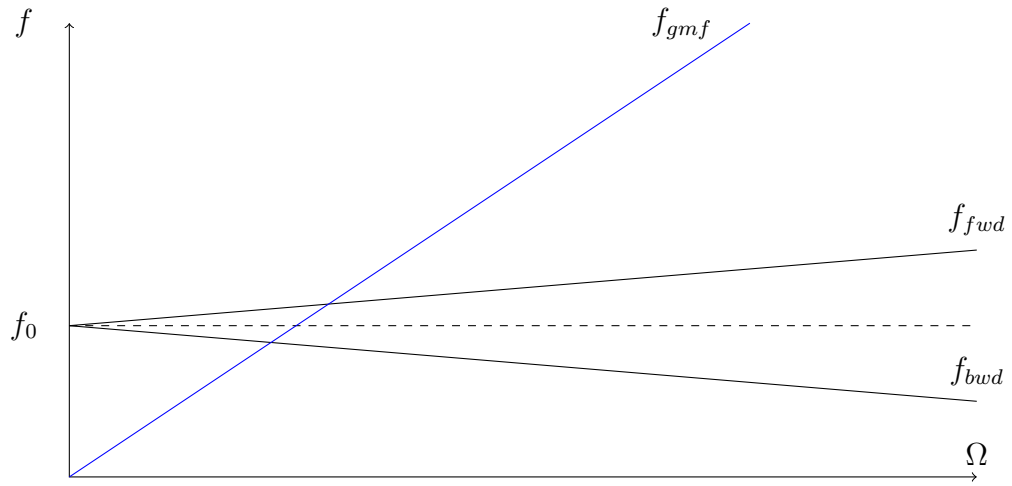


Figure 1.6: Fixed reference frame Campbell diagram showing forward and backward frequencies

shapes, while for axial mode shapes equation 1.8 holds.

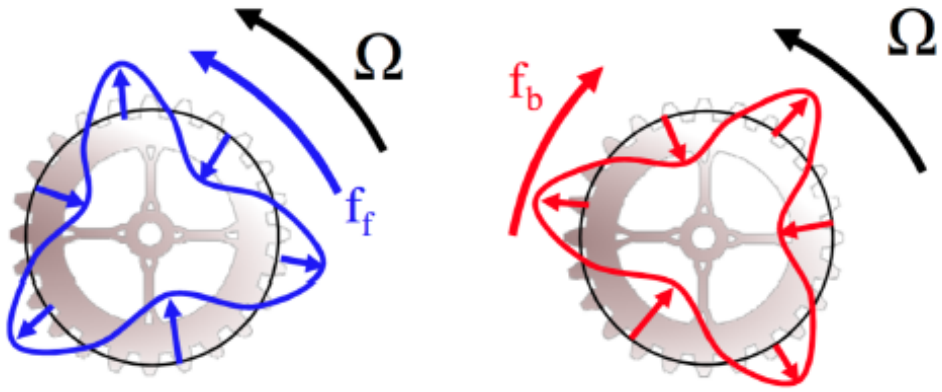


Figure 1.7: Visual representation of forward and backward frequencies

1.2 Maintenance philosophies

From [2] it is given an overview of maintenance philosophies employed at different times in history concerning plants and machinery.
At the beginning of the industrial age vibration knowledge was rudimentary, hence frequency analysis was never applied. Machines simply were set to run until a fatal breakdown occurred, at which point a diagnosis was executed and the machine was repaired. During that time interval the plant might have been on shutdown depending on the relevance of the broken machine. Such a philosophy, known as run to failure maintenance, has many evident drawbacks.
First of all, if spare pieces are needed they have to be acquired only after the breakdown has occurred, causing a delay in the maintenance operations which produces a relevant money loss. Besides, maintenance personnel will likely be overworked, due to working without any planning and to the pressure of avoiding any money wasting time delays. This is without a doubt an inefficient way of planning maintenance, it can work only if applied to machines which shutdown does not effect production.

Maintenance can also be scheduled and performed following a calendar based on the work hours of the machines. This approach is called preventive or time-based maintenance. It works well on machinery that operates only at times and not continuously. With respect to the run to failure philosophy it offers the advantage of avoiding unplanned shutdowns, which depending on the machine relevance to the production processes can save a money loss to the plant. The main disadvantage of preventive maintenance is that repairs or pieces replacements might occur too early or too late. It is in fact impossible to predict exactly when a component will wear out due to fatigue. This leads to either perfectly working components being replaced and discarded or to shutdowns due to an unplanned failure. There also is the occurrence of degraded machine performance caused by a component damaged before its scheduled maintenance. Another risk is posed by an incorrect maintenance operation scheduled to a well working machine. While preventive maintenance seems to offer better efficiency with respect to run to failure maintenance, it can be improved with more sophisticated failure prediction methods.
Similar to preventive maintenance, condition-based maintenance consists in scheduling repairs only if a degradation in performance is detected. Machines will undergo testing to reveal their performance (usually some kind of efficiency test) periodically and shut down at the most convenient time for the plant to allow for maintenance operations to be performed. This philosophy also allows the purchasing of necessary repair pieces with large time advances. Since condition-based maintenance relies heavily upon machine monitoring if such an operation is carried out improperly, or by unskilled personnel, unnecessary maintenance operations would be performed, with possibly perfectly working components being replaced. It is therefore required to train maintenance personnel adequately in order to adopt this philosophy minimizing its potential drawbacks.
The last and most advanced maintenance philosophy is called proactive or reliability centered maintenance. It is identical to condition-based maintenance with the addition of the identification and fixing of the source of the problem in order to avoid its repetition over time. For example in case a shaft unbalance has caused a failure in a machine, the maintenance personnel should work on the balancing of the shaft properly to avoid the defect from causing more failures in the future. The philosophy emphasizes the need to improve machines reliability once a failure is detected, in some cases there can be the need to redesign equipment to avoid recurrence of the problem. Again, similarly to predictive maintenance, proactive maintenance allows for purchase of spare parts with time advantage and maintenance scheduling. If carried out properly it offers the most reliable maintenance of all the various philosophies; its only drawback lies on the highly trained personnel required in order to perform the different operation.

1.3 Vibration analysis role in maintenance

Vibration analysis is a key tool to perform condition-based maintenance [2]. Condition-based maintenance revolves around controlling certain mechanical parameters in order to determine approximately the work time before failure of a machine. Besides vibration analysis, there are several ways to monitor a machine such as thermography, ultrasounds, oil and wear debris analysis, etc. Each one of them is particularly effective on some machines more than the others. Among the various diagnosis techniques it can be said that vibration analysis is without a doubt the most effective technique for rotating machinery. At times also acoustic detection can be relevant for gearboxes because they are cyclic machines producing a tonal noise, which means that the noise frequency spectrum consists of sinusoidal components with low-level background noises. Concerning bearings and gears,

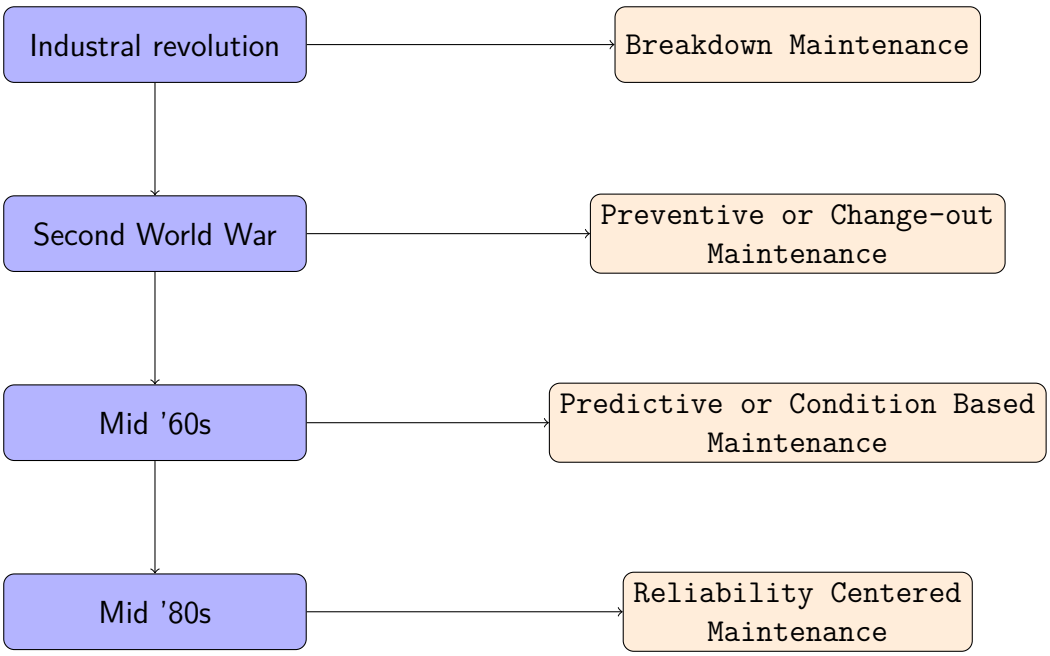


Figure 1.8: Maintenance philosophy in modern history

they are generally diagnosed through oil or lubricant analysis which if scanned through a microscope can reveal information about the condition of such pieces. Sometimes worn machines release debris while functioning, particle analysis can be a useful tool for such machines. Lastly, thermography is generally performed on running machines to spot mechanical or electrical defects in generators, overhead lines, and boilers. It can also detect damaged carbon fiber in composite aircraft structures.
Similarly to the human body, machines exhibits symptoms whenever they are not working properly. However not every symptom can be easily detected, this is where predictive maintenance come into help to perform the proper diagnosis.
The vast majority of machines failure derives from either unbalance or misalignment. Vibrations amplitude measurement not only can detect such faults, but it can also identify poor maintenance practices, such as improper bearing installations, which will eventually lead to such failures.
In general it can be said that vibration analysis covers a multitude of roles throughout the maintenance life cycle of a machine. It is useful as the machine is installed in the plant to reveal improper deployment; it is a relevant tool in failure prevention and performance monitoring; and, finally, it is a fundamental diagnosis tool once a failure or a performance degradation occurs.

1.4 Data acquisition

In order to acquire a vibration data, sensors are mounted on the machine. There exist many kinds of sensors, the most commonly used are velocity, acceleration and proximity sensors, each one of them is more suitable to different machines and frequencies. There is not a universally appropriate sensor, the most convenient choice depends on the situation. A brief introduction about the most commonly employed sensors is now given.
Despite measuring different machine characteristics, sensors all work in a similar way converting the vibration mechanical energy in to some sort of electric signal.
Velocity sensors are the oldest sensors employed to measure vibrations on rotating machinery, and they are still considered to be sensors of very wide application. The most

common configurations of a velocity transducer are the ones displayed in figure 1.9. The configuration on the right is known as coil-in-magnet, as the coil (which is free to move upwards and downwards) moves, the magnetic flux changes generating a tension at the ends of the coil which describes the coil motion. The configuration on the left, called magnet-in-coil, is composed of a permanent magnet which is free to move, a spring, which supports the magnet, and some oil, which fills the sensor and serves as damper. Similarly to the coil-in-magnet configuration, the motion of the magnet varies the magnetic flux through the coil generating a tension describing the magnet motion. Both the sensor

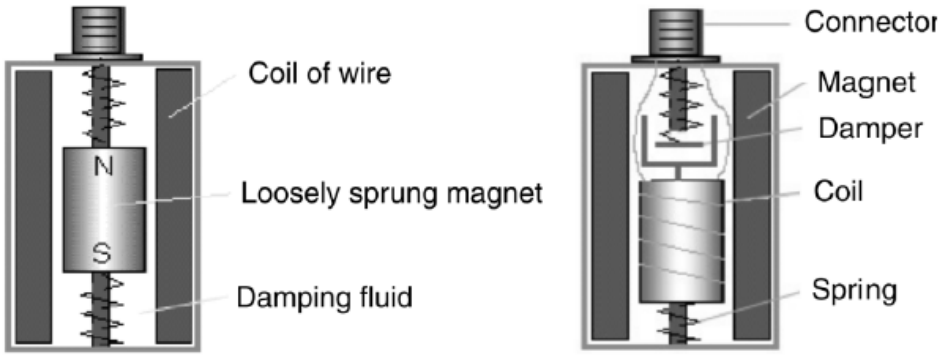


Figure 1.9: Velocity pickup mechanisms

architectures are sensitive to gravity action on to the magnet and to external electrical and magnetic fields. It is, therefore, important to consider such aspects while mounting the pickup.
Velocity pickups have the highest sensitivity out of every vibration sensor, which results in large output values. This is especially useful in case the external noise is present. The sensitivity is constant over a frequency range which depends on the specific sensor. For velocity pickups such interval is usually 10 Hz to 1 kHz. At low frequencies the relative motion between coil and magnet is practically null, as the two parts move at the same speed. This means that speed readings below 10 Hz tend to be very inaccurate.
Acceleration transducers are the most common vibration sensor for rotating machinery application. They are small, compact, and do not have moving parts which could wear over time and require a sensor recalibration from time to time. A typical acceleration sensor works measuring the inertial force exchanged by a seismic mass thanks to a piezoelectric material. The output signal is proportional to the vibration acceleration. When the mass is subjected to an acceleration it exerts a pressure on to the piezoelectric crystal which produces a tension proportional to its deformation, and hence to the pressure. As shown in figure 1.10 the transducer requires an amplifier to work. Because of this component, acceleration sensors need an external power input, unlike velocity pickups. The range of frequencies over which this kind of sensors are effective typically is from 1-2 Hz to 8-10 kHz.
Proximity probes see most of their usage on high-speed turbomachinery. Their typical architecture is depicted in figure 1.11 and consists of a probe, an extension cable and an oscillator/demodulator. A high frequency signal is generated by the movement of the target then going through the extension cable and is emitted by the probe. The oscillator/demodulator receives the signal and demodulates it, the DC portion of it is directly proportional to the distance and the AC to the vibration.

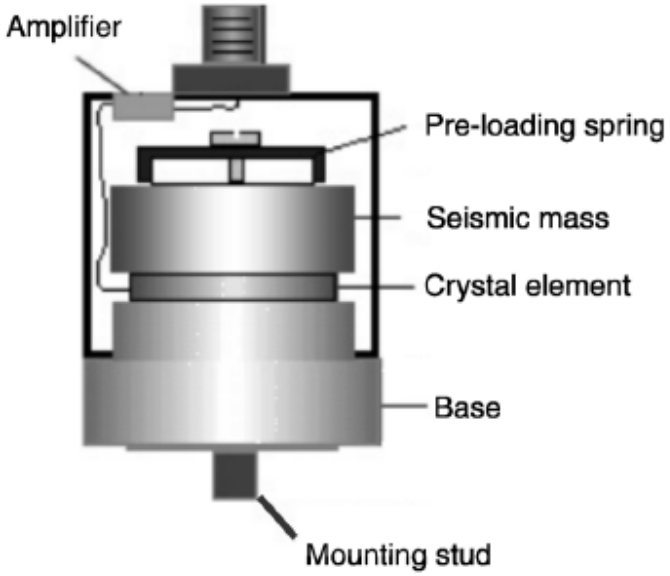


Figure 1.10: Acceleration transducer mechanisms

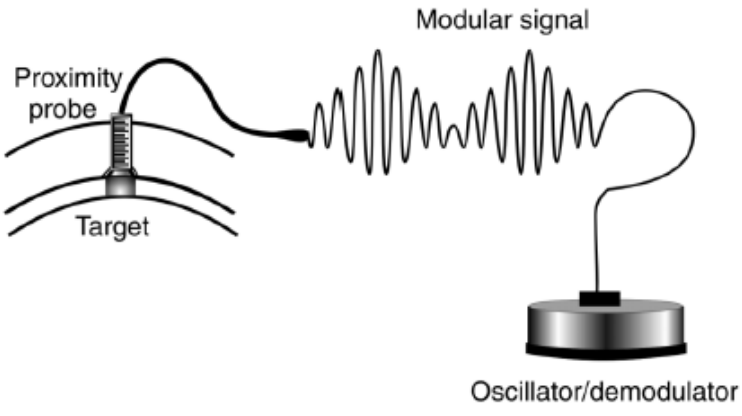


Figure 1.11: Eddy current transducer mechanisms

Chapter 2

Introduction to signal analysis

2.1 Signal classification

There are several kinds of signal that can be analyzed, each type requires a different set of analysis parameters. For instance every signal can be classified as either stationary or non-stationary. The first group is generated whenever a test run is conducted at constant speed, hence the frequency content is stationary over time, while the latter is often the result of a system which is going through acceleration or deceleration. It can also be said that a signal, in order to be defined as stationary (or time invariant), has to maintain its average value constant over time, and not dependent on sampling time.
Among stationary signals there are deterministic signals, which output depends solely on the input received, and random signals, which output cannot be exactly predicted based on the input, but whose average, variance and other statistical properties do not change over time. Non-stationary signals can be divided in transient signals, which begin and

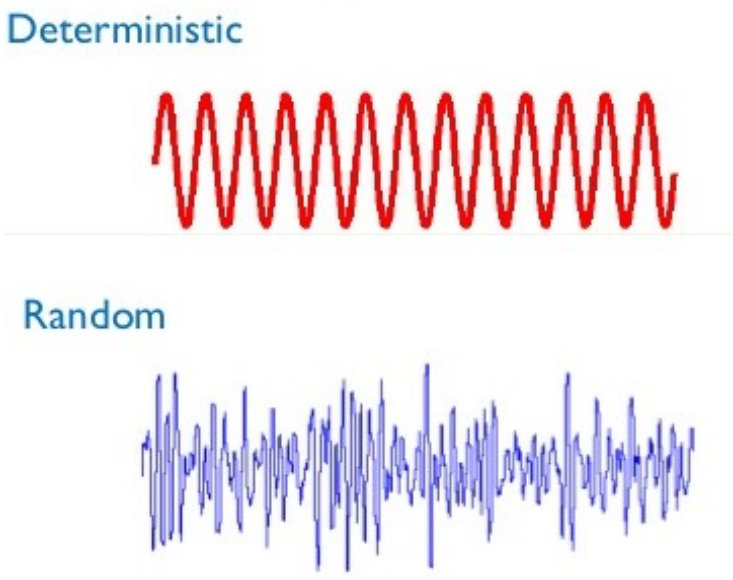


Figure 2.1: Examples of random and deterministic signals

end at zero, and continuous signals in which this does not happen.
A typical gearbox vibration data is a non-stationary, deterministic, continuous signal. As a matter of fact usually the gearbox input shaft undergoes cycles of acceleration and deceleration since the machine has to be tested at the various operative conditions. This kind of speed profile makes the resulting signal inherently non-stationary. Besides, the signal is strictly dependent on the speed and the torque at which the machine is

operating, hence the deterministic feature. According to the speed profile of the test the signal could be both continuous, if the data do not begin and end at stationary condition, and transient, the latter being a more common feature.

2.1.1 Fourier transform

According to Fourier theorem, every periodic signal can be represented as a Fourier series, which is a sum of sine and cosine functions with variable amplitude, phase and period. The Fourier series is defined as follows:

$$x(t) = a_0 + \sum_{n=1}^{\infty} \left[a_n \cos\left(n \frac{2\pi}{T} t \right) + b_n \sin\left(n \frac{2\pi}{T} t \right) \right] \tag{2.1}$$

representing a signal through the Fourier series means determining the weight of each n term, where n is the harmonic. The amplitude of the n^{th} harmonic is defined by the coefficients a_n and b_n, which are called the Fourier coefficients, and are defined as follows:

$$a_n = \frac{2}{T} \int_{-T/2}^{+T/2} x(t) \cos\left(n \frac{2\pi}{T} t \right) dt \tag{2.2}$$

$$b_n = \frac{2}{T} \int_{-T/2}^{+T/2} x(t) \sin\left(n \frac{2\pi}{T} t \right) dt \tag{2.3}$$

where T is the interval in which the signal is defined.
Signal analysis aim is to identify every frequencies composing the signal and its contribution to the overall amplitude. To obtain the frequency spectrum of a signal the Fourier transform operator is employed. Its job is to convert a signal domain from temporal to frequency based. For example, in figure 2.2 a signal originated from the sum of the first three harmonics of the Fourier series of a square wave which is described by the following equation:

$$f(x) = \frac{4}{\pi} \sum_{n=1,3,5,\cdots}^{\infty} \frac{1}{n} \sin\left(\frac{n\pi x}{L} \right) \tag{2.4}$$

where $2L$ is the length of the square wave, and n is the index identifying the harmonics. The leftmost plot shows the sum of the three components in the time domain. The frequency domain representation highlights the amplitude of the periodic components of the signal and is the result of applying the Fourier transform to the time domain signal. Thanks to this representation it is possible to understand which harmonics contribute the most to the overall amplitude of the signal.

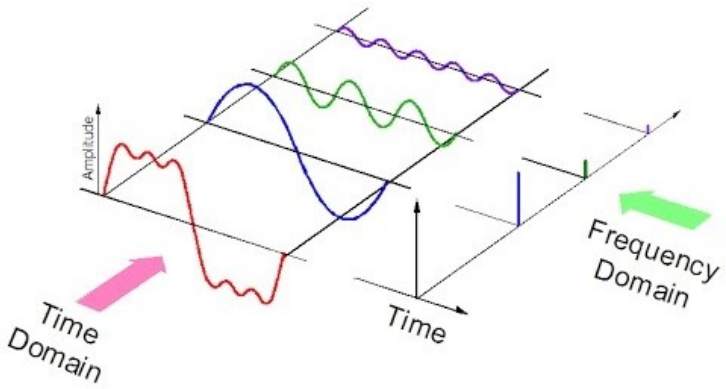


Figure 2.2: Graphical representation of frequency and time domains

Another basic example of the use of the Fourier transform is the one that follows: let us consider a signal $S(t)$ sampled at a sampling frequency $F_s = 1000\,Hz$ composed of the sum of three other sine signals of frequencies $20\,Hz$, $60\,Hz$ and $85\,Hz$

$$S(t) = 0.7\sin(2\pi \cdot 20 \cdot t) + \sin(2\pi \cdot 60 \cdot t) + \sin(2\pi \cdot 85 \cdot t) \tag{2.5}$$

as figure 2.4 shows, the three sine waves frequencies are conceived when added to each other. In a real case though, there generally is a random noise added to the signal, which results in an even less clear time dependent signal shown in picture 2.5. It is evident that after adding the random noise to the three sine waves it is impossible to understand which frequencies compose the signal $S(t)$. In order to obtain the frequency spectrum of the signal it is required to perform its Fourier transform, which reveals the frequencies of the periodic waves composing the signal (if any).

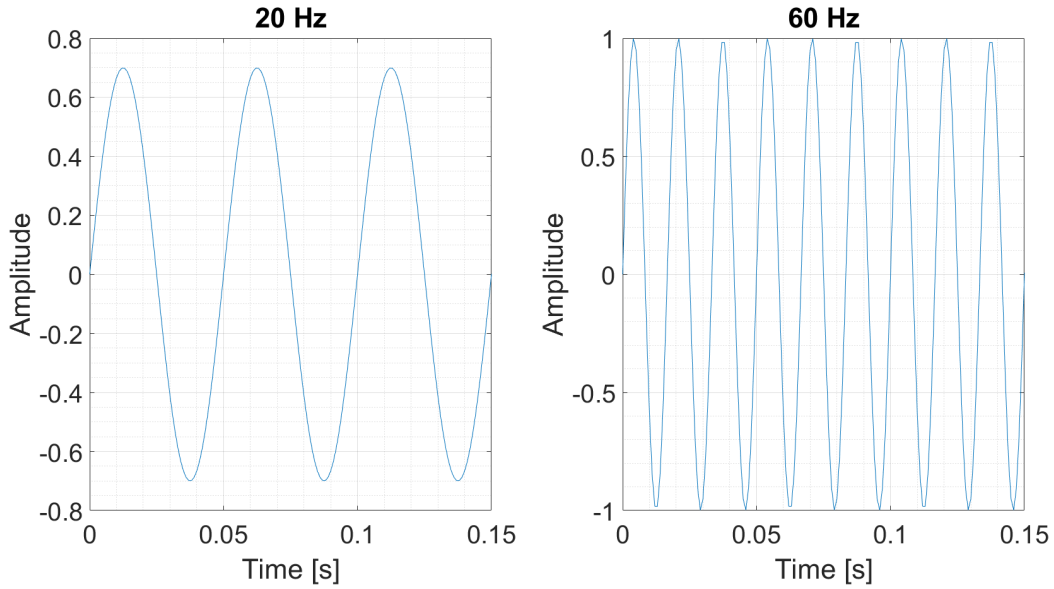


Figure 2.3: Sine waves composing the signal with frequencies of 20 and 60 Hz

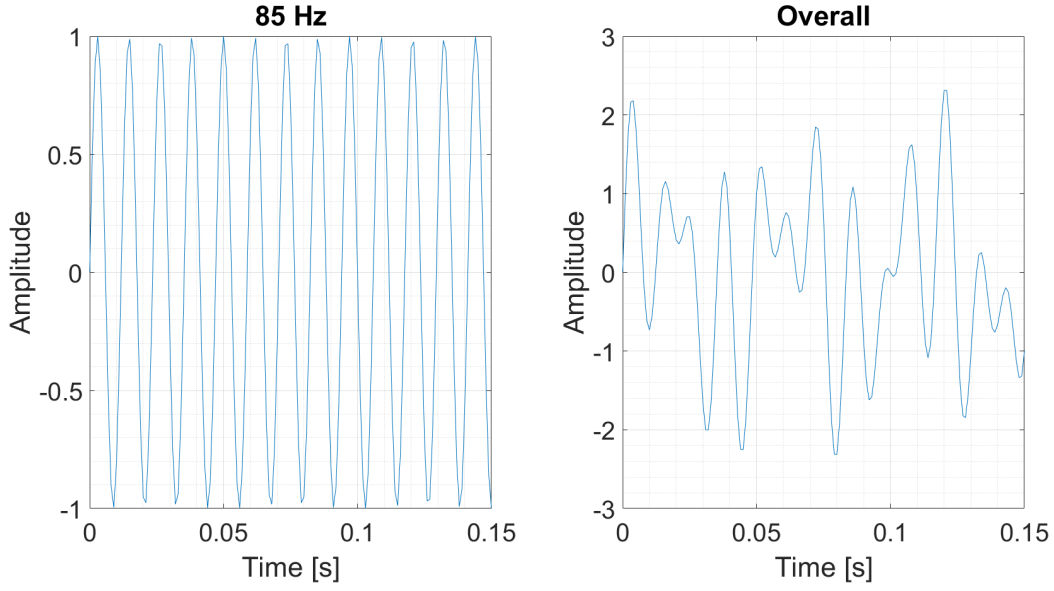


Figure 2.4: Sine wave composing the signal with frequency of 85 Hz and overall signal

In figure 2.6 the spectrum of the signal $S(t)$ is shown. It is obtained with a Fast Fourier

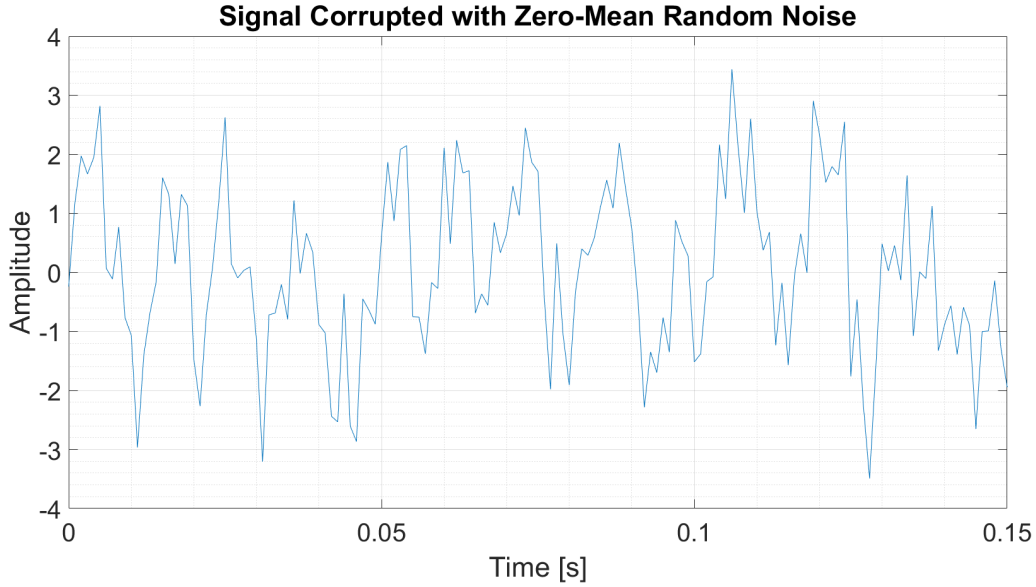


Figure 2.5: Signal with added random noise

Transform (FFT) which is a particular algorithm, usually employed by calculators, executing the Fourier transform. In general a signal has both a positive and negative frequency spectrum which are symmetrical with respect to the zero frequency value. Displaying both sides is redundant, hence usually only the positive side is displayed. In a two sided spectrum though half the energy is displayed in the positive side and half in the negative. Therefore to generate a single sided spectrum the negative side has to be discarded, while the positive side values have to be multiplied by two (except for the zero frequency element), for more information on this matter see [3].
Thanks to the signal spectrum it is evident that there are three frequencies which mostly contribute to the signal amplitude. These frequencies are, as expected, 20, 60, and 85 Hz. It is interesting to note that the spectrum identifies the correct amplitudes of the first two sine waves (0.7 and 1 from equation 2.5), but not of the one at 85 Hz. This error is due to the so-called spectral leakage, which will be explained later on. The Fourier

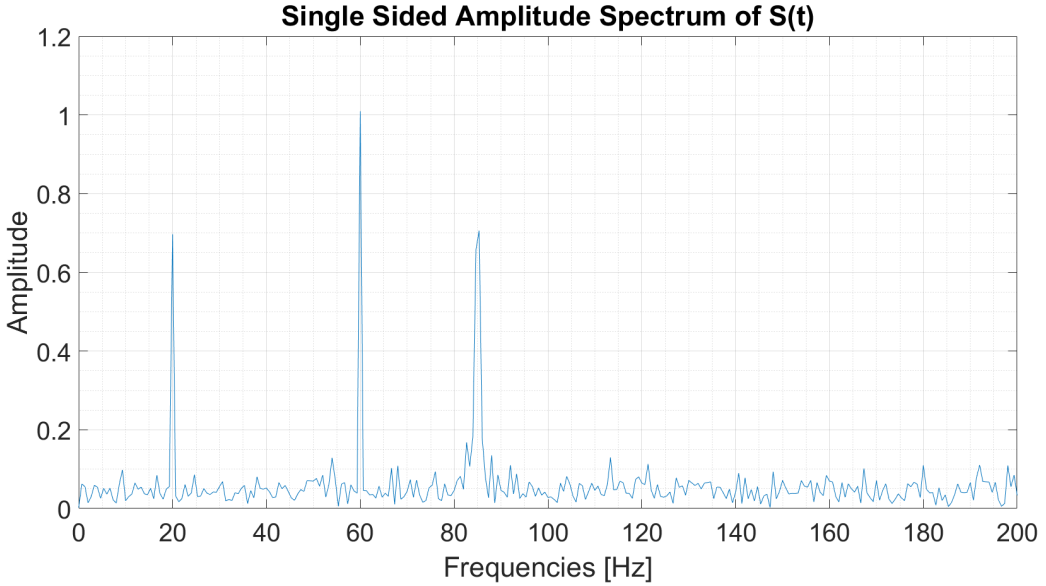


Figure 2.6: Spectrum of the signal $S(t)$

transform varies according to the signal type, in general, given a time dependent signal

$x(t)$, its frequency spectrum is $X(f)$ and is defined as:

$$X(f) = \int_{-\infty}^{+\infty} x(t)e^{-j2\pi ft}dt \tag{2.6}$$

in which the exponential represents the frequency f contribution to the signal $S(t)$. This formulation is valid for continuous signals which extend for an infinite time period.
The inverse operation is the inverse transform:

$$x(t) = \int_{-\infty}^{+\infty} X(f)e^{j2\pi ft}df \tag{2.7}$$

which allows to represent a continuous signal as the sum of an infinite number of periodic exponential weighted by $X(f)df$. Since $X(f)$ is usually complex, it is described by an amplitude and a phase spectrum.
Whenever it is required to perform a signal analysis with a calculator, it is impossible to maintain the continuity hypothesis of the signal because this would result in an integration over an infinite time interval. Thus, the signal has to be discretized into a finite number of values both in the time and frequency domain. In order to achieve such a task the signal generally gets truncated into small time intervals, this operation transforms the integral of equation 2.6 into a summation of finite terms. It is evident that the discretization will introduce an error in the signal. Once the signal is not continuous it is possible for a calculator to apply the Discrete Fourier Transform (DFT) which usually grants a sufficiently accurate approximation of the Fourier transform:

$$X(kf_0) = \frac{1}{N}\sum_{n=0}^{N-1} x(nT_0)e^{-\frac{j2\pi kn}{N}} \tag{2.8}$$

while the inverse transform is:

$$x(nT_0) = \sum_{n=0}^{N-1} X(kf_0)e^{-\frac{j2\pi kn}{N}} \tag{2.9}$$

2.2 Signal preprocessing

Before applying the DFT the analog signal has to undergo the following operations:

1. sampling
2. truncation
3. frequency discretization

2.2.1 Sampling

Sampling a continuous analog signal $x(t)$ means measuring its values $x(iT_s)$ at given time instants (iT_s), with a time interval between two measurements being defined by the sampling frequency F_s, which specifies how many samples of signal are measured each recording second. To study how sampling works, and its fundamental properties, it is useful to refer to the ideal case in which sampling is carried out by an impulse train.
In figure 2.7 a generic continuous signal $x(t)$ is represented with frequency spectrum $X(f)$ limited up to frequency f_m. That signal is ideally sampled multiplying it by an impulse train $s(t)$ shown in figure 2.8, whose frequency spectrum is shown in figure 2.9, the single

pulses are separated by F_s frequencies. The impulse train in the time domain is defined as:

$$s(t) = \sum_{i=-\infty}^{+\infty} \delta(t - iT_s) \tag{2.10}$$

while in the frequency domain:

$$S(f) = f_s \sum_{k=-\infty}^{+\infty} \delta(f - kf_s) \tag{2.11}$$

the sampling is obtained by multiplication of the signal by the temporal train impulse in

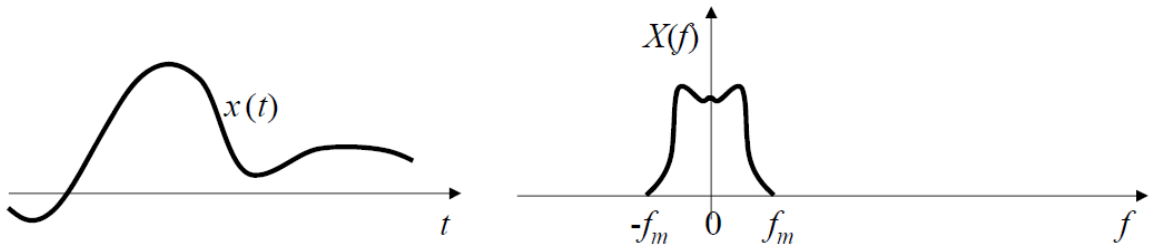


Figure 2.7: Generic signal with a limited bandwidth

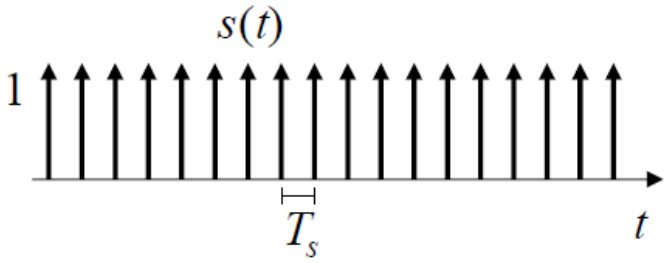


Figure 2.8: Pulse train in time domain

the time domain.

$$x_s(t) = x(t) \cdot s(t) = x(t) \sum_{i=-\infty}^{+\infty} x(iT_s)\delta(t - iT_s) \tag{2.12}$$

x_s is shown in figure 2.10 while in the frequency domain a convolution between $X(f)$ and $S(f)$ is required.

$$X_s = X(f) * S(f) = X(f) * f_s \sum_{k=-\infty}^{+\infty} \delta(f - kf_s) = f_s \sum_{k=-\infty}^{+\infty} X(f - kf_s) \tag{2.13}$$

The result is shown in figure 2.10 and 2.12. The sampled signal frequency spectrum is composed of repetitions of the original signal spectrum $X(f)$, offset to multiple of the sampling frequency F_S. In order to avoid intersections between the spectrum repetitions it is clear that the time separating them has to be equal or bigger than $2f_m$. In other words that condition requires the sampling frequency F_s to be greater than, or equal, to twice the highest frequency for signal with limited bandwidth, this condition is known as the Nyquist condition. When such condition is violated, frequency components above

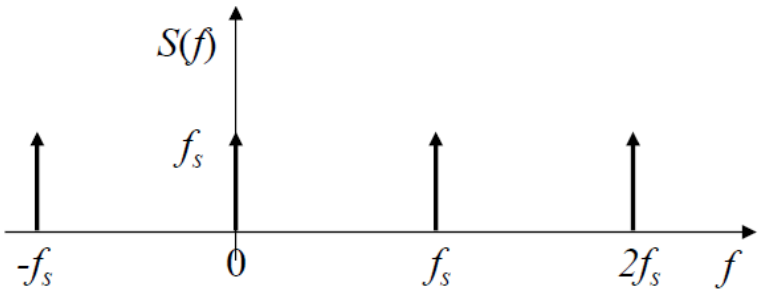


Figure 2.9: Pulse train in frequency domain

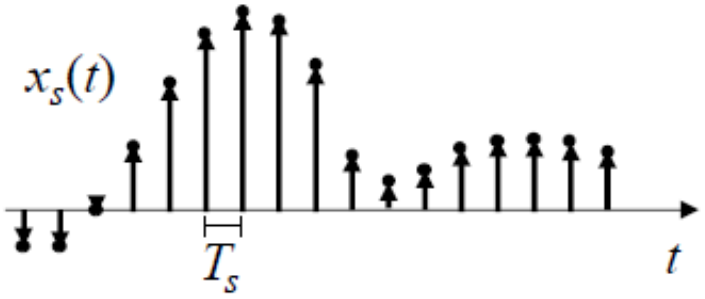


Figure 2.10: Sampled signal in time domain

half the sampling frequency appear as frequency components below half the sampling frequency, resulting in an erroneous representation of the signal.

$$f_s = \frac{1}{T_s} \geqslant 2f_m \tag{2.14}$$

Usually, in order to avoid aliasing, the analogue signal is filtered by means of an antialiasing filter with a cutoff frequency equal to $F_s/2$ before passing to the analogue-to-digital converter, as shown in figure 2.13. The filter is a lowpass whose purpose is to remove any possible frequency beyond Nyquist. Due to the fact that it is impossible to design a filter whose frequency response does not include a transient band of frequencies not properly filtered, the practical maximum frequency is considered to be:

$$f_m = \frac{F_s}{2.56} \tag{2.15}$$

to include the effect of the transient band [6]. In other words this means that a sinusoidal signal requires at least 2.56 samples not to be affected by aliasing.
Whenever the analogue data are not filtered by means of an antialiasing filter, Nyquist condition assumes a slightly different meaning: instead of posing a limit value for the F_s, it defines the highest frequency not affected by aliasing.
Whenever Nyquist condition is not valid aliasing occurs which might lead to erroneous frequency spectrum representation similar to figure 2.14.

2.2.2 Truncation and windowing

Referring to a real sampling process, the samples sequence both is discrete and has a beginning and an end, thus there will be a finite number of samples composing the signal. Despite the signal being finite, for both stationary and non stationary signals, DFT algorithms require the input signal to be divided into a discrete number of portions, in order

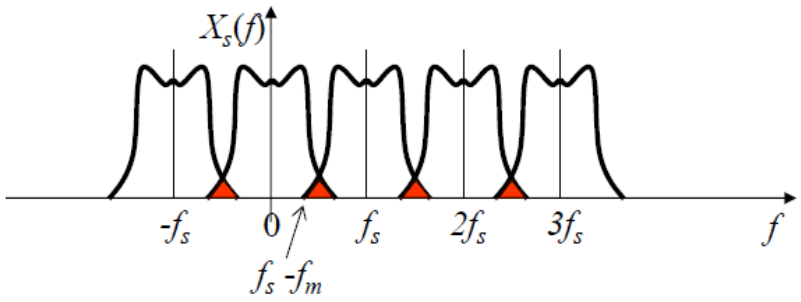


Figure 2.11: Signal spectrum failing to respect Nyquist condition

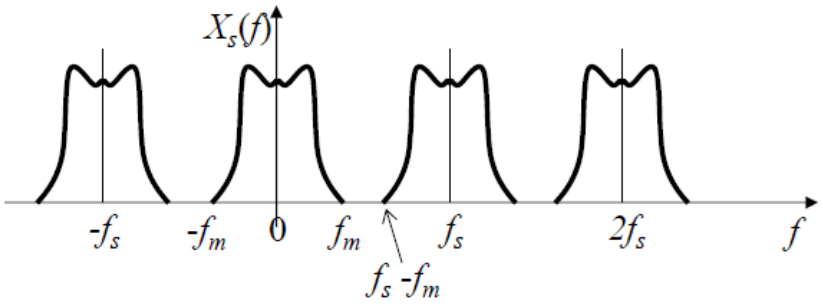


Figure 2.12: Sampling frequency impulses

to do so the signal is truncated at fixed time instants. The time between two truncation instants is constant and is called windowing period, one of the most influential analysis parameters.
Dividing the signal into several segments prior to the frequency spectrum extraction introduces an error known as spectral leakage. This phenomenon is the result of the assumption in the DFT algorithm that the signal record is periodic over a period equal to the windowing period. If the signal segment has a non-integral number of cycles within its windowing period, this assumption is violated and spectral leakage occurs. The phenomenon causes energy to be assigned not just to the right frequency, but even to the adjacent ones. Generally speaking every frequency spectrum is affected by spectral leakage to some extent.
There are only two cases that completely avoid spectral leakage. The first one is if the user is sampling synchronously with respect to the signal, in which case it would be feasible to include an integral number of cycles within a time segment. The other case consist in capturing a transient signal that fits entirely in the time interval. In most real life cases, however, the signal to be analyzed is unknown and can be considered stationary with respect to the time segment, which means that the signal is present before and after the sampling, and cannot be defined as transient with respect to the windowing period. Hence avoiding spectral leakage entirely is impossible.
Despite the impossibility of avoiding spectral leakage, there is a way to effectively reduce its influence, which is applying a windowing function to the truncated signal.
A generic windowing function $w(t)$ is a time dependent function which multiplies the time dependent signal segment.

$$x_w(t) = x(t) \cdot w(t) \tag{2.16}$$

While in the frequency domain applying a windowing function means performing a convolution between the signal transform $X(f)$ and the windowing function transform $W(f)$.

$$X_w(f) = X(f) * W(f) \tag{2.17}$$

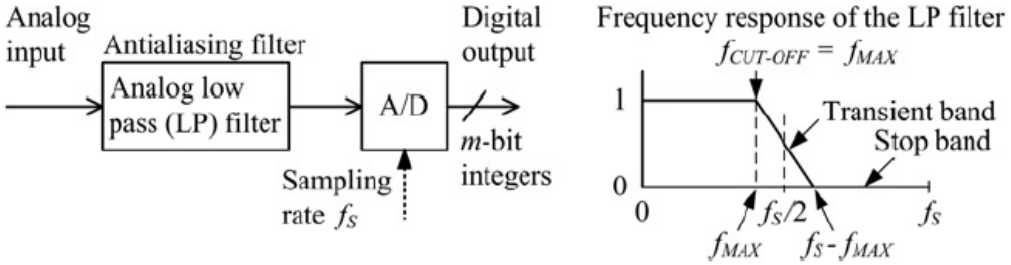


Figure 2.13: Antialising filter

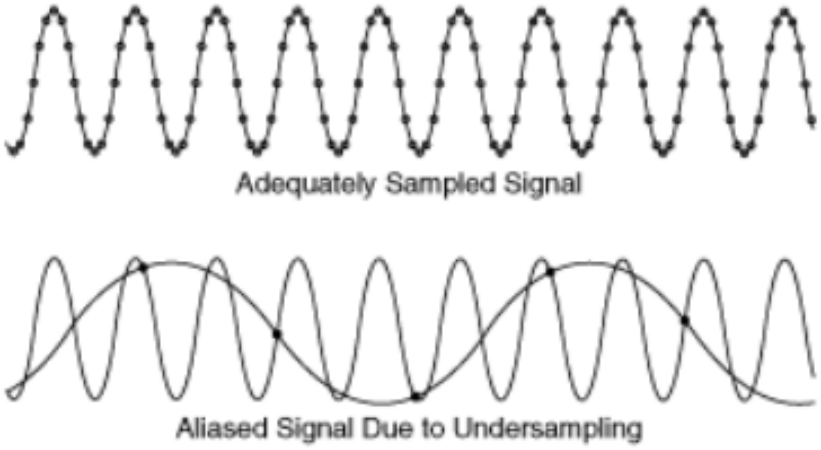


Figure 2.14: Visual representation of aliasing

Applying no windowing function means applying the rectangular window defined as:

$$w(t) = \begin{cases} 1 & \text{for } -\dfrac{T}{2} \leqslant t \leqslant \dfrac{T}{2} \\ 0 & \text{elsewhere} \end{cases} \tag{2.18}$$

The rectangular window does not alleviate spectral leakage, hence usually other windowing functions are used such as Hanning, Hamming, Blackman and FlatTop shown in figure 2.15, which minimize the error by canceling the signal at the time instants near the beginning and the end of the segment. So for every segment the central part gives more contribution to the signal. It is important to remember that windowing functions tend to to improve significantly either amplitude or frequency precision, not both, some functions (like Hanning) compromise between the two and allow for a decent improvement of both. The continuous windowing function $w(t)$ must be discretized to apply it to the signal which is usually done considering the same time interval $T = 1/F_s$ between two signal values defined by the sampling frequency F_s. Hence the function becomes dependent only on the N points required to discretize the window.
The signal segment must also be multiplied by a coherent gain (different for each windowing function) after being windowed in order to depict the correct frequency spectrum amplitude, every windowing function has a different coherent gain.

$$w(n) = \frac{w^*(n)}{Coherent\,Gain} \tag{2.19}$$

To better understand how applying a windowing function to the truncated signal can improve spectrum precision, let us introduce an example. In figure 2.16 an example is shown about windowing a 5 Hz sine signal with amplitude of 2, sampling frequency of

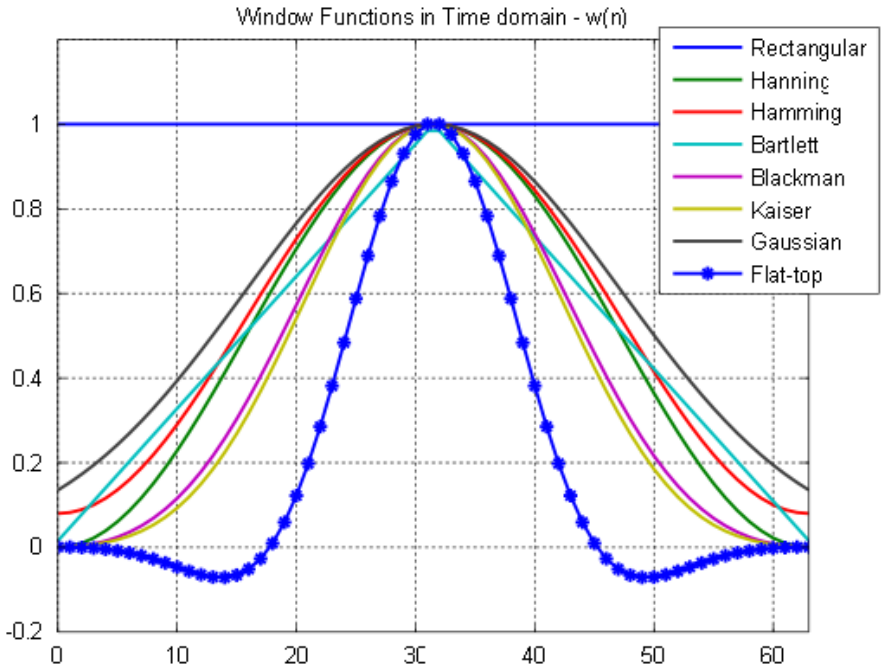


Figure 2.15: Windowing functions time domains

1000 Hz, and windowing period of 1.5 seconds, in the middle figure in red the Hanning window is graphed and in blue the result of $x(t) \cdot w(t)$, on the right plot it is possible to see that the blue spectrum (representing the windowed signal spectrum), which has a $\Delta f = 1/T_w = 2/3 = 0.66$ Hz, is narrower, meaning that its spectrum distributes energy to fewer frequency bins as it should be considering that the only frequency present in the spectrum is 5 Hz. Also its amplitude of about 1.7 is closer to the actual value of 2 than the rectangular window case. It is important to notice that in this case the 5 Hz signal does not show an integer amount of cycles within the windowing period, since $5 \cdot 1.5 = 7.5$. To highlight the influence of having a signal periodic over the windowing period, let us now consider a sine signal at 4 Hz. This time the signal is periodic over the windowing period since $4 \cdot 1.5 = 6$, thus truncation does not introduce spectral leakage. Observing figure 2.17, the right most plot shows that the most accurate spectrum is the one obtained by the rectangular window. As a matter of fact, despite the amplitude error being null, the rectangular window cuts off unwanted frequencies more steeply than Hanning window, which results in a narrower rectangular window spectrum.
This difference is due to the nature of the windowing process. As a matter of fact, by reducing the amplitude of the signal at time instants close to the edge of the windowing period, the frequency content of the signal segment changes slightly from the original segment. This operation introduces a small spectral leakage itself, and, as a result, in case truncation does not introduce its own spectral leakage, such as for the 4 Hz signal case, the Hanning function yields a worse spectrum than the rectangular function. The difference between the two spectra is, however, negligible since they both capture the exact amplitude and the one obtained with the Hanning window is only slightly broader than the one obtained with rectangular window.
The difference is not negligible in case the truncation introduces spectral leakage, such as the case of figure 2.16, in which the spectrum of the Hanning windowed signal is far more precise than the one resulting from a rectangular windowing (or no windowing). For this reason it is usually a good practice to apply a windowing function to truncated signals.

It is important to remember that in case truncation introduces a spectral leakage, windowing the signal only reduces the precision loss, the error due to truncation will always be present no matter the windowing function. As previously mentioned, several windowing functions exist, they all have different features, but the main trade off between them concerns having a steeper neighboring frequencies attenuation (such as the rectangular window, Hanning and Hamming) or preserving the amplitude value of the original signal (such as the FlatTop). The first kind of functions allow for a more accurate frequency representation, while the second one results in minimal amplitude error. Hence, windowing function choice is strongly dependent on the analysis aim, and no 'right' windowing function exists. More information concerning the windowing functions can be found in literature, such as [4], and [3].

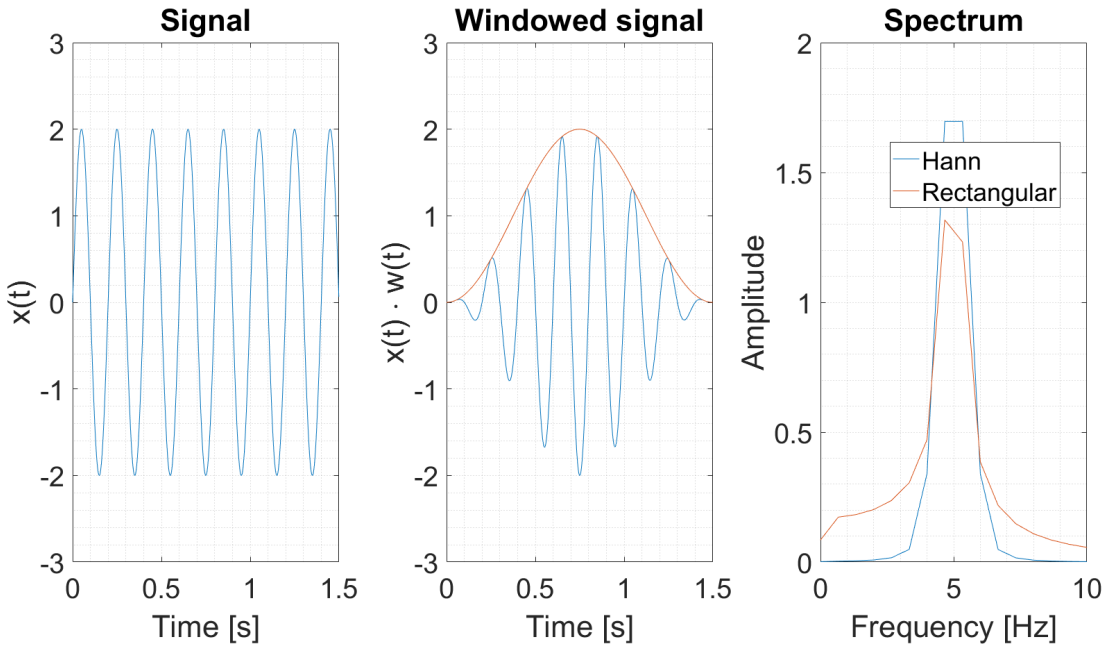


Figure 2.16: Windowed 5 Hz sine signal and its spectrum with $\Delta f = 2/3$ Hz

As previously mentioned, windowing a signal inherently cancels the contribution of the side portions of the segment to the signal, therefore it is required to overlap segments in order not to lose pieces of signal segments. As a matter of fact analyzing a non-stationary signal, meaning that its frequency contribution changes over time, requires the signal separation into small segments so that a stationary approximation can be valid. The smaller the segment the smaller the frequency variation the closer to reality the approximation will be.
As shown in figure 2.18 no segment contribution is neglected through the process. The amount of overlapping required depends on the windowing function applied, for example relatively large windows in the time domain (such as Hanning) require overlap of about 50 %, while for narrow windows (like FlatTop) higher overlap values are required.
When choosing the most appropriate overlap it is important to balance the contribution of two factor to the signal analysis: computational cost and the frequency content rate of change over time. Having high overlap on one hand grants an almost stationary signal segment, on the other it strongly raises the computational cost of the analysis. For every windowing function an optimum overlap value exists that keeps into account the computational weight of the process, and the need to avoid signal losses. Those values are usually selected.

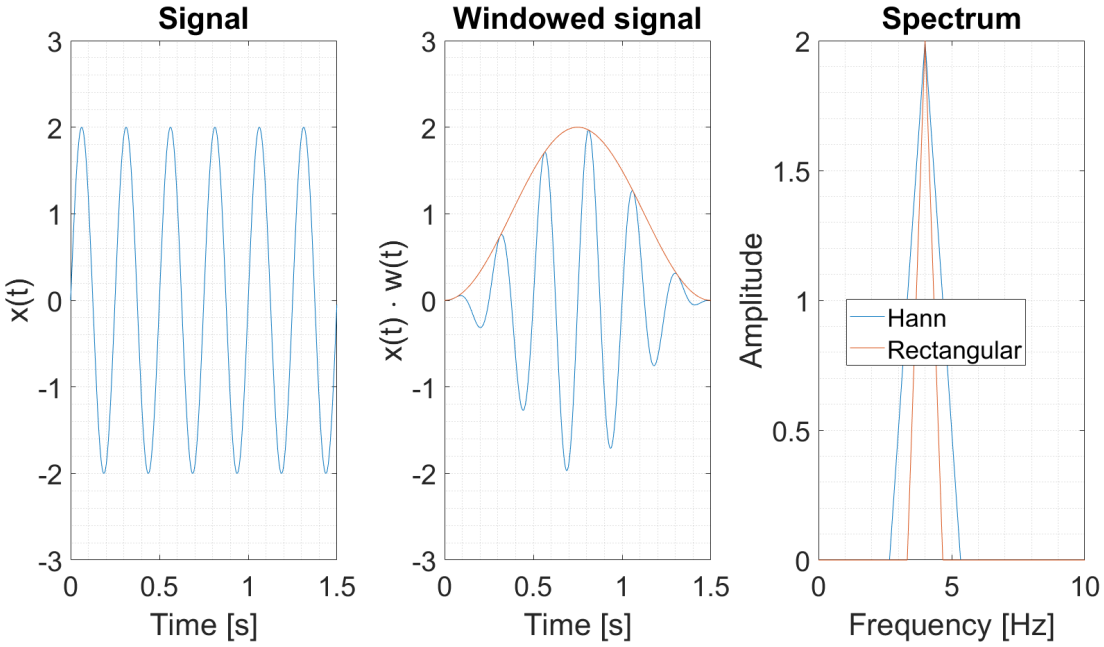


Figure 2.17: Windowed 4 Hz sine signal and its spectrum with $\Delta f = 2/3$ Hz

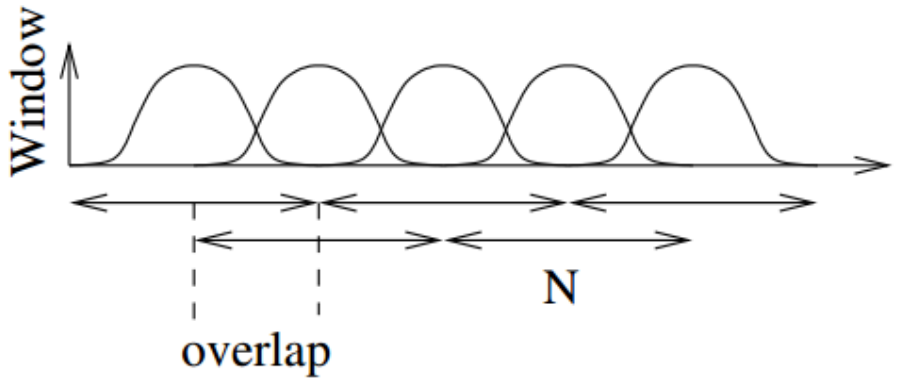


Figure 2.18: Overlapped signal

2.2.3 Frequency discretization

The result of the FFT is a discrete frequency spectrum which by its nature cannot show every possible frequency value between the minimum and the maximum frequency considered. The discrete frequency values are called bins. The discretization of the frequency domain generates the so called picket fence effect, the name is due to the fact that the frequency spectrum (which should be continuous) is represented through the pickets of a fence, as shown in figure 2.19. The dots represent the plotted spectrum values, while the line draws the real signal spectrum amplitudes. The figure highlights the fact that by discretizing the frequency domain some peaks are not properly captured. It is therefore clear that transforming the spectrum from continuous to discrete introduces an error. For example in the right plot of figure 2.16 part of the reason why the spectrum amplitude does not reach the correct value of 2 is that the frequency 5 Hz lies exactly in the middle of two frequency bins. As previously stated, this error represents a form of spectral leakage introduced by the discretization of the frequency domain.
The values of the frequency spectrum are multiples of the frequency resolution Δf, if the

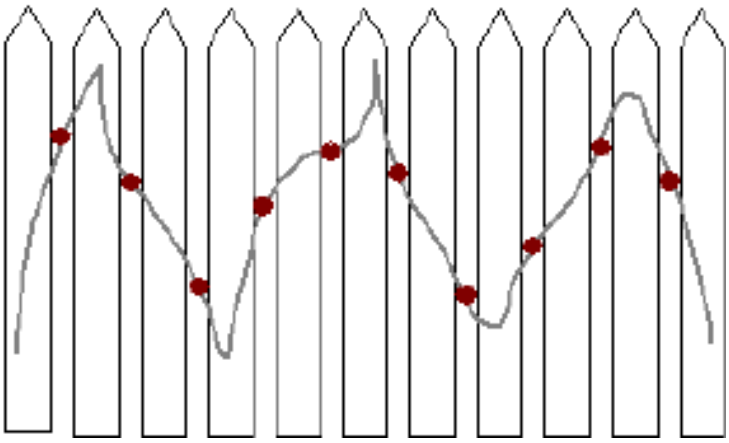


Figure 2.19: Visual representation of the picket fence effect

signal spectrum contains frequencies which are not integer multiples of Δf their representation will be affected by an error concerning both frequency and amplitude. The picket fence effect error is heavily dependent on the windowing functions, applying no windowing (or applying a rectangle windowing function) results in the biggest error among the various functions. Another factor greatly influencing the picket fence effect error is the distance between the frequency to represent and the two closest frequencies bins.
In order to quantify the error 4 parameters shown in figure 2.20 are introduced:

- ΔdB is the amplitude difference between the two highest peaks adjacent to the maximum, this represents the distance between the real frequency and the two closest bins. As a matter of fact a $\Delta dB = 0$ means that two neighboring bins have the same amplitude, which happens whenever the true frequency lies exactly in the middle of the two bins and its energy is equally shared between the two closest bins. If the true frequency was to be closer to one of the two bins, it would share more energy to the closest one, allowing for a $\Delta dB \neq 0$. The more the ΔdB is high, the less likely is that a true frequency is sharing energy among several bins
- ΔL represents the amplitude difference between the real maximum and the maximum obtained with the FFT
- Δf represents the difference from the real frequency and the closest bin
- f_0 is the FFT frequency resolution

In figure 2.21 are plotted the various windows errors caused by picket fence effect concerning frequency ($\Delta f/f_0$) and amplitude (ΔL). The errors are graphed as function of ΔdB, a null value of such parameter indicates that the frequency depicted is exactly in the middle of two bins, such condition represents the worst case since it results in the highest possible picket fence error for every kind of window. In particular it is easily noted that the rectangle window shows the worst behavior among the other windows given the same ΔdB for both frequency and amplitude. Concerning the FlatTop window, it seems to display a very low ΔL and $\Delta f/f_0$ for nearly any value of ΔdB, in particular the ΔL always is nearly null this allows to clearly display frequency content, but also means that adjacent frequencies will have similar amplitudes. These features make the FlatTop window less suitable to distinguish the contribution of frequencies near the one of interest. As stated in [4], to perform a successful analysis some parameters have to be set which are directly related to each other.

- windows frequency $f_0 = \dfrac{1}{T}$

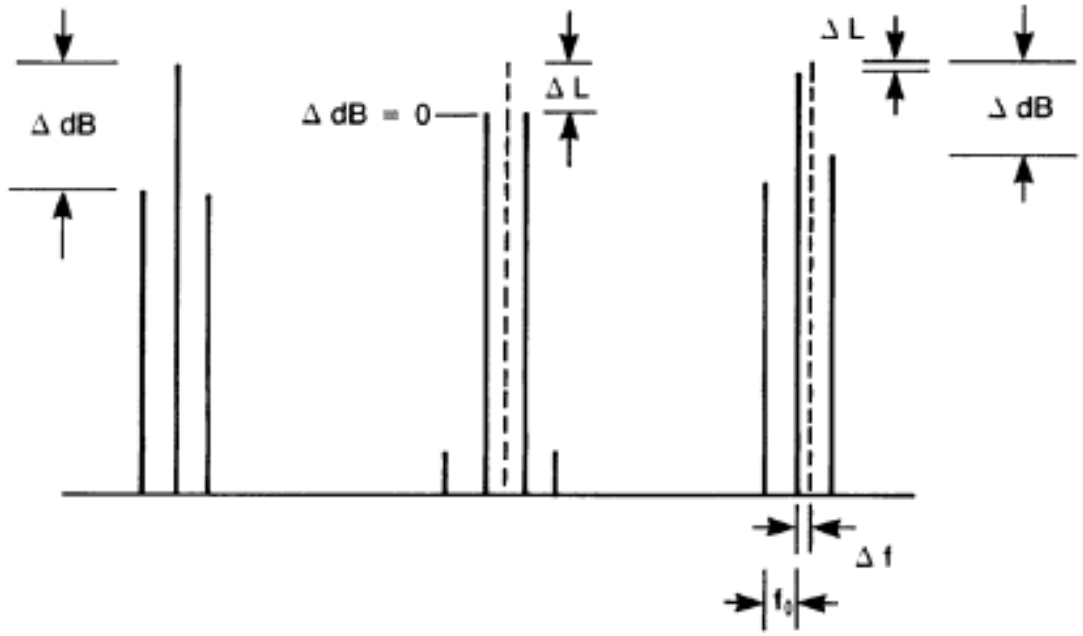


Figure 2.20: Picket fence effect errors of different windows

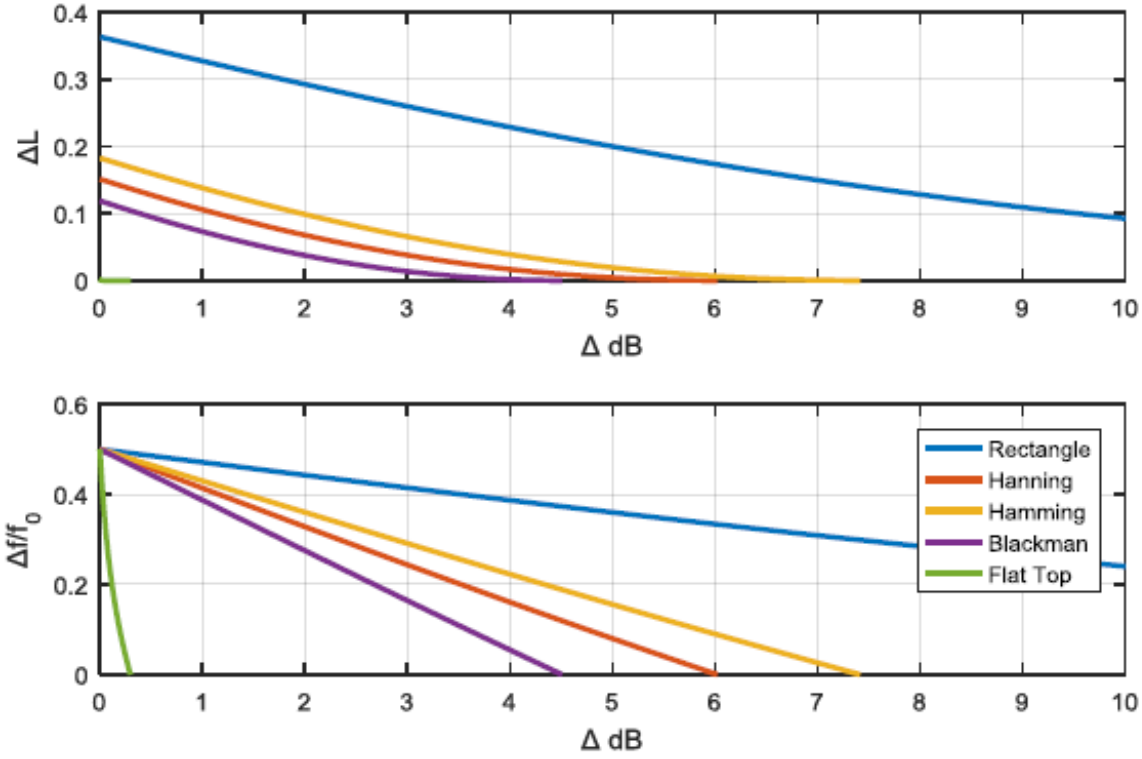


Figure 2.21: Picket fence effect errors of different windows

- sampling frequency $F_s = Nf_0$
- windowing period $T = NT_0$
- sampling period, time between two sampling $T_0 = \dfrac{1}{F_s}$

- number of samples in a window $N = \dfrac{T}{T_0}$

As shown in the relations above the various parameters are not independent, as a matter of fact only two have to be set, the other are derived. In general the sampling frequency cannot be changed and depends on the test run. Which leaves for only one parameter to be set which is the windowing period T. It is important to remember that there are more parameters to set than the ones listed above (e.g: overlap, windowing function), whose values do not directly influence other parameters. The choice of an appropriate value for T revolves around whether the user wants to achieve a low frequency resolution or a low amplitude error. As a matter of fact, having a short windowing period allows for the signal segments to be considered almost stationary, resulting in a low amplitude error. On the other hand, considering that the frequency resolution is defined as follows:

$$\Delta f = \frac{1}{T} = \frac{1}{NT_0} = \frac{F_s}{N} \tag{2.20}$$

a small windowing period value will cause a big Δf, resulting in a higher picket fence error.
There is not a correct choice to be made concerning the windowing period, the parameters all depend on the features of the signal to be analyzed, and on the aim of the user.

Chapter 3

Frequency Analysis

Frequency analysis is usually applied to rotating machinery (gearboxes, compressors, turbines, pumps, etc.) in order to detect possible manufacturing defects, such as misalignments, surface imperfections, mass imbalances and many others. Another usage for frequency analysis is to check the presence of resonance peaks due to the crossings between frequency modes and orders. This allows to understand the fatigue cycles of the machine and to predict its lifetime.

3.1 Frequency analysis code implementation

The aim of this work has been to write a MATLAB code able to perform both frequency and order analysis, although in this section only the code concerning frequency analysis will be considered.
Before launching an analysis, the user has to specify inputs such as:

- the directories concerning the raw data, and the position where output figures will be saved
- speed and order table. The first holds information about ratios between the various useful machine speeds (i.e.: the various gearbox shafts speeds). At least one of which must be read from the tachometer channel of the raw data file. The order table indicates which orders are to be extracted
- autoslice table. It specifies the settings for the autoslice function
- output settings. It defines which figures to save and their format

The expected output in the case of frequency analysis is a colormap displaying the frequency spectrum of the signal with respect to time or rotational speed.
In the previous sections the basis of frequency analysis have been explained. Particular focus has been placed on the choice of certain parameters, such as windowing period, windowing functions, overlap, and so on. It is important to add that before performing the frequency analysis, the code automatically sets some analysis parameters, according to the raw data characteristics and the user requests.

3.1.1 Code architecture

The code architecture concerning the processes required to perform the frequency analysis is shown in figure 3.1. The right branch of the the diagram concerns the automatic choice of parameters which will be explained in the next paragraph. The block describing

the frequency analysis consists in the various steps described in the previous paragraphs which include signal truncation, signal windowing, and performing the actual Fast Fourier Transform.

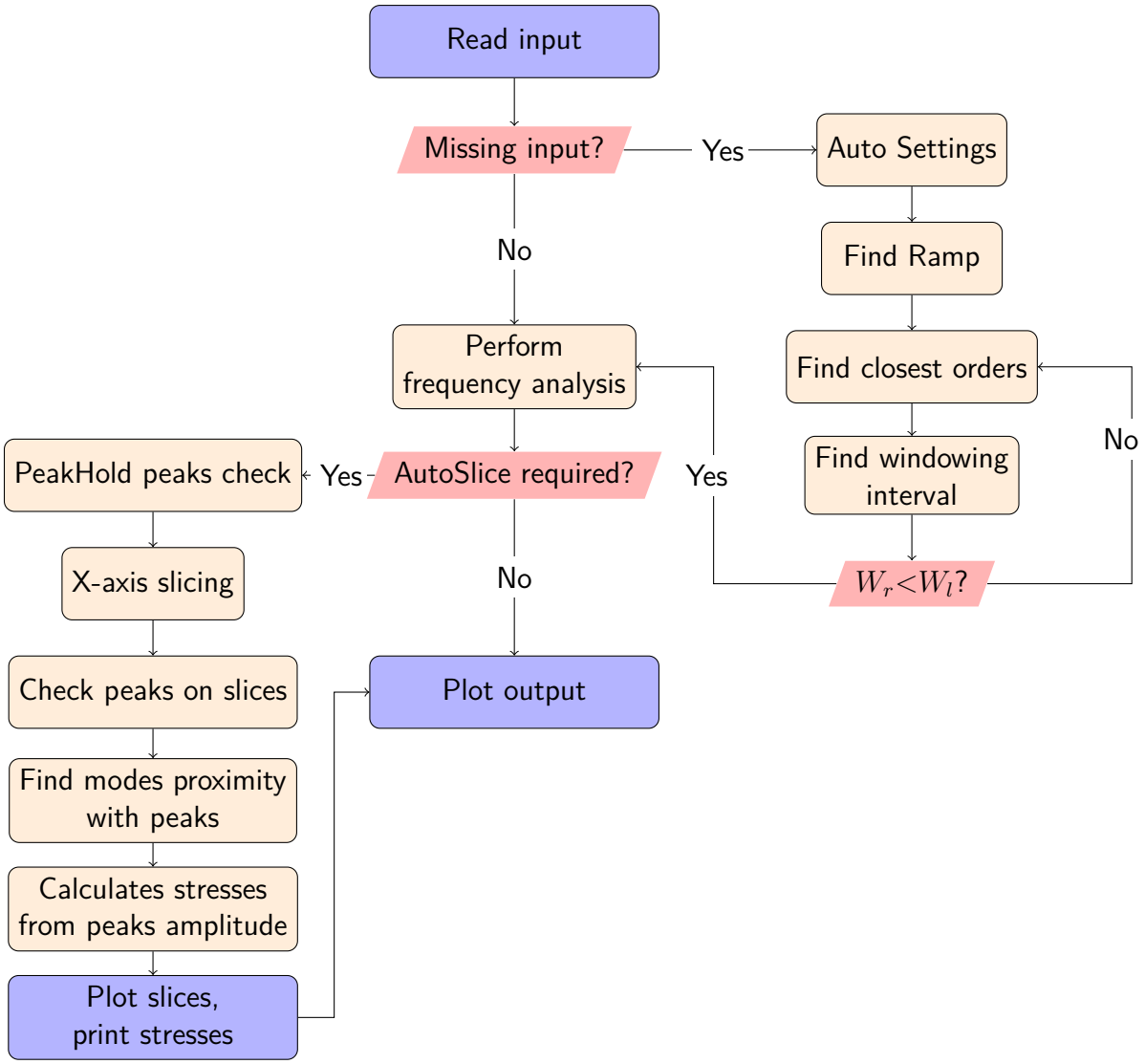


Figure 3.1: Frequency analysis workflow

3.1.2 Code output

In figure 3.1 the last block of the work flow diagram describes the production of the output plot, which is usually called Campbell diagram (i.e. a diagram with speed x-axis and frequency y-axis). Each analysis code function has a different kind of output, in this section the frequency analysis output will be briefly illustrated.
As previously explained the frequency analysis yields the frequency spectrum of the signal, which represents the contribution each frequency gives to the overall signal amplitude. Up to Nyquist frequency, which is equal to $F_s/2.56$, the frequency spectrum will be accurate, while above Nyquist frequency results are affected by aliasing.
Typically the signals to be analyzed are generated by sensors (strain gauges, proximity, accelerometers, etc.) on various parts of the gearbox, which during the test run is driven at different speeds. It is therefore of interest to display the frequency spectrum as a function of speed or time. To sum up, the frequency spectrum, which is already a frequency versus amplitude plot, has to be represented also as a function of either speed or time. To achieve

this kind of representation a 3D visualization is required. The code forces the user to a frequency y-axis, and an amplitude z-axis, while the x-axis can be set to either speed or time. The 3D plot is called colormap as the z-axis is represented by means of colors rather than being geometrical dimension, and is shown in figure 3.6, with a speed x-axis. To represent the amplitude the code generates a NxM matrix, where N is the number of elements in which the frequency y-axis is discretized, and M is the number of elements in which the x-axis is discretized, while the actual values of the matrix elements are the spectrum amplitudes.
Right of the colormap the so-called peakhold is plotted, a graph whose points are the maximum amplitude values for every frequency represented. Below the colormap the amplitude of the requested orders is shown. In order to extract the amplitude value of each order, the dedicated function first calculates the speed of every order at each time instants or each reference speed value according to the x-axis setting, which simply is the reference speed times the order value. Then it converts such speed values from rpm to Hz. This allows to look for the amplitude of specific frequencies in the spectrum. Once those values are found, they are plotted in the diagram below the colormap. The closer two orders are the lower the frequency resolution of the spectrum has to be to allow for a proper extraction. This limitation will be better explained in the following section. The code also produces an overall of the signal, which can be either the RMS or the maximum value among the window elements, where the RMS (root mean square) is defined as:

$$x_{rms} = \sqrt{\frac{1}{n}\sum_{i=1}^{i=n} x_i^2} \tag{3.1}$$

where n is the total number of samples. In figure 3.6 with a yellow line an order is highlighted, while the signal overall is black. From this plot is easy to see peaks due to crossings between modes and gearbox orders, in particular comparing the orders amplitudes with the overall value it is possible to understand whether the plotted order make up for most of the signal amplitude or whether a significant part of the amplitude is held by other orders. In figure 3.6, order highlighted has nearly the same amplitude of the overall, especially near the order peaks, which means that for such speeds that order is the most important amplitude contributor. The overall obviously has many other peaks which are not followed by the highlighted order plot, those peaks are caused by orders not shown in the plot. The figure produced by the code has, however, a callback function triggered by clicking on a legend item which toggles the visibility of every orders plot, allowing the user to cycle through them to associate each order to every peak. There are other callbacks active on the figure, one plots the modes over the colormap using numbers to identify the mode. Also the callbacks associated to zooming, panning and datatip have been modified. The first two have been associated to the same callback which updates the color scale according the the interval of amplitudes shown after zooming or panning, allowing the user to visualize also the smaller peaks, which would not be easily detectable with a logarithmic scale based on the maximum value of the entire colormap. The datatip has been modified to show the amplitude of the selected point, along with the frequency, x-value, and order. Besides, the callback also updates the peakhold to show the peaks of the x-axis zoomed zone, along with the peaks of the whole frequency dominion. By default MATLAB datatip would show x and y axis values and the color value as z, which does not match the actual amplitude value, since the amplitude values are converted into color values by the MATLAB function dedicated to producing colormaps. Besides for the x and y values selected the callback respectively performs a vertical and an horizontal slice and plots them in the peakhold and order plot.
As previously mentioned in this section, the frequency analysis can yield a colormap with

speed x-axis. As shown in figure 3.2, representing a typical speed profile, the same speed Ω_1 is reached by the reference shaft in two different time instants t_1 and t_2. Conse-

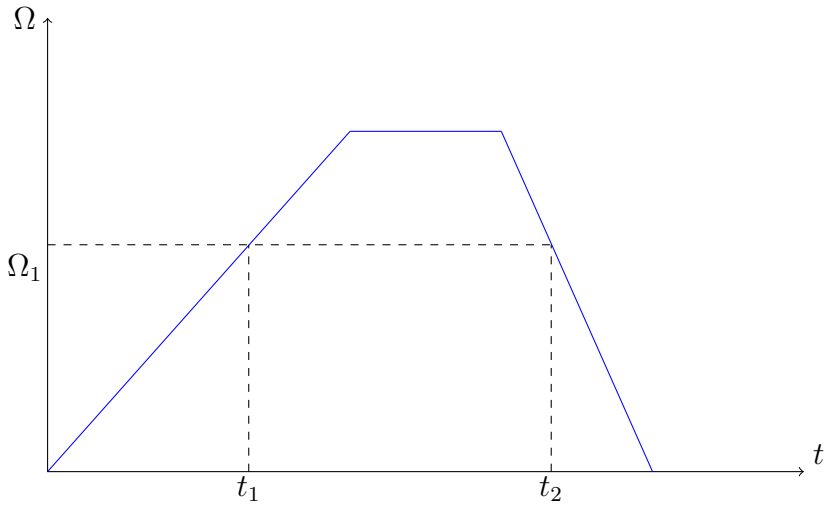


Figure 3.2: Typical ascending and descending run test speed profile

quently, in order to display data with a speed x-axis the code has to rearrange the matrix storing the amplitude values. As a mater of fact the matrix initially has a time referenced x-axis, regardless of the user x-axis request, which means that in the shown example, the amplitude matrix would be referred to the speed profile of figure 3.4. Subsequently, if the x-axis is required to be in the speed format, the code sorts the speed values to which the columns of the amplitude matrix are associated. Due to MATLAB function for plotting the colormap limitations, both x and y axis have to be evenly spaced. Concerning the y-axis this is not a problem since the frequency axis is always evenly spaced with pace equal to $1/T_w$.
The same cannot be said for the speed x-axis. As a matter of fact the speed x-axis values depend on the acceleration at the time span of the windowing period of each window. For example for a window of span 0.5 seconds considering a linear speed ramp of 1 rpm/s the two windows speed values will be spaced by 0.5 rpm (starting from a null speed, the average speed of the first window will be 0.25 rpm, while the second 0.75 rpm); while for the same window if the ramp is 2 rpm/s the first window will have an average speed of 0.5 rpm while the second one will be 1.5 rpm, resulting in a spacing of 1 rpm. That is without overlapping the two windows, which would partially mitigate this difference in spacing. Nevertheless in case of multiple speed ramps having different steepness the uneven spacing will always be present. The main issue with MATLAB default colormap function forcing an evenly spaced x-axis representation is that the amplitude matrix is referred to an unevenly spaced x-axis. Therefore the colormap resulting will depict the amplitude matrix referred to wrong x-values. A possible solution to address this problem is to modify the amplitude matrix, along with the x-axis to be both evenly spaced before generating the colormap.
There are two ways to achieve evenly spacing. The first one is to define a resolution (the parameter $x\,axis\,resolution$). The code then generates an evenly spaced vector starting from the minimum x-axis speed value and with last element being the maximum speed value with pace equal to $x\,axis\,resolution$. To obtain the amplitude values referred to the new x-axis, the code cycles through every new x-axis value, finds the interval of columns in the amplitude matrix within $x_{int} - x\,axis\,resolution/2$ and $x_{int} + x\,axis\,resolution/2$. Among them it finds the maximum value of every row (representing the maximum value of every frequency contribution at the specified speed interval) and it associates it to the

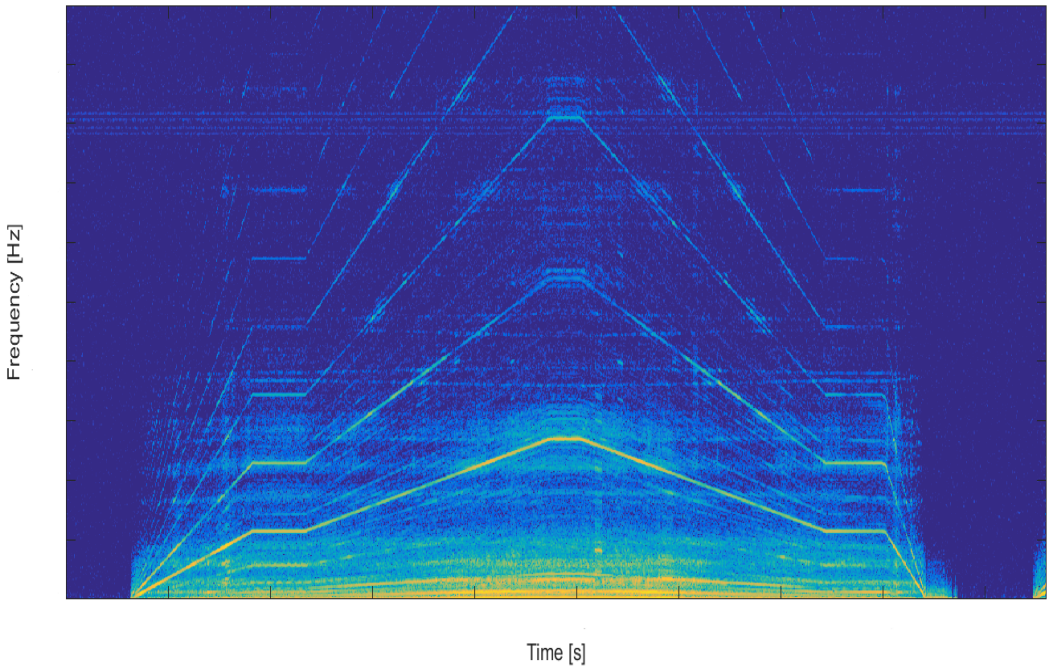


Figure 3.3: Time x-axis colormap

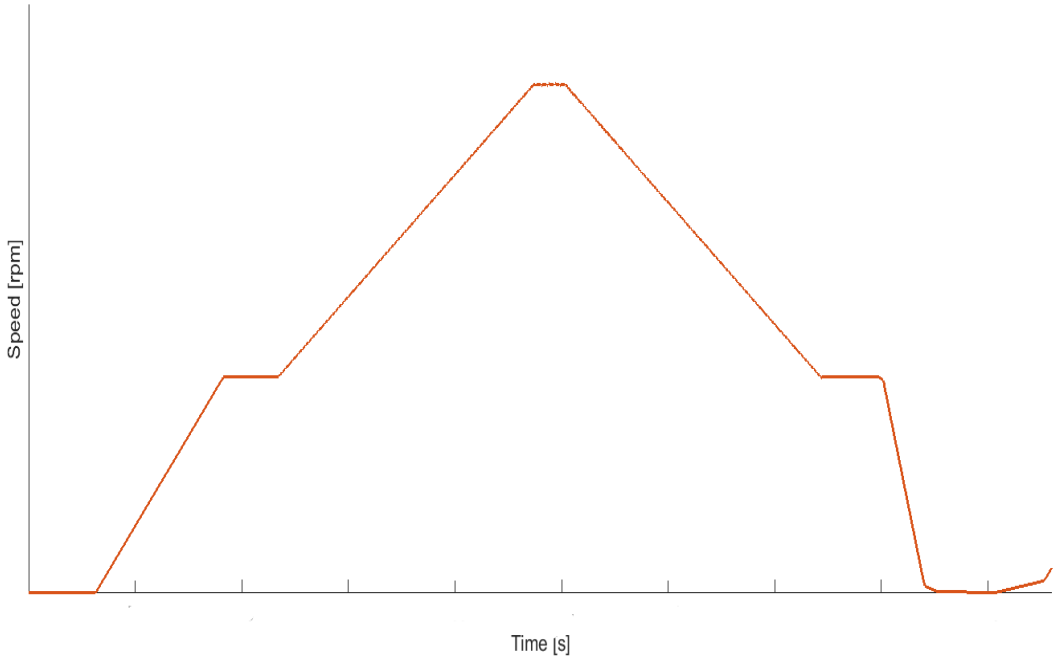


Figure 3.4: Speed profile with time x-axis

new amplitude matrix. Selecting the maximum of every row is just a conservative choice, rather than the maximum the mean value could be chosen, as well as the RMS.
To better visualize this process let's suppose to have the following amplitude matrix, whose columns have already been sorted:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1M} \\ a_{21} & a_{22} & \dots & a_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \dots & a_{NM} \end{bmatrix}$$

the vector x_{ref}, whose length is M, associates to every column the relative speed, while the vector x_{int} is the new speed x-axis. Its dimension is different than x_{ref} since it only depends on the minimum and maximum speed values, and the resolution imposed by the user. For every value of x_{int} the code finds the last value of x_{ref} for which $x_{ref} \leq x_{int,i} - x\,axis\,resolution$, and the first value for which $x_{ref} \geq x_{int,i} + x\,axis\,resolution$. After this values are found, the code crops the amplitude matrix considering only the columns corresponding to the indexes of the values of x_{ref} satisfying the relations just mentioned. Then it finds the maximum value of every row. The result is a column vector of M elements, which is stored and becomes a column of the new amplitude matrix, which can be plotted as function of speed.
The second way to evenly space the amplitude matrix is called matrix stretching and consists in considering the smallest spacing between the x-axis vector and generating a new x-axis vector with that spacing. Subsequently the code cycles through every element of the newly generated x-axis vector and checks its position relatively to the original one, and assigns the value of the closest bigger element of the original vector to the element of the new vector. From the perspective of the amplitude matrix, this means that the columns are duplicated to allow the representation of a variable resolution plot. This operation is shown in figure 4.12, in particular it is possible to see that a new error might be introduced due to this practice dependent on the relative position of the elements of the two vectors and on the dx. In particular some x-axis elements can have wrong amplitudes. The maximum positioning error is dx, which is the smallest difference among the values of the unevenly spaced vector. This value is the speed difference between the two window at the lowest acceleration. This error though is generally negligible.

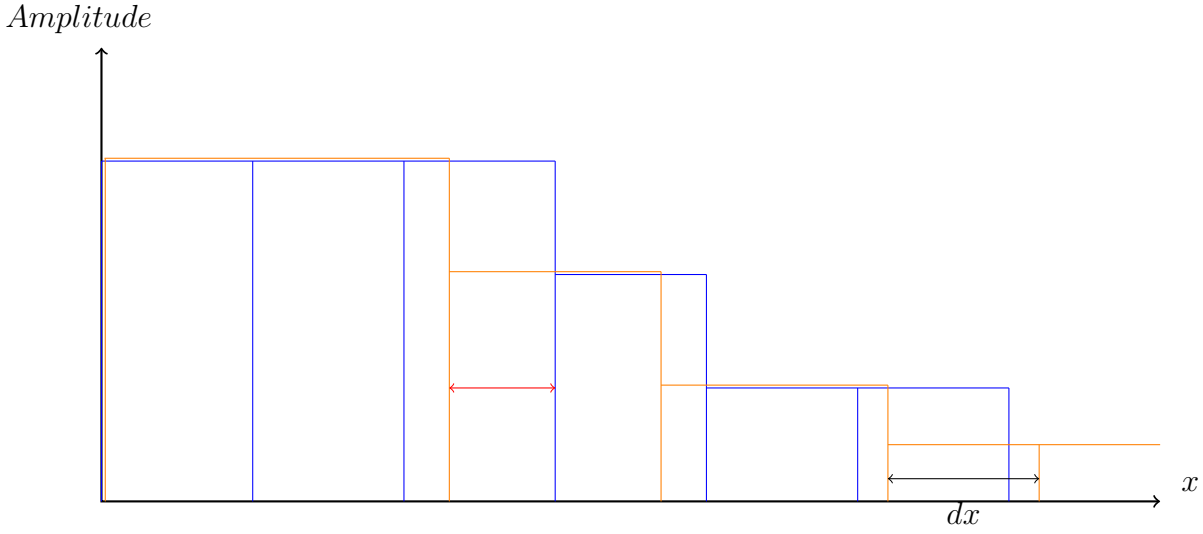


Figure 3.5: In green the original unevenly spaced x-axis vector, in blue the new evenly spaced one. In red the error introduced by this procedure

In case the x-axis is set to be time, the vector representing it is already evenly spaced with spacing equal to $1/F_s$, so no further operations are required before plotting. The colormap having time x-axis is shown in figure 3.3, along with the associated speed profile of the reference shaft in figure 3.4. In the colormap with time x-axis orders are quite evident, since they all follow the same ascending and descending ramps. Even though they are not showed in the figures, the output has the same format as the one for speed x-axis shown in figure 3.6.

3.1.3 Automatic parameters selection

As previously mentioned it is not mandatory to manually define the frequency analysis parameters. The user can either specify some or every parameter or let the code set them according to the specific data to be analyzed. Nevertheless, for some user defined settings the code verifies whether the specified values are compatible with the analysis. The complete list of parameters the code can automatically set is in table 3.1. Selecting the appropriate parameters to obtain a meaningful frequency analysis is not a simple task since it heavily influences the quality of the output. Many parameters depend on each other or on the data being analyzed, and they all depend on the focus of the analysis, which can be either keeping the lowest amplitude error or granting the highest frequency resolution. The focus of the analysis is the first parameter to be set since most of the remaining ones either directly or indirectly depend on it. In case no specification is given for this parameter the code considers the focus to be amplitude precision.
The choice of the following parameters is particularly important:

- windowing function
- overlap
- windowing period
- frequency range

while many of the parameters listed in table 3.1 do not require many operations to be determined, the ones listed above require some computing in order to be set appropriately. Due to their significant influence on the analysis result, it is worth spending some computing time to define them appropriately.
The windowing function is easily set as from Geninatti [5] it is clear that the FlatTop windowing functions allows for better amplitude precision, while Hanning better performs in case the user requires a lower frequency resolution. The code will follow that logic choosing the windowing function. At the moment only these two windows are present in the code as they serve well their purpose, many more could be added in the future in case the need of less biased windowing functions will present. With the choice of the windowing function comes the number of bins, which cannot be picked autonomously since it depends on the windowing function, and represents the window side lobes steepness in the frequency response diagram. In particular the steeper their side lobes attenuation is the fewer bins the window requires to separate two orders. For example, considering the Hanning window frequency response in figure 3.7, the diagram has a fairly narrow main lobe and requires just 3 bins in order to decade from 1 to $-6\,dB$. On the other hand the FlatTop function requires more bins (exactly 5) to perform the same attenuation. The number of bins indicates the minimum number of segments in which a window frequency interval can be discretized, each bin being a discrete value of said interval. Therefore, it also indicates how well a windowing function prevents adjacent frequency components

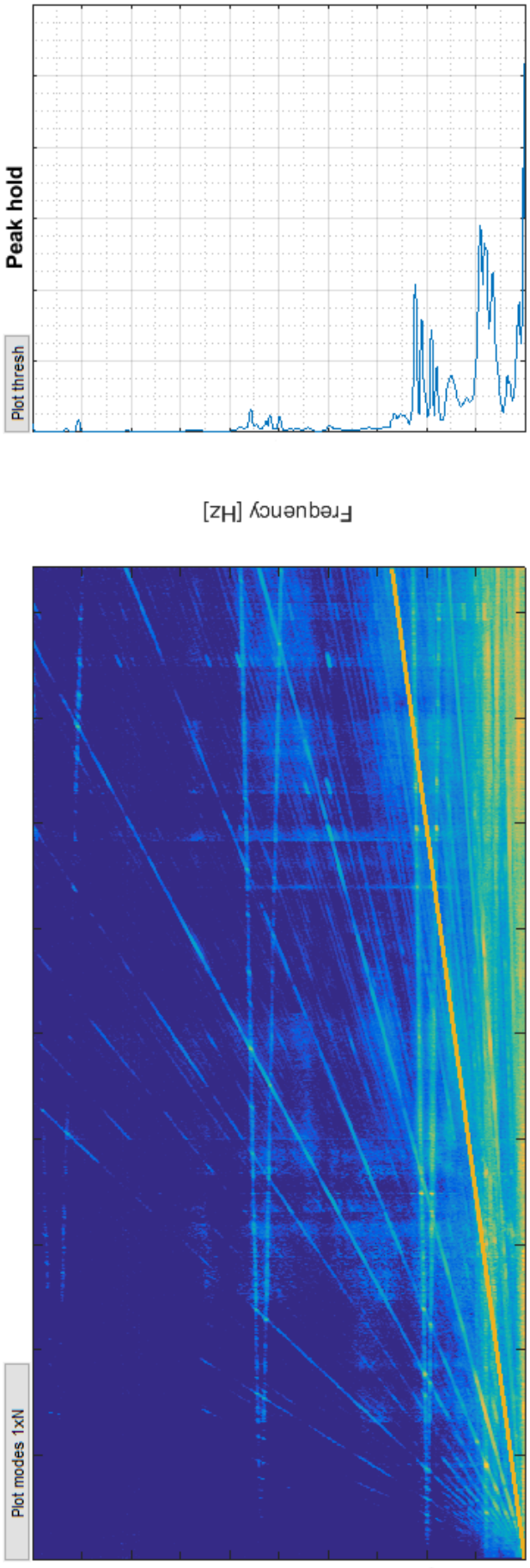


Figure 3.6: Code output with speed x-axis

from leaking into the nearby frequency bins, its value is directly affected by the behavior of the frequency response side lobes of the function. This aspect is extremely relevant as windows with a low number of bins are better suited to separate orders having close frequencies than windows with high number of bins.
Another window parameter showing a great influence on the window behavior is the width of the main lobe, which mainly affects the frequency resolution. As a matter of fact the closer the frequencies to be separated are, the narrower the main lobe has to be to achieve such separation. On the other hand, as the main lobe becomes narrower, spectral leakage increases, thus lowering amplitude precision, and increasing window energy spreading to the neighboring frequencies. This explains why the FlatTop window, which has a wide main lobe if compared to others windowing functions, yields better amplitude precision; while Hanning with its narrow main lobe, allows for better separation of close frequency, hence better spectral resolution. The difference between the two windows main and side lobes are shown in figure 3.7. The leftmost lobe is obviously the main lobe while the narrower ones following on the right are the side lobes.

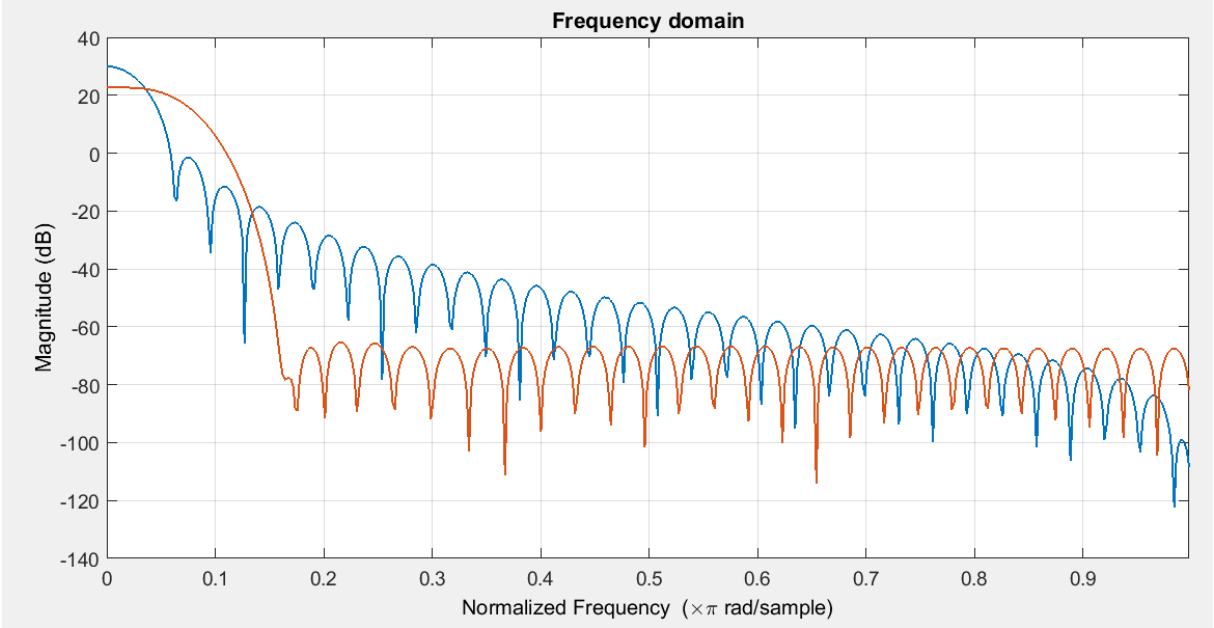


Figure 3.7: Frequency response of hanning window (blue) and FlatTop (orange). The normalized frequency being defined as f/F_s

Once the windowing function is being chosen the interval of frequencies which will compose the colormap frequency spectrum has to be defined. From equation 2.15 defining the Nyquist criterion it is clear that the highest frequency not affected by aliasing will be:

$$f_{max} = \frac{F_s}{2.56} \tag{3.2}$$

the frequency interval depicted in the colormap resulting from the analysis thus will start from zero and end at $F_s/2.56$. It has to be stated that, while verifying an user defined frequency interval, the code allows for the narrower interval defined from zero to $F_s/2$, but emits a warning stating that the highest displayed frequencies might be affected by aliasing. As explained in section 2.2.1, the coefficient 2.56 is in place to take into account the transient of the frequency response of the lowpass filter composing the antialiasing filter, which starts cutting off frequencies higher than $F_s/2$. Hence, to be sure to omit frequencies affected by aliasing, it is a safe practice to consider Nyquist frequency to be $F_s/2.56$.

Defining the previous parameters is a fairly simple task both from an algorithm than from a computational weight stand point, the same cannot be stated about the definition of the windowing period T_w. To select it the code finds the two closest orders to separate and defines an range of potential windowing periods spanning from W_l to W_r. The two windows assume the following meanings:

$\mathbf{W_l}$	$\mathbf{W_r}$
• smallest period granting order separation	• largest window before df/dt makes orders blend
• best amplitude precision	• worst amplitude precision
• worst frequency precision	• best frequency precision
• worst frequency resolution	• best frequency resolution
• best time resolution	• worst time resolution

Table 3.2: Windowing period interval definition

The code will have to pick a windowing period so that the following relation is true.

$$W_l \leq W \leq W_r \tag{3.3}$$

W_l being defined as

$$W_l = \frac{2 \cdot Bins}{\Delta f} \tag{3.4}$$

and represents the smallest of the admissible windowing period. In particular it defines the highest frequency resolution the colormap can have (see equation 2.20), while still being able to distinguish the orders. It, therefore, represents the best fit for an analysis where amplitude error has to be minimized.
In table 3.2, the equation defining the W_l the term Δf represents the minimum frequency difference that the analysis will consider. In other words, orders having frequencies closer than Δf will not be separated properly. Since an order is a multiple of a reference speed the Δf can be written as

$$\Delta f = \Omega_{min} \cdot (O_2 - O_1) \tag{3.5}$$

where Ω_{min} is the rotational speed defining the two closest frequencies still distinguishable, a parameter arbitrarily chosen. A graphical illustration is given in figure 3.8. For speeds greater than Ω_{min} the two orders frequencies will be separated by a larger value than Δf, hence extraction of the two orders will be successful. The equation defining W_l can also be written as:

$$W_l = \frac{2Bins}{\Delta Orders \cdot \Omega_{min}} \tag{3.6}$$

where in the specific case $\Delta Orders$ represents the two closest orders among the ones to be extracted. The window called W_r represents the upper limit of the possible windowing periods. It represent the largest window before the df/dt makes the two orders blend together. Its value is heavily influenced by the orders absolute value since they represent the variation of frequency over time. To better understand the limitation posed by the ramp steepness let us consider an example.
Let us imagine the user wants to perform a frequency analysis of a signal sampled at $F_s = 3250\,Hz$ composed of two sine waves with constant amplitude equal to 1 and variable frequencies over time such that:

$$f(t) = f_0 + \beta t \tag{3.7}$$

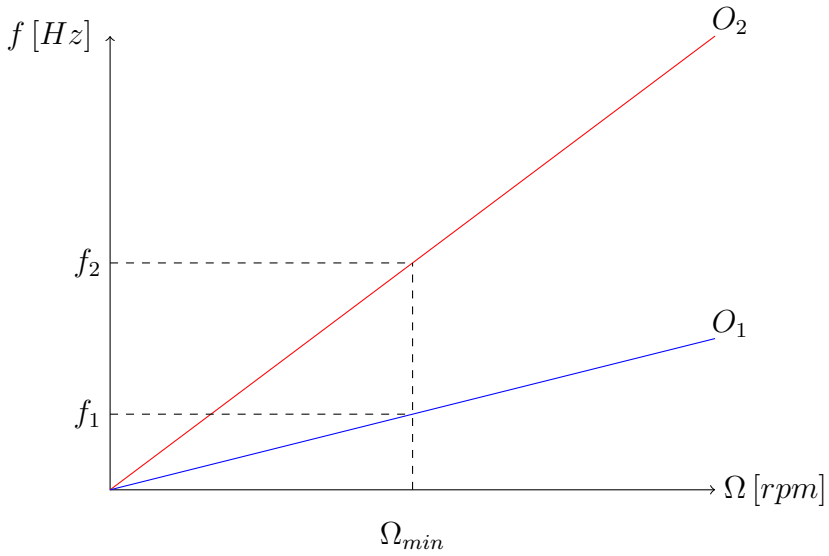


Figure 3.8: Graphical explanation of Δf

where $\beta = df/dt$,

$$\frac{df}{dt} = O \cdot \frac{d\Omega}{dt} \tag{3.8}$$

where $d\Omega/dt$ is the angular acceleration of the test run. At first let us suppose $d\Omega/dt = 1\,Hz/s$, and the two orders to be analyzed to be equal to $O_1 = 20$ and $O_2 = 30$. Thus

$$\beta_1 = \frac{df_1}{dt} = O_1 \cdot \frac{d\Omega}{dt} = 20\,Hz/s \tag{3.9}$$

and

$$\beta_2 = \frac{df_2}{dt} = O_2 \cdot \frac{d\Omega}{dt} = 30\,Hz/s \tag{3.10}$$

the frequency spectrum of the signal should show the two orders having amplitude equal to 1 during the whole run, besides showing two lines indicating the two orders frequencies increasing linearly over time. Setting the Ω_{min} to 10 rpm, from which the Δf can be calculated.

$$\Delta f = \Omega_{min}(O_2 - O_1) = 100\,Hz \tag{3.11}$$

From equation 3.6 the W_l is defined as $0.0167\,s$, using a FlatTop windowing function. The W_l defines the lower limit of the plausible T_w, any value greater than W_l results in a successful order separation. To obtain a low frequency resolution the T_w is picked to be $1.6\,s$. The result should show a colormap having good frequency resolution while being affected by some amplitude error. From figure 3.9 it is clearly visible that, despite the windowing period being relatively high, the frequency resolution does not seem to be low (remembering that the frequency resolution should be $1/1.6 = 0.6250\,Hz$). The bins seem to be too large to depict accurately the two orders frequency lines. That is due to T_w being relatively large, as a matter of fact the frequency content cannot change within the windowing period, hence the frequency lines resemble more a stairway than actual lines. In particular, in a windowing period the two orders see their frequencies change of a Δf defined as:

$$\Delta f = \frac{df}{dt} \cdot T_w \tag{3.12}$$

(not to be confused with the Δf of equation 3.6) which for O_1 is 32 Hz and for O_2 is 48 Hz. For low values of time, when the two orders frequencies are close, this means that they blend together, preventing a precise extraction. This effect, which later on will be referred

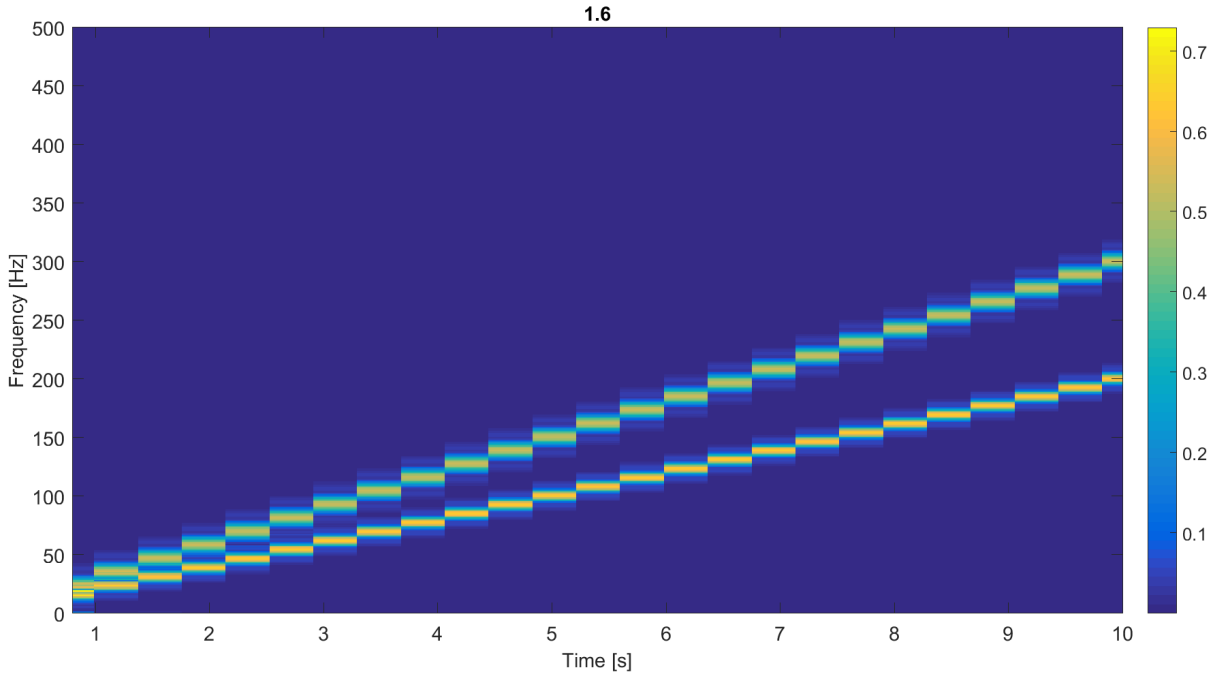


Figure 3.9: Colormap with $T_w = 1.6\,s$, zoomed from $0\,s$ to $10\,s$, signal with $d\Omega/dt = 1\,Hz/s$

to as stairway effect, can be reduced with a lower T_w, which would allow the frequency spectrum to be updated more frequently over time. Despite this issue, the colormap is readable and achieves a sufficient degree of clarity.
Now suppose to repeat the test run with a greater angular acceleration. Setting $d\Omega/dt = 6\,Hz/s$, keeping the same value of T_w of 1.6 seconds, same windowing functions and same signal, the resulting colormap is shown in figure 3.10. From the colormap

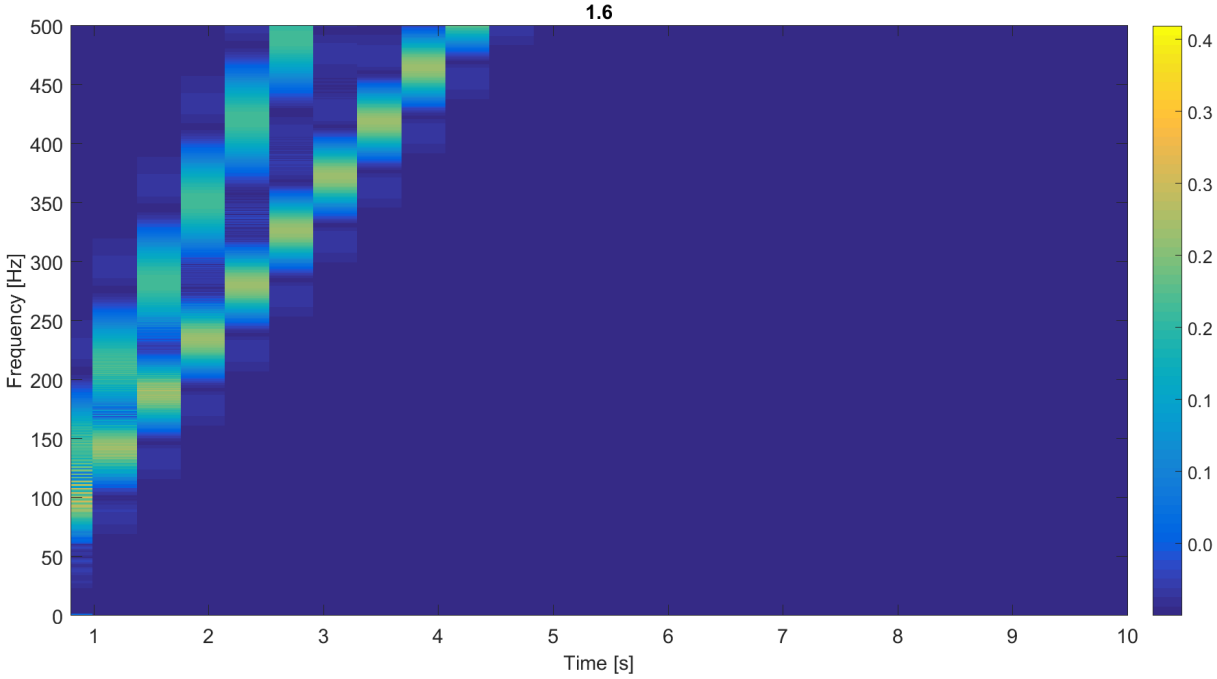


Figure 3.10: Colormap with $T_w = 1.6\,s$, zoomed from $0\,s$ to $10\,s$, signal with $d\Omega/dt = 6\,Hz/s$

it is possible to see that due to the increase of $d\Omega/dt$ the stairway effect is even more

present. As a matter of fact now the Δf of a window is six times greater (192 Hz for O_1 and 288 Hz for O_2), so the two orders are separable only after longer times, where the frequency difference between the two orders is greater. The effect is so relevant that the colormap is hardly even readable.
Now the same signal is analyzed with a smaller windowing period of 0.4 seconds. The re-

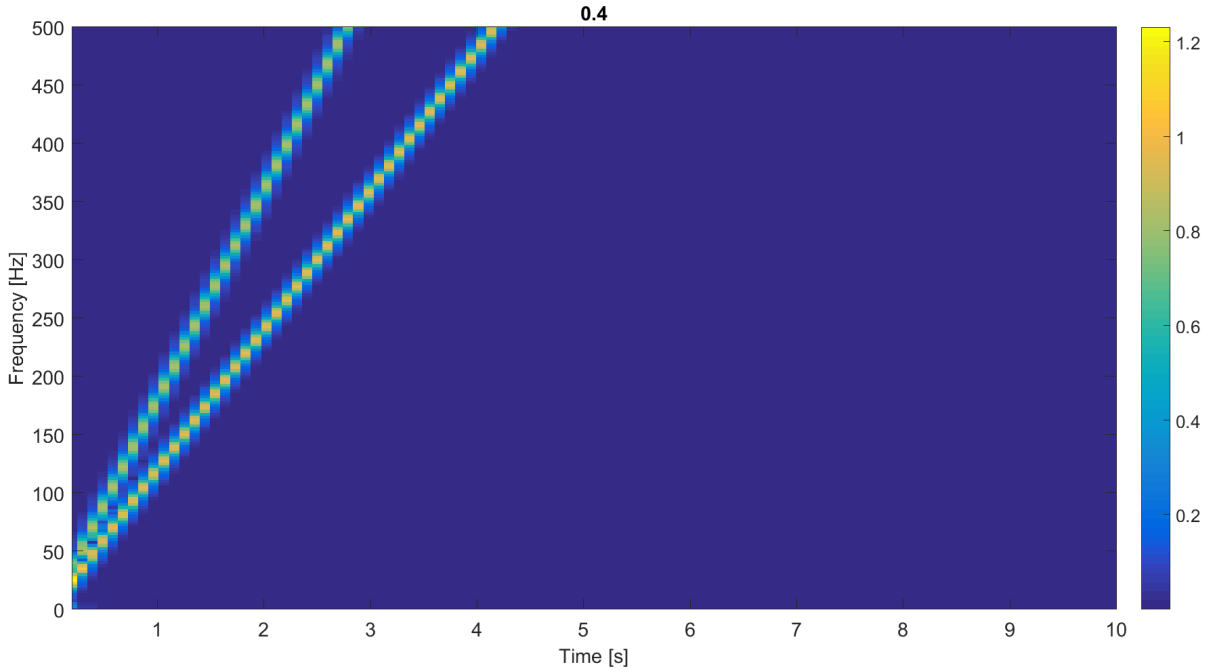


Figure 3.11: Colormap with $T_w = 0.4\,s$, zoomed from $0\,s$ to $10\,s$, signal with $d\Omega/dt = 6\,Hz/s$

sulting colormap is shown in figure 3.11. Decreasing the windowing period has alleviated much of the stairway effect, now the Δf are respectively for O_1 and O_2 48 Hz and 72 Hz. The colormap is readable, but probably the frequency resolution is slightly too high, being now equal to 2.5 Hz.
From these example it is clear that $d\Omega/dt$ is a parameter to be taken into account whenever performing a frequency analysis. In particular, while theoretically increasing the windowing period should result in a lower frequency resolution, and therefore a clearer colormap, the angular acceleration poses a limit on the highest T_w allowable before the two orders merge together due to the ramp steepness. In general, test runs having high $d\Omega/dt$ are more troublesome to be analyzed, since they pose more strict requirements from the windowing period point of view.
In some unfortunate cases $W_r > W_l$, which means that there is no windowing period satisfying both conditions imposed by W_r and W_l. This happens when the Δf in the two windows definition in table 3.2 is a small value. From the definition of Δf in equation 3.5 it is clear that as the two orders to be separated tend to the same value $\Delta f \to 0$, hence W_l tends to increase, while W_r decreases. This poses a limit on the closest orders to be separated which cannot be crossed. As a matter of fact, before selecting the windowing period, the code verifies whether $W_l \leq W_r$ is true. If it is false, rather than considering the $\Delta Orders$ of the two lowest values among the orders to be extracted, it sets the parameter to the second closest orders, and checks again the condition on the two windows. The code keeps changing $\Delta Orders$ until it finds a value satisfying both conditions. After every change in $\Delta Orders$ the code warns the user of which orders are too close to be separated. In figure 3.12, an example of the two windows, as the $\Delta Orders$ diminishes, is shown. From figure 3.12 it is possible to see that as the $\Delta Orders$ decreases, the values of

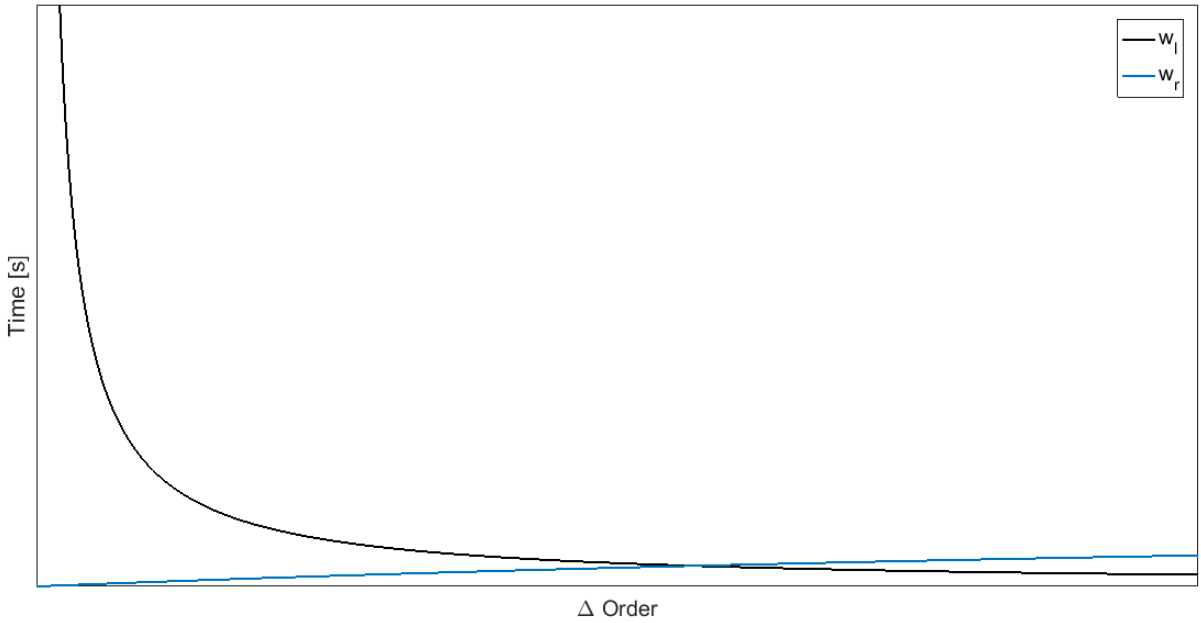


Figure 3.12: W_l and W_r at different $\Delta Orders$

the interval of possible windows becomes narrower, and at a certain value $W_l = W_r$. For smaller $\Delta Orders$ the window condition will not be satisfied and the code will not be able to select an adequate windowing period.
Once this process is finished the code selects either W_l or W_r as windowing period for the frequency analysis according to the specified analysis focus. Obviously if the focus is on minimizing amplitude error, then the chosen window would be the shortest of the interval (i.e. W_l), while for better frequency resolution the picked window would be the longest (i.e. W_r). Every window between W_l and W_r would yield a successful frequency analysis, though to pick any of the intermediate periods with a motivation would require more calculations and knowledge of the actual shapes of the peaks that are to be investigated through the analysis [5]. Such procedure requires the knowledge of the signal as a function of time free from errors which is an available information only for custom made signals, but not for real case scenarios in which very little is known of the signal before being processed through a post processing of any kind.
In equation 3.8 it is shown that orders value influences the df/dt of the test. In particular the value df/dt is known once the $d\Omega/dt$ is known. To calculate the value of the W_r thus, the code has to recognize and measure angular accelerations. Usually the test run speed profiles have constant acceleration, sometimes one single ascending or descending ramp is present, other times several ramps are performed within the same test. Considering how different experimental data can be, simply performing a finite difference of the speed data, followed by an average of every acceleration obtained that way would not be a precise enough method to know the acceleration of the reference shaft. For data containing several ramps that method would give results very far from reality, hence a more sophisticated algorithm is to be employed.
Such task is not easily performed since the speed signal is often affected by some noise. The first step of the acceleration finding function is to downsample the speed.
This is done for two reasons:

1. it reduces the memory occupied by the speed variable, improving computing times

2. it acts as a lowpass filter, removing some of the signal noise

signal downsampling must be applied with caution as lowering the F_s means that aliasing occurs for lower frequencies, besides a smaller amount of data means that the resolution of the speed data as a function of time will be larger. Both these problems do not concern speed data: aliasing is not an issue since the signal frequency spectrum will not be extracted as it is not meaningful; and speed run test profile is always linear, hence it could be precisely represented even by very few points (as a matter of fact an accurate reading of the slopes could be done with only two points per ramp). In figure 3.13 an example of

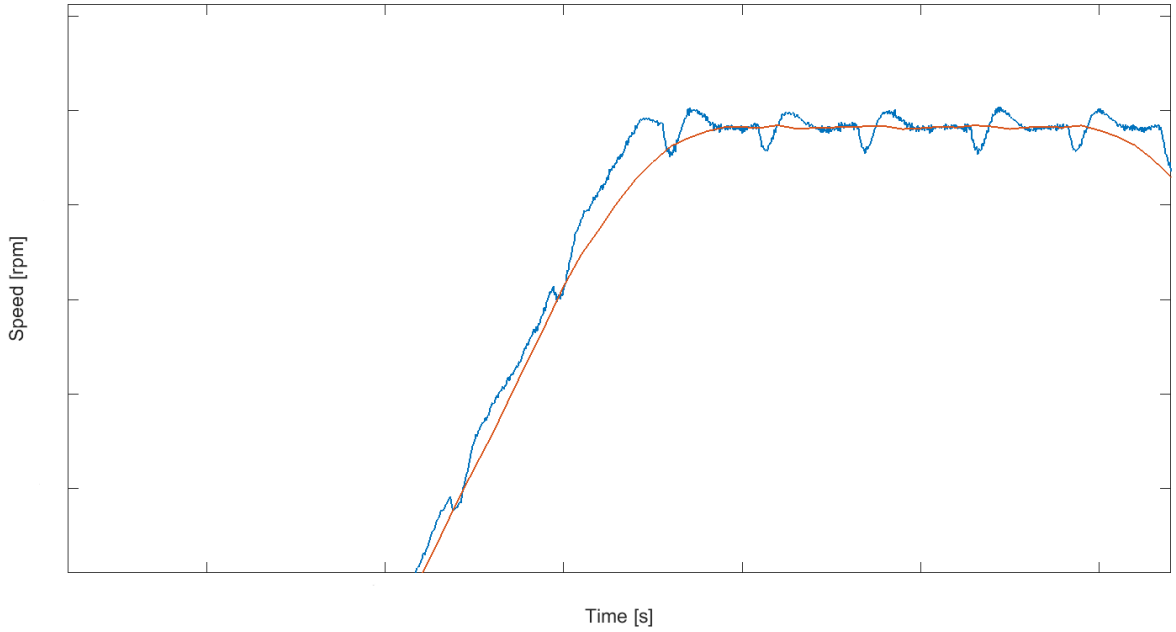


Figure 3.13: Effect of downsampling on speed data, blue is the original data, orange is the downsampled version

downsampling is shown as the speed reaches a zone with null acceleration. The blue line being the original signal at sampling frequency of F_s gets downsampled to a frequency of $F_s/10^4$. The high frequency noise has disappeared while the mid to low frequency noise is still present. Besides, the picture supports the fact that the downsampling does not depict a misleading trend with respect to the original signal.
Once the signal is downsampled, knowing the time interval between consecutive samples acceleration is calculated by means of finite differences of the speed vector.

$$a_i = \frac{v_{i+1} - v_{i-1}}{2\Delta T} \tag{3.13}$$

In this case noise affects the signal to an extent that the acceleration signal is hardly meaningful. In particular, the signal has a succession of peaks above and below the mean value which is the value it should depict ideally. Some kind of smoothing is required to figure out the speed ramp slopes. Conventional filters (such as IIR or FIR) are not suitable in this case for various reasons. First of all they either cause an overshoot of the signal, which would alter significantly the windowing period determination. Besides, they are quite heavy computationally since the designing of the filter requires the knowledge of the frequencies to be damped, which can only be obtained through an FFT. In addition, they usually introduce a phase delay which can significantly alter the original signal. To allow for the required smoothing the code employs a filter based on peaks. In particular, it first looks for both peaks above and below the mean value. Subsequently, for every peak, it checks whether the following one is not of the same kind, if such condition is satisfied the

peaks value becomes the mean value between itself and the following peak. After every peak is analyzed this way, this procedure is repeated until the code finds no more peaks, which usually occurs between 1000 and 2000 iterations depending on the signal. It must be said that while this filtering process is not computationally heavy (it generally takes about 5 seconds) it is not able to level the acceleration even with an infinite number of iterations. This is due to the fact that the code defines a peak an element of a vector which value is higher than the ones of both its neighboring elements. There is a threshold value above which the peak has to be with respect to its neighbor to be considered as such, generally the threshold is 1^{-8}. Setting it to lower values will increase significantly the computing time, but yields results comparable to the ones at the original threshold.

In figure 3.14 the acceleration profile of an entire test run is shown. The smoothing

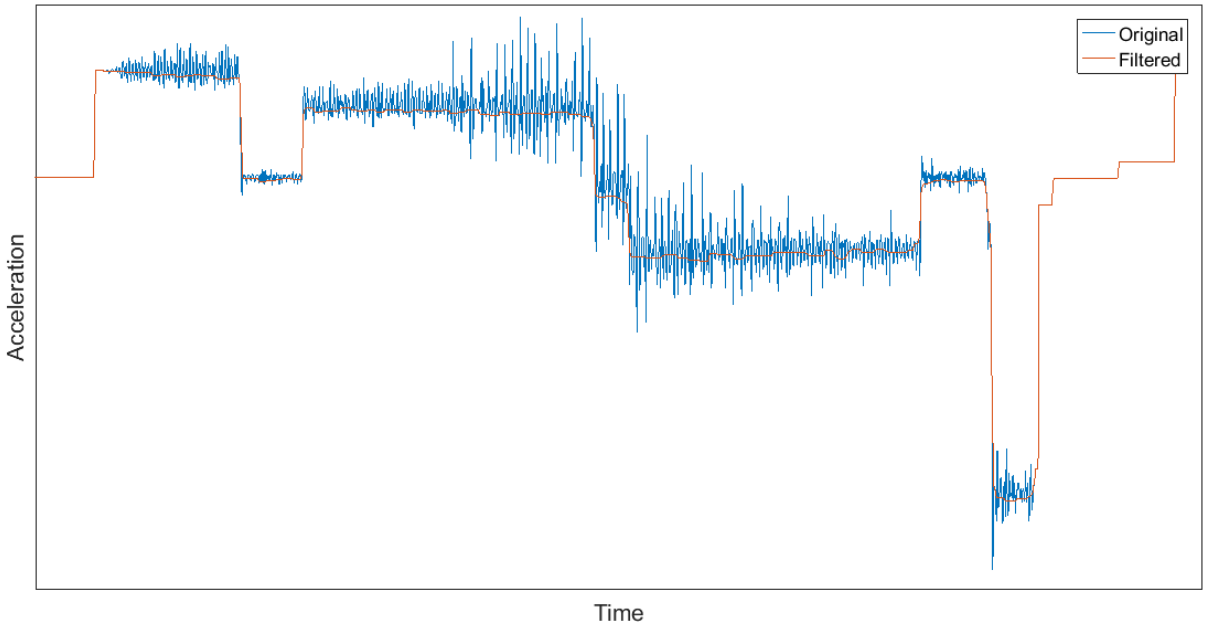


Figure 3.14: Test run acceleration profile filtered and original

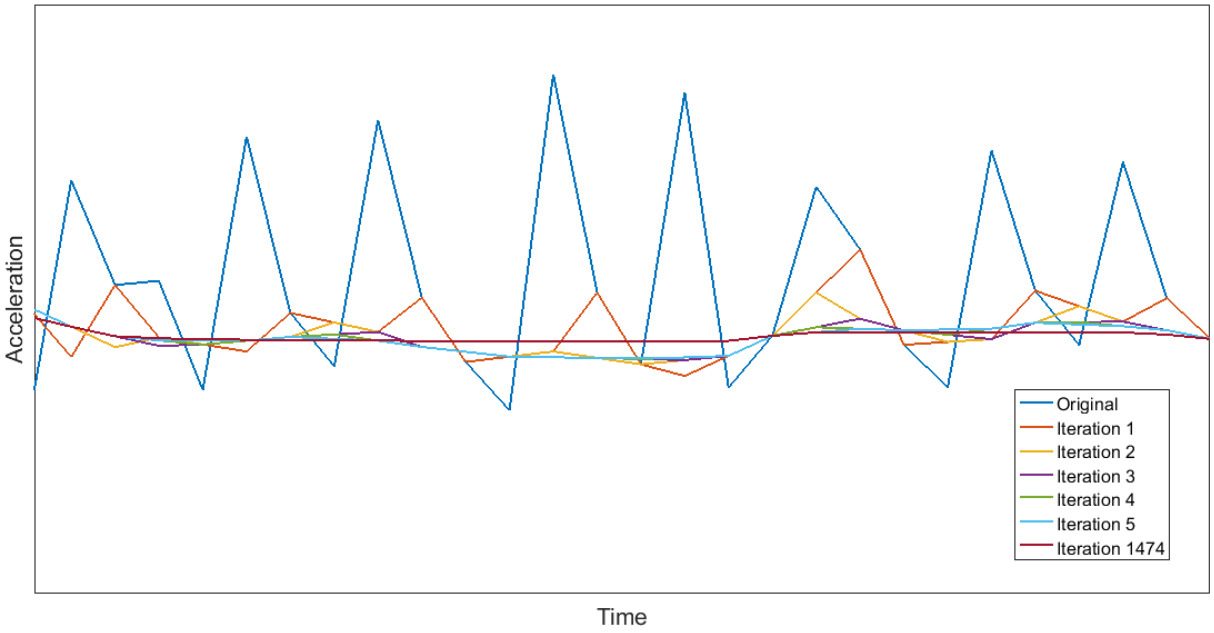


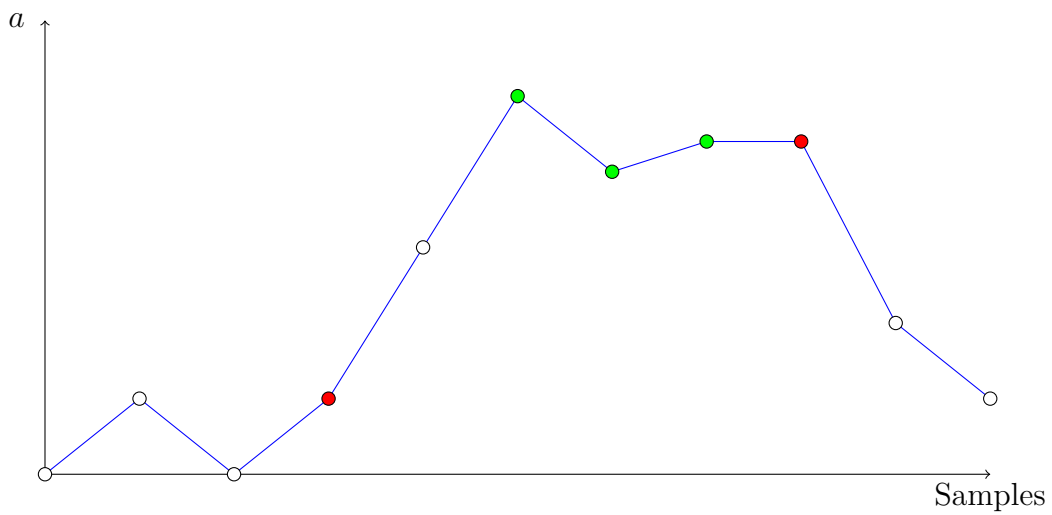
Figure 3.15: Focus on a short time interval of the first 5 filtering iterations and final result

applied by the code is evident, and allows for easy determination of speed ramps.

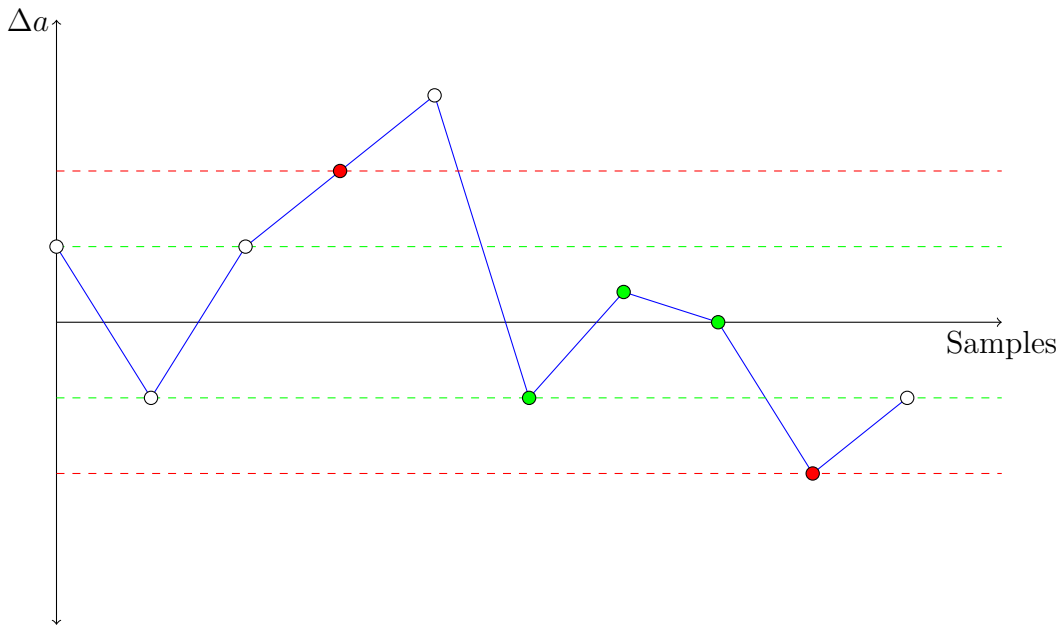
This filtering method is not flawless however, like previously mentioned there can be some particularly noisy speed data which would require some extra filtering, which could be easily achieved by a moving average filtering of the speed data. The problem is that the process should be automatized and the code should be aware of when this is a required operation to be performed. In such cases the ramp finding function might fail to recognize the correct ramps and stop its execution, the code would then warn the user then abort the research of a W_r and assign to the windowing period a default value equal to $0.25\,s$.
The output required from the ramp finding function is a vector containing every acceleration value measured within the test run speed profile. To produce this data from the filtered acceleration vector, the first step is defining where the ramps begin and end. To perform this the user has to set the ramp tolerance parameter to either 'low', 'medium', or 'high'. This parameter includes every tolerance required to define the ramps from the filtered acceleration signal. In case the user does not have any information allowing him to choose the appropriate tolerance setting, the parameter can be left blank, the function will automatically set it to 'high'. The function checks where the difference between two consecutive values is higher than the trigger tolerance value, and marks it as a ramp beginning, in figure 3.16 those points are marked in red. Subsequently performs the same operation to find the index at which ramps end, again checking two consecutive acceleration values against trigger tolerance. At this point the function loops for every ramp beginning and checks for two consecutive values being closer than the constant tolerance to mark the beginning of the constant acceleration zone. From this value checks for the first couple of consecutive values whose difference is higher than constant tolerance to define the end of the constant acceleration zone. To compose the output vector, the function calculates the mean value of the acceleration vector elements included in the constant acceleration zone. The last operation is the cleaning up of the output vector from acceleration values lower than the minimum acceleration value, or higher than the maximum acceleration. This whole ramp finding process is inserted in an error handling sequence, hence, in case an error stops the function from reaching the end, the sequence restarts with a lower setting threshold, until the setting is 'low', at that point the windowing period is set to the default value.

3.1.4 Autoslice function

As shown in figure 3.1 depicting the frequency analysis work flow, the final sequence is optional and is useful if the analysis is focused on finding crossings between modes and orders, and evaluating their amplitude. The left branch of figure 3.1 shows the main sequences composing the Autoslice algorithm, which role is to identify peaks crossing a set threshold and assigning to each of them a mode and an order. Besides, the user can insert additional inputs concerning the stiffness of every mode, thanks to which the code is able to calculate the stress caused by the peaks highlighted.
Before looking for peaks the function applies the correction to the modes natural frequencies f_0 according to the temperature indicated. The user has to input two temperatures, one is the reference temperature at which the given f_0 are calculated, the second is the temperature at which the test run is performed. Often times lubricant oil in aeronautical gearboxes is also used in heat exchangers to cool down some components, therefore it is not unusual to see oil temperatures of around $100°C$, which can significantly modify the mechanical properties of the gears. As a matter of fact increasing the gears operational temperature causes a reduction in material stiffness which cannot be neglected. To consider this effect, the code firstly calculates the young modulus at the reference temperature, then applies the effect of the test run temperatures. To calculate the young modulus the code simply performs a linear interpolation between the value at $20°C$, the



(a) Acceleration vector



(b) Difference between acceleration vector elements

Figure 3.16: Example of acceleration vector being analyzed by the ramp finding function. Red samples are the ones limiting the ramp, green ones are the ones limiting the samples taken into consideration while performing the average to fill the output vector. The green dashed line is the constant tolerance

value at $150°C$ and the query point, hardly ever temperatures outside this interval are required, nevertheless they would not be cause code failures. Once the young modulus at the reference and test run temperatures are calculated the following relation is used to determine the new natural frequencies.

$$f_{0,corr} = f_0 \cdot \sqrt{\frac{E(T_{test})}{E(T_{ref})}} \tag{3.14}$$

Sometimes the test run or the reference temperature might not be known. As shown in table 3.1, in case some temperature data is missing the code automatically avoid any temperature correction, warning the user that the modes frequency considered by the code might be slightly off the ones visible in the colormap.

After applying the temperature correction (if needed), the code takes into account another effect that can modify modes frequency: gyroscopic effects. In case some of the modes to be analyzed are radial, their slopes is not simply the nodal diameter value, but, as mentioned in chapter 1, there is a correction to be applied to the slopes. The user can select a number from zero to one indicating the entity of the radial correction to be applied, one being full radial correction, zero being full axial correction. For every number in between the code would make a weighted average between the two slopes and apply the result to calculate the slopes of the mode. The correction applies both to the forward and the backward frequencies.
Once the corrections are applied, the first task performed by the Autoslice function is looking for peaks in the peakhold that cross the given thresholds, in figure 3.17 an example of this operation is shown. The threshold is typically variable with respect to frequency. To allow for detection of peaks at high frequencies, which typically have lower amplitudes if compared to the ones at low frequencies, it is a good practice to specify lower amplitude thresholds at those frequencies. Performing this operation is equivalent to checking for

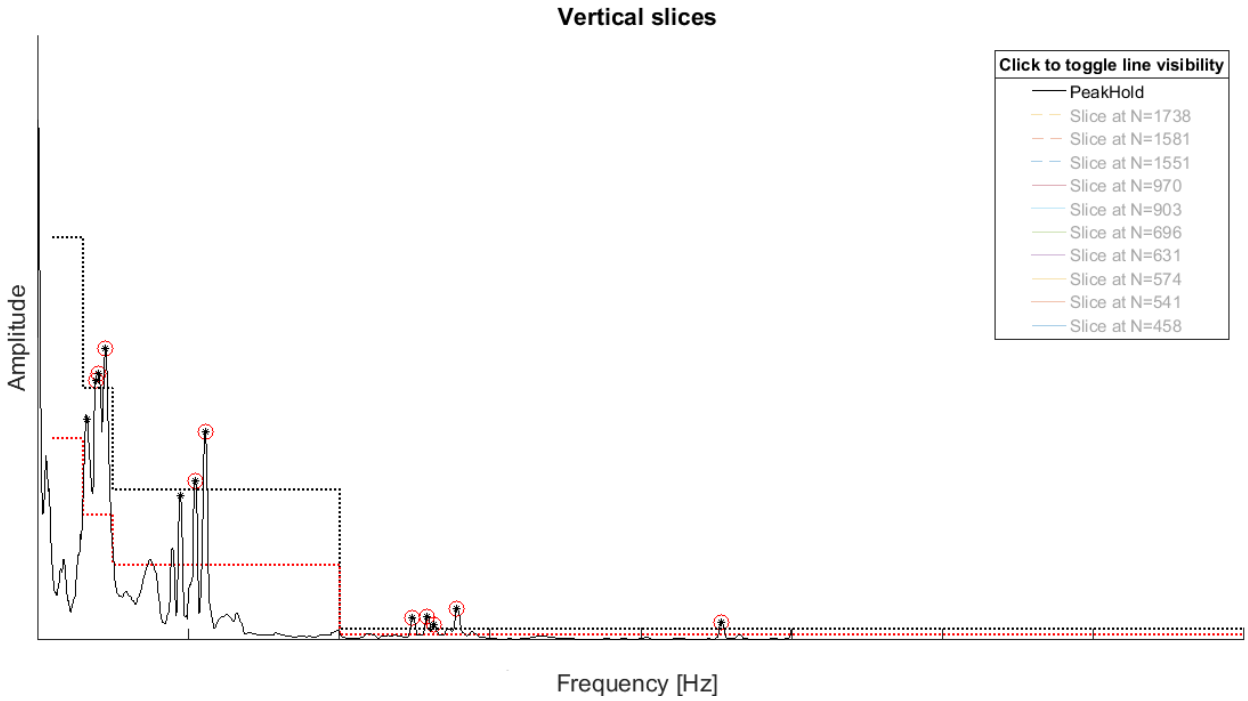


Figure 3.17: Peakhold (continuous line) with threshold (dotted line)

every x-axis value (i.e. every amplitude matrix column) if the maximum value crosses the given threshold, with the sole difference being that the peakhold does not give any information of the x-axis value of each peak. For this reason, once the relevant peaks are found, the function locates the peaks in the colormap by looking for the maximum amplitude value at every peak frequency. Once the x-value of the peaks is found the function slices vertically the colormap, thus extracting the amplitude value of every frequency at the peaks x-axis values, and checks that data against a lowered version of the threshold. The magnitude of the threshold reduction is specified by the user. In figure 3.17 the black dotted line is the threshold, while the red dotted line is the lowered threshold, which is a half of the original value. The red circled dots are the peaks crossing the regular threshold while the points marked with an asterisk only cross the lowered threshold.
Once the peaks are correctly identified, the function checks for each one of them whether their location in the colormap is close to any mode. To do so the user has to input two tolerances, one relative and one absolute. The relative one is used in the following

equation:

$$\frac{|f_{pk} - f_{mode}|}{f_0} < tol_1 \tag{3.15}$$

where f_{pk} is the peak frequency, f_0 is the natural frequency of the mode, and f_{mode} is the vector of the mode frequency as function of the x-axis. The absolute tolerance is used in the following equation:

$$f_{pk} - f_{mode} < tol_2 \tag{3.16}$$

the function checks every peak with respect to every mode both forward and backward specified by the user. For every peak found to be the result of a crossing between a mode and an order, the function calculates the stress caused by it. This is meaningful only for data concerning displacement, and is simply done by multiplying the peak amplitude and a stiffness coefficient (S_c), related to young modulus, which gives the ratio between displacement and stress deriving from it (equation 3.17).

$$S_c = \frac{\sigma}{u} \left[\frac{MPa}{\mu mm}\right] \tag{3.17}$$

Every mode has its own stiffness coefficient, which typically increases exponentially increasing the mode number of nodal diameters.
After the stresses are computed, the contribution of forward and backward peaks gets summed to yield the total stress of the mode since the two frequencies coexist. While forward and backward are two separate harmonics, they still belong to the same mode, hence in case both of them produce a peak crossing even with different orders, their stress contribution has to be summed up.
Once every computing operation is performed the function passes every relevant data to a dedicated output function, whose role is to print text file summarizing the peaks data, and showing the stresses caused by the crossings between orders and modes. The output produced changes slightly depending on the kind of x-axis of the analysis.
In particular the text file contains the following fields for each peak:

- Amplitude, which shows the peak amplitude
- N, which indicates at what speed the peak occurred. Alternatively, in case the x-axis is temporal, this field gets renamed and shows the time at which the peak occurred
- Frequency, which shows at which frequency the peak belongs to, it is its y-axis position
- Order, knowing the frequency and the speed at which the peak occurred, it is a quick task to calculate the order at which the peak belongs. It simply is $f/(N/60)$; where f is the frequency of the peak, and N is the speed, which is divided by 60 if the unit is rpm (which is usually the case)
- Stress, which shows the sum of the stresses caused by the peak to every mode

Below the table with the peaks data, the text file displays two matrices, concerning the stress data for forward and backwards harmonics. The matrices a NxM with N rows equal to the number of peaks, and M number of modes. To obtain the stress data for the previous table, the output function sums the stresses concerning the peak of interest (it sums every value on the same row) of the forward and backward stress matrices, and then sums the two stresses to yield the total stress caused by the peak.
Besides the text file, the function produces another output, produced by another dedicated function, which is a plot showing the vertical slices alongside the peakhold. It is shown in figure 3.17, thanks to a callback it is possible to toggle the visibility of the various slices.

Parameter	Parameter meaning	Auto value	Condition to be satisfied
Analysis focus	Specifies whether the following automatic choices will favor amplitude precision or frequency resolution	Amplitude	NA
Windowing function	Either Hanning or FlatTop windowing, depends on the analysis focus	Depends on the analysis focus	NA
Overlap	What percentage of truncated signal gets overlapped while performing the FFT	Depends on the windowing function	NA
Windowing period	Temporal length of the segments in which the signal gets truncated	Default value is $0.25\,s$	$W_l \leq W \leq W_r$
X-axis	Colormap x-axis, either speed or time	Time	NA
f_{max}	Maximum frequency whose amplitude will be displayed in the colormap	$\frac{F_s}{2.56}$	$f_{max} < \frac{F_s}{2.56}$
x-axis resolution	Resolution of the x-axis, only if speed is selected, for time x-axis this is $1/F_s$	1 [RPM]	Limited according to frequency range to prevent colormap being too heavy
Temperature correction	Toggles temperature correction for modes f_0	No correction	NA
Amplitude reduction	Coefficient of reduction for amplitude threshold in Autoslice	1	NA
Absolute tolerance	Tolerance used to detect crossing between modes and orders	$100\,[Hz]$	NA
Relative tolerance	Tolerance used to detect crossing between modes and orders	0.03	NA
Reference speed	Speed used to calculate the modes frequency	Input speed	NA
Orders tolerance	Tolerance used to extract orders from colormap	$tol = \frac{O_2 - O_1}{O_2 + O_1} \cdot 100$	$tol \leq \frac{O_2 - O_1}{O_2 + O_1} \cdot 100$

Table 3.1: Complete list of parameters handled by the automatic parameter selection function

Tolerance	**'high'**	**'medium'**	**'low'**	**meaning**
Trigger tolerance	0.01	0.4	0.6	Minimum difference between two consecutive accelerations values required to consider them the beginning of a ramp
Constant tolerance	0.05	0.07	0.1	Minimum/maximum difference between two consecutive accelerations values to define the start/end of a constant acceleration zone
Minimum acceleration	0.005	1	1.5	Ramps having lower acceleration values than this threshold will be deleted
Maximum acceleration	10	20	35	Ramps having higher acceleration values than this threshold will be deleted

Table 3.3: Ramp finding thresholds, the values are in rpm/s

Chapter 4

Order Analysis

As mentioned in chapter 3, frequency analysis allows to highlight phenomena that are periodic over time. In fact in the frequency spectrum the amplitudes relative to such phenomena will stand out from the random aperiodic noise. In some cases, though, the phenomenon to be analyzed is related to the machine cycle, hence rather than repeating itself after a given time interval, it repeats itself after a regular number of revolutions. For example let's consider for instance a vibration analysis performed on an internal combustion engine operating at variable speed. If the test run is divided into different time segments of equal length, which is the operation performed while windowing the signal, every segment will include a different amount of strokes since the engine shaft speed is not constant over time. Therefore, due to the phenomenon not being periodic over time, it would not be clearly represented through a frequency analysis.
The most effective tool to analyze internal combustion engine strokes, and every other phenomena periodic over the reference shaft revolutions, is the so-called order analysis. An order is simply defined as a pure number representing the multiple of a reference rotational frequency, in the case of the combustion engine a typical reference speed could be the rotational frequency of the shaft.
From a computational standpoint, order analysis performs exactly the same passages of the frequency analysis, the only difference is that the signal analyzed has to be sampled at constant angle intervals. Once the resampling procedure is completed, the signal undergoes the same preprocessing described in section 2.2. In table 4.1 the parallelism between frequency and order analysis parameters is shown. From table 4.1 it is important to notice that, similarly to frequency analysis, also for order analysis there is a Nyquist theorem which limits the frequency domain not affected by aliasing, the Bruel Kjaer signal analyzers company has discovered that in order to have a precise resampling $2 \cdot 2.56 = 5.12.$ samples per period of the highest frequency of the range are required [6].

Frequency analysis	**Order analysis**
Time	Angle
Frequency	Order
Sampling frequency	Points per revolution
Nyquist condition	Nyquist condition
$F_S \geqslant 2.56 f_{max}$	$ppr \geqslant 5.12 Ord_{max}$
Sampling period $dt = T_0 = \frac{1}{F_s}$	Sampling angle $d\theta = \frac{1}{ppr}$
Windowing period T	Windowing revolutions N_{rev}
Frequency resolution $df = \frac{1}{T}$	Order resolution $dOrd = \frac{1}{N_{rev}}$

Table 4.1: Order analysis parameters correspondence with frequency analysis

There is, however, an important practical difference: in the case of frequency analysis the sampling frequency F_s is not a parameter that can be freely set, like the windowing period for example. It is rather a hardware feature, which solely depends on the settings of the hardware employed during the test run. For order analysis, though, the points per revolution (*ppr*), the parameter equivalent to the sampling frequency, can be chosen freely since the signal undergoes resampling. The only downside to selecting a big *ppr* is that the computational cost of the analysis will increase exponentially. Extremely high values of *ppr* are not usually needed, for example a value of $4 \cdot 360 = 1440$, which would generate a signal with 4 samples between each degree, yields a sufficiently accurate data for most purposes. Considering that sampling frequencies usually are of the order of at least 10^4 Hz, which means that 10^4 samples are being taken each second of the test run, and that the maximum reference shaft speed generally is around 2000 rpm ($\sim 209\, rad/s$), the original data contains about $10^4 \cdot 2\pi/209 \sim 300$ samples each revolution, which is two orders of magnitude higher than the density of resampled data at $ppr = 1440$. From this estimate it is clear how the time sampled data is dense enough to allow the *ppr* to be raised significantly above 1440 points per revolutions if need be. This also means that resampled data are almost always lighter than their time sampled version, which mitigates the extra computational time required for the actual resampling of the signal.
Concerning the other parameters, they are analogous to their frequency analysis counterparts. There is a windowing number of revolutions to be picked which is determined from the two closest orders, and the windowing function, which reduces the spectral leaking of the analysis.
The advantages order analysis offers over frequency analysis are twofold.

1. as shown in the following sections, resampling the signal synchronously to the reference shaft allows to have a signal with null df/dt. Thus orders are represented as horizontal lines, negating the speed ramp steepness effects described in section 3.1.3

2. in case some noise not synchronous with the reference shaft is present in the signal, it can be easily filtered by averaging the frequency spectrum of different revolutions altogether

4.1 Synchronous resampling

As mentioned in the previous section, order analysis requires the resampling of the raw input signal, in this section an overview of said operation is given.
The steps toward resampling a time sampled signal into an angle sampled one are the following:

1. obtain a valid speed reading of the reference shaft

2. upsampling the signal

3. low pass filtering

4. cubic or linear interpolation

while upsampling and the subsequent low pass filtration are not strictly necessary, but rather can be used to increase the precision of the resampling, steps 1 and 4 cannot be skipped.
In order to obtain the speed data of the reference shaft, an optic sensor is mounted on it. In case the reference shaft is not accessible, or equipping it with the sensor would result

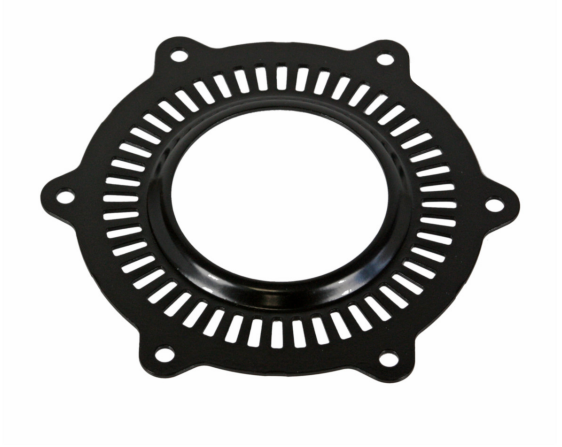


Figure 4.1: Phonic wheel

unpractical, it can be mounted on another shaft, and, knowing the transmission ratio between the two, the speed of the reference shaft can be determined. The voltage data produced by the sensor is then converted into speed data by some dedicated functions in the code, which produce the speed vector as a function of time and the vector specifying the beginning time of each revolution the reference shaft performs during the test run.
The optical sensor usually employed for this task is composed of an emitting and a receiving part. The emitting end generates a light beam which is pointed toward the receiving end. The sensor generates a voltage signal representing the intensity of the light beam perceived by the receiving end. Between the two parts a phonic wheel is positioned, which is shown in figure 4.1, the wheel rotates with the shaft whose speed is to be measured. By revolving, the wheel alternatively blocks or clears the passage to the light beam generated by the emitting part of the optical sensor. The optical sensor then measures pulses representing the time periods in which the teeth do not block the beam. With this data, generally referred to as tachometer signal, knowing the angular distance between the phonic wheel teeth, and the time passing between two pulses, it is possible to calculate the rotational speed of the shaft rotating with the phonic wheel. As a matter of fact, a revolution is performed once z pulses are measured, where z is the number of teeth of the phonic wheel. The time between z pulses is known since the signal being analyzed is sampled at constant time instants defined by $1/F_s$.
In order to efficiently count the pulses from the tachometer data string, the signal has to become digital, which means that rather than assuming continuous values it can only contain either zeros (representing the light beam being blocked) or ones (representing the light beam being able to light the receiving part of the sensor). To perform the conversion the tachometer data string is passed through a Schmitt trigger, which operates by defining two thresholds, one upper and one lower. The samples whose amplitude is inferior to the lower threshold become zeros, the ones above the upper become ones. Whenever a sample is located in between the two thresholds it maintains the value of the previous sample. In table 4.1 the functioning of the device is summarized. In case the tachometer

Analog sample value	**Digital sample value**
$x(n \cdot T_s) \leq th_l$	$x(n \cdot T_s) = 0$
$x(n \cdot T_s) \geq th_u$	$x(n \cdot T_s) = 1$
$x(n \cdot T_s) > th_u \wedge x(n \cdot T_s) < th_u$	$x(n \cdot T_s) = x((n-1) \cdot T_s)$

Table 4.2: Schmitt trigger applied to the n sample of a time dependent $x(t)$ signal

signal passed to the Schmitt trigger is not as regular as the one showed in figure 4.5, some problems might arise, requiring some more operations to be performed in order to obtain a correct reading of the phonic wheel crossings. An example of such a case, and the consequences resulting from it, is described in section 5.1.3.
Rather than the speed data, to perform the resampling a *synchronozing pulse* vector is required. This vector contains the time at which every revolution starts, it is of paramount importance for the resampling procedure, and is generated from the process just described. As a matter of fact, it already has been pointed out that one revolution is defined by counting an amount of pulses equal to the wheel teeth. Any possible error during the conversion would strongly affect the outcome of the resampling.
Once the *synchronizing pulse* vector is obtained, it is possible to generate the synchronized data vector since the *ppr* defines how many points are required within the samples of a single revolution which start and end times are known. To perform that operation an interpolation is needed to generate the new sample points at the times specified by the *synchronizing pulse*. The interpolation can be either linear or cubic, other kinds of interpolations are rarely used as they would result heavier to perform and they would not provide an equivalent increase in precision. It is clear that cubic interpolation is to be preferred over linear whenever precision is the most important aspect of the analysis, at the cost of heavier computational loads, especially for upsampled signals. Interpolations always introduce an error, in order to reduce it an upsampling might be required.
Upsampling consists of artificially adding new samples of null value between already existing ones. After the interpolation, the signal which underwent upsampling has a higher precision due to the artificial increase of sampling frequency. This, though, comes with two main downsides: performing operations on an upsampled signal is computationally heavier with respect to the original signal, due to the increased amount of data to be processed. Besides, the process introduces high frequencies in the oversampled signal which have no physical meaning and have to be removed, the whole procedure is pictured in figure 4.6.
Once the signal is upsampled to remove the high frequencies a low pass filtering occurs. The filter is usually a digital one and can either be a FIR (finite impulse response) or an IIR (infinite impulse response), it has to be designed having a flat passband and null ripple or the signal will be either amplified or reduced. This poses a limit on the steepness of the transient band and favors FIR over IIR. Requiring the resampling function to perform some filtering over an upsampled signal can be a daunting task, and significantly increase the computational time.
From Geninatti master thesis [5], it is demonstrated that in order to have a sufficiently accurate resampling, a cubic interpolation without upsampling is usually sufficient, as shown in figure 4.4. As a matter of fact, in the examples shown in [5], upsampling raises computational cost more than increasing the precision of the operation. In general this is true only for signals with sufficiently high F_s, where for each revolution thousands of samples are present. If the F_s is not so high with respect to the rotational frequency, upsampling might be an effective tool to increase precision. Upsampling might be needed also in cases where the *ppr* is extremely high, to the point where the interpolation is requesting about the same amount of samples each revolution as the ones present in the time sampled signal. In figure 4.2 and 4.3, taken from [7] the resampling of a signal whose frequency is varying linearly over time is shown. It is clearly shown that samples equally spaced in the angle domain are not equally spaced in the time domain. In the example of the figures the df/dt is positive therefore frequency is increasing over time. For rotating machinery this occurs whenever the test run is performed at an increasing rotational speed. As a matter of fact in the colormap in figure 3.3 in chapter 3 it is clear

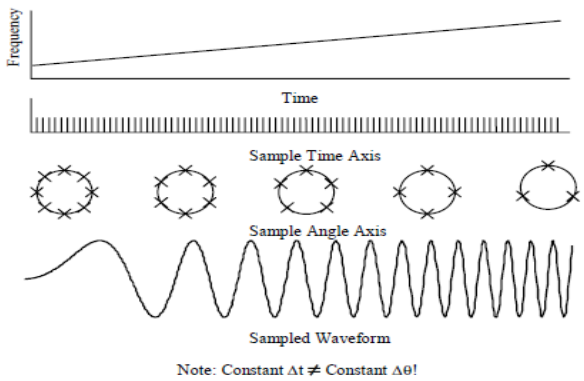


Figure 4.2: Chirp with samples equally spaced in the time domain

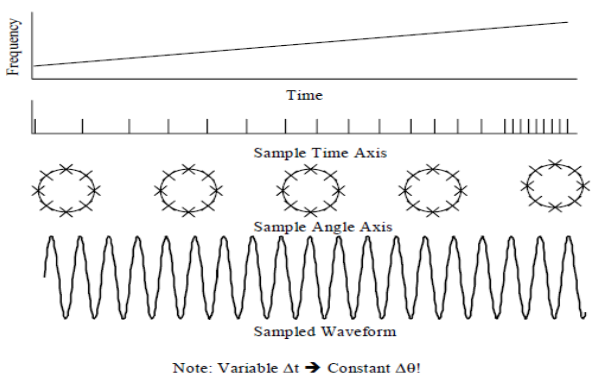


Figure 4.3: Chirp resampled with samples equally spaced in the angle domain

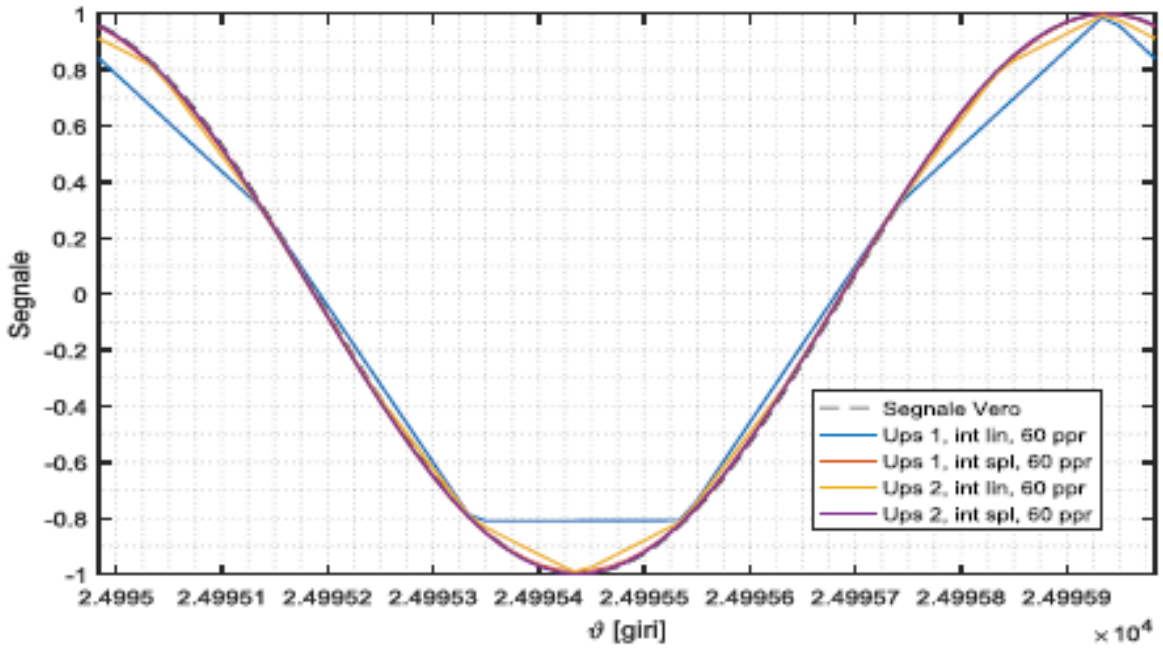


Figure 4.4: Different upsampling values and interpolation methods effects

that the order lines follow the same outline as the speed with respect to time. Looking at figure 4.3, it is clear that a signal sampled at a constant angular interval ensures the df/dt to be null, as the speed variation of the reference shaft have no effect on the amount of samples each revolution. This means, though, that for lower rotational speeds the samples are separated by long time intervals, since the reference rotor might take some seconds to perform a revolution. This can result in variable x-axis resolution for Campbell diagrams. This usually is not a concern since the colormap resulting from the order analysis, showed in the following section, pictures horizontal lines, which is coherent with a null df/dt, and is well readable even with a variable x-axis resolution.
In order to have the same amount of samples, at each revolution the sampling has to occur at an increasing frequency over time, as shown in figures 4.2 and 4.3.
Within a time interval from figure 4.3, frequency and thus rotational speed varies, this effect is not considered by the resampling algorithm which considers the frequency constant between the begin and end of a revolution, this introduces an error which is usually negligible, since the time interval in which speed is considered constant is generally very small with respect to the acceleration.

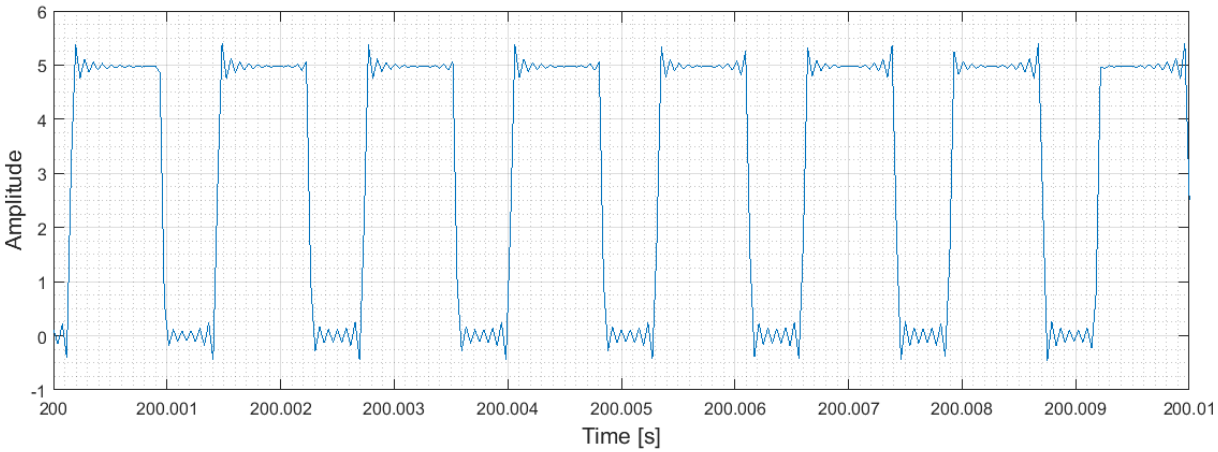


Figure 4.5: Example of tachometer signal

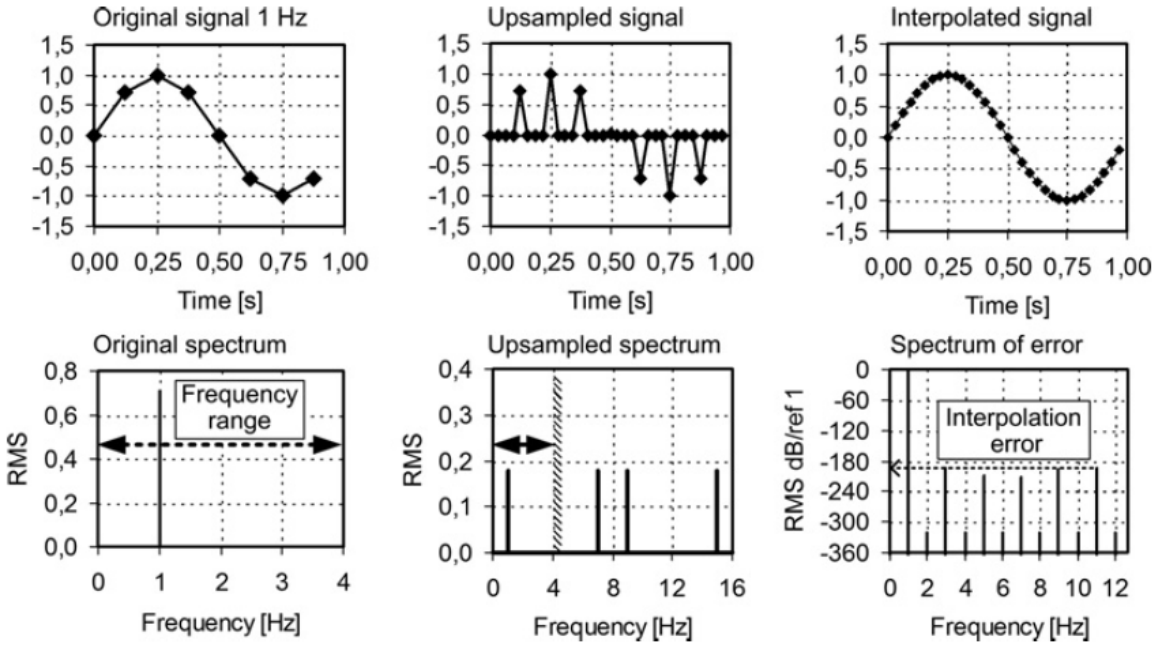


Figure 4.6: Upsampling operations

4.2 Order analysis code implementation

A dedicated function performing order analysis has been written in the code. Similarly to the one described in chapter 3 for frequency analysis, the function requires some input, which can either be inserted by the user or automatically determined by the code. In both cases for certain parameters the code carries out a control to verify whether the input is compatible with the user requests. The input that are mandatory to be inserted by the user are:

- directories concerning the location of the raw data, and the location where output files are saved
- speed and order table. Already described in chapter 3
- output settings, specifying the format and what output to save

in the case of an order analysis no input is requested by the Autoslice function since it can only be employed in case of a frequency analysis. The expected output is, also in this case, a colormap representing on the z-axis the amplitude of the spectrum, on the y-axis the order, and on the x-axis either speed or time, according to the user preference.

4.2.1 Code architecture

The code architecture concerning the order analysis function is represented in figure 4.7. Similarly to the frequency analysis there is a sequence dedicated to the automatic selection of certain input, the code actually calls the same function which is able to determine and verify both frequency and order analysis parameters. The block concerning the synchronous resampling performs the operations described in section 4.1.

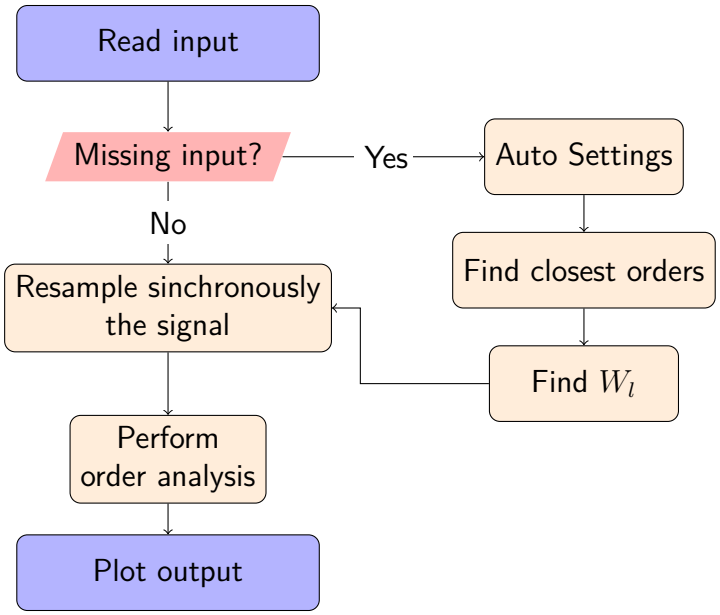


Figure 4.7: Order analysis workflow

4.2.2 Code output

After the frequency spectrum is generated, the function passes the relevant data to a dedicated function which produces a colormap and saves it in the specified format (.fig or .png) in the output folder. As mentioned in the previous sections, order analysis highlights phenomena periodic over a constant number of revolutions. In the output colormap this is visible as the lines representing specific orders have higher or lower values of amplitude. Considering a typical colormap produced by the output function, the y-axis always represents the orders. There is a limit posed by Nyquist theorem which limits the highest order not affected by aliasing which is $ppr/5.12$. The x-axis, similarly to the frequency analysis function, can either be time or the reference speed. In case of a speed x-axis, if the test run speed profile is not monotone, and the reference shaft reaches the same speed at more than one time instant, the interpolation described in chapter 3 is required in order to correctly rearrange the frequency spectrum data with respect to a monotonically increasing speed x-axis. From figure 4.8, it is possible to see that the resolution of the x-axis seems to vary with respect to the speed value, in particular bins at lower speeds seem to be repeating themselves.
It is important to remember that the resampled signal has *ppr* number of samples per

revolutions, while performing the frequency analysis, windows composed of several revolutions are processed each time. At the same time the x-axis is generated by averaging the x-axis reference values (angle for order analysis, and time for frequency) of the relative window elements. Obviously the windowing revolutions are constant and do not vary their length with respect to speed, therefore at lower speeds the windowing revolutions span over a longer time than at higher speeds. Since the speed is assumed to be varying linearly over time, this means that at lower speeds between the first and the last elements of the windowing revolutions there is a higher speed difference than from the same two elements belonging to a window positioned at higher speeds. After the average value is calculated, two windows at lower speeds are represented by average values significantly apart from each other, while two windows at high speeds have x-axis values averages fairly close to each other. If this difference is higher than the x-axis resolution, multiple x-axis elements pick their value from the same spectrum values resulting in bins duplication.
For example the colormap in figure 4.8 spans from a null speed to 1800 rpm, it is possible to see that for speeds lower than 600 rpm several bins are duplicated. In particular, due to this effect the speed range from 0 to 400 rpm is represented by only 5 bins yielding a resolution of 80 rpm, while the user defined x-axis resolution is 1 rpm. The user must be aware of the fact that bins duplication prevents the x-axis resolution setting to influence the colormap at lower speeds, while the same parameter functions properly for higher speeds.
The only way the user can act to prevent bins duplication is to reduce the number of windowing revolutions, hence allowing for a smaller speed difference between the two windows x values. The effect is visible in figures 4.9 and 4.10. The first one shows the zoom on the region between 0 and 400 rpm of the colormap in figure 4.8, which is obtained with a $N_{rev} = 50$; while the second one shows the same region of the colormap of the same raw data but obtained with $N_{rev} = 20$. Reducing the windowing revolutions parameter results in a higher order resolution, which is defined as $1/N_{rev}$, with the consequent risk of failing to separate the closest orders, so this is not always a practical choice. Another way to reduce the bins duplication is to increase the x-axis resolution, although this inherently reduces the precision of the colormap at high speeds, where bins duplication is absent. In general the only way of preventing bins duplication without any side effects is to have a test run performed at low accelerations.
In case of a time x-axis a different problem arises. The resampled signal is evenly spaced with respect to the angle domain, as described in section 4.1, hence for any given time interval Δt there is a different amount of data depending on the speed of the reference shaft. At high speeds the reference shaft is completing more revolutions than at lower speeds in a given Δt, thus at lower speeds the data density with respect to time is lower than at higher speeds. Differently from the speed x-axis, in the case of a time x-axis the spectrum data do not have to be rearranged according to their speed, so the user is not required to select the x-axis resolution, and the code does not go through the section which evenly spaces the amplitude matrix. This poses the same problem discussed in section 3.1.2 concerning the MATLAB default function for generating the colormap not being able to do so with an unevenly spaced input. For colormaps having time x-axis to achieve a proper representation the only way is to employ the amplitude matrix stretching described in section 3.1.2. In case of a speed x-axis both interpolation and matrix stretching are available options. In figure 4.11 a colormap obtained with the MATLAB default function for plotting colormaps. It is evident how the region between 150 and 200 seconds in the order plot, which has a relatively high amplitude value, is not depicted at the correct time in the colormap above, but it results shifted of about 50 seconds. While the colormap cannot be plotted with uneven spacing, the same can be done in a regular

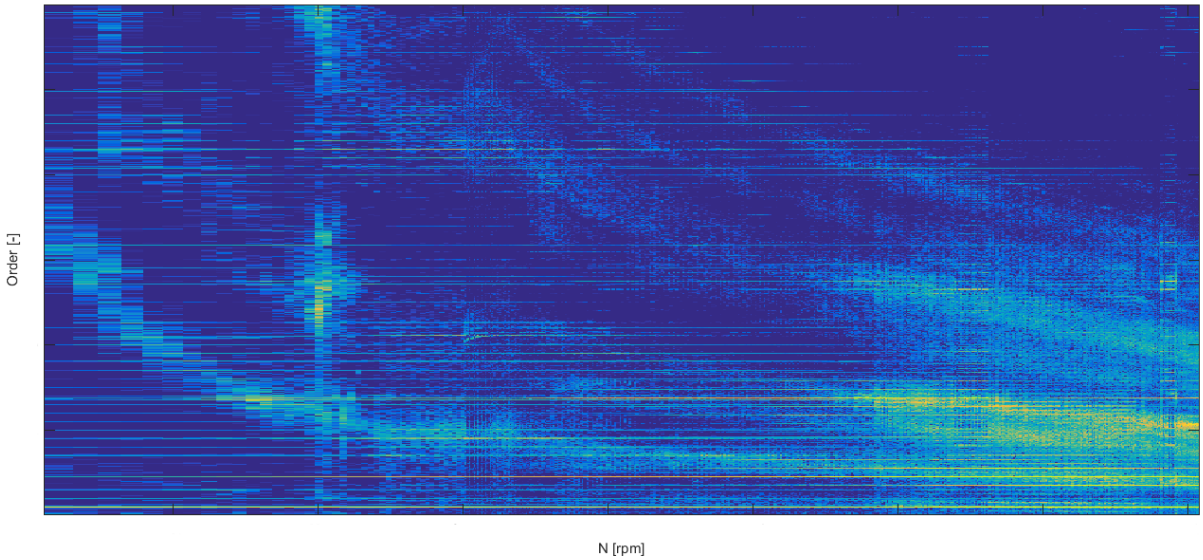


Figure 4.8: Order analysis colormap with speed x-axis

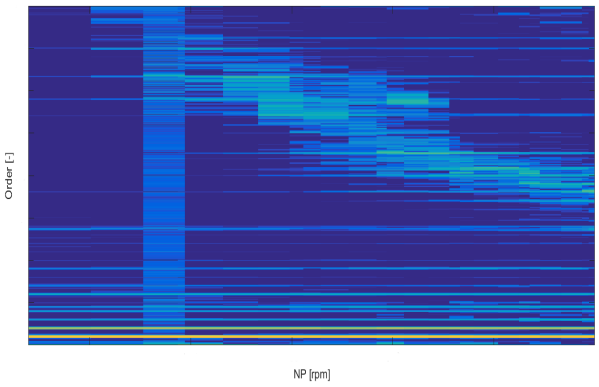


Figure 4.9: $N_{rev} = 50$ colormap zoom

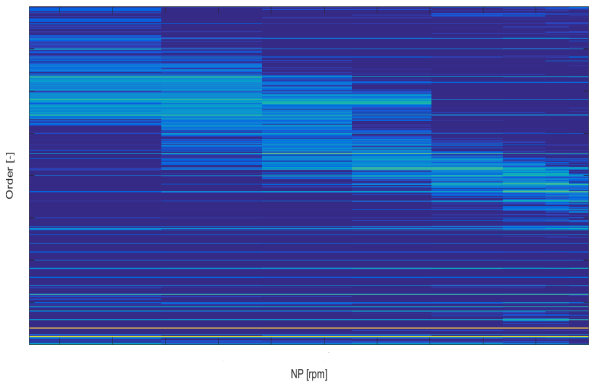


Figure 4.10: $N_{rev} = 20$ colormap zoom

2D plot, thus the data shown in the ordermap is correct, and consequently the error lies in the colormap representation of the spectrum. In figure 4.12, the same colormap has been plotted after going through the matrix stretching procedure. From that plot it is possible to see that while the resolution is lower at the beginning and the end of the test run (referring to the speed profile of figure 3.4 those are times at which the speed of the reference shaft is the lowest), the order plot and the colormap are consistent with each other, so every amplitude matrix value is assigned to the correct x-axis value.
The z-axis is a chromatic scale rather than a geometric one, for clarity purposes. Similarly to the frequency analysis colormap the z-axis represents the amplitude of the vibration. An example of the colormaps produced by the function can be seen in 4.13. In particular there are the same side plots described for the frequency analysis colormap (peakhold and order), with the same callbacks associated to the figure. The same considerations regarding the overall and the orders value made for the frequency analysis colormap still hold for the order analysis one.

4.2.3 Automatic parameter selection

The code allows for some analysis parameters to be left blank and a dedicated function has the task to fill in the missing input. In case the user has defined every parameter, the code verifies whether they are coherent with each other or not, and if they can yield a sufficiently accurate analysis. For example the maximum order requested to be displayed is always checked to be within Nyquist limit. Some parameters are immediately set by the function as it just sets a default value considered to be sufficient for a preliminary analysis, others require some more complicated operations. As for the frequency analysis particular care must be taken while setting the signal analysis parameters (windowing function, overlap, windowing period, etc.), since the precision of the results strongly depends on the appropriate choice of them. A complete list of parameters handled by the automatic setting sequence is in table 4.3.
The process is similar to the one described in chapter 3, but there are some significant differences. The most noticeable concerns the windowing period, which, as stated in table 4.1, becomes the windowing revolutions. In particular, concerning the choice of this parameter there is no W_r to find, as shown in figure 4.7. As a matter of fact, while the code has to pick the windowing revolutions it only considers the W_l, which is derived considering the two closest orders. Recalling that W_r is dependent on the df/dt of the test run, considering that a resampled signal has a null df/dt, the acceleration of the reference shaft has no influence on the colormap clarity. This means that increasing the windowing period of an order analysis will always result in a lower frequency resolution and a greater amplitude error. No other effects due to the $d\Omega/dt$ influence the colormap. This is a positive feature of order analysis, because to choose an appropriate windowing period, the function can skip the ramp finding function, saving some significantly heavy computations. The only condition that the windowing period should satisfy is being bigger than W_l, which is defined as:

$$W_l = \frac{2 \cdot Bins}{\Delta O} \tag{4.1}$$

W_l in this case is dimensionless (representing revolutions), while in the frequency analysis case it has $\Delta f = \Delta O \cdot \Omega_{min}$ as denominator, resulting in W_l having the time dimension. Not having a W_r, the windowing revolutions solely depends on the value of $\Delta Orders$. The value of W_l defines the y-axis resolution as seen from table 4.1, the closer two orders are the lower the resolution has to be to allow the graphical separation of the two. In particular the higher the windowing revolutions are the lower the order resolution.

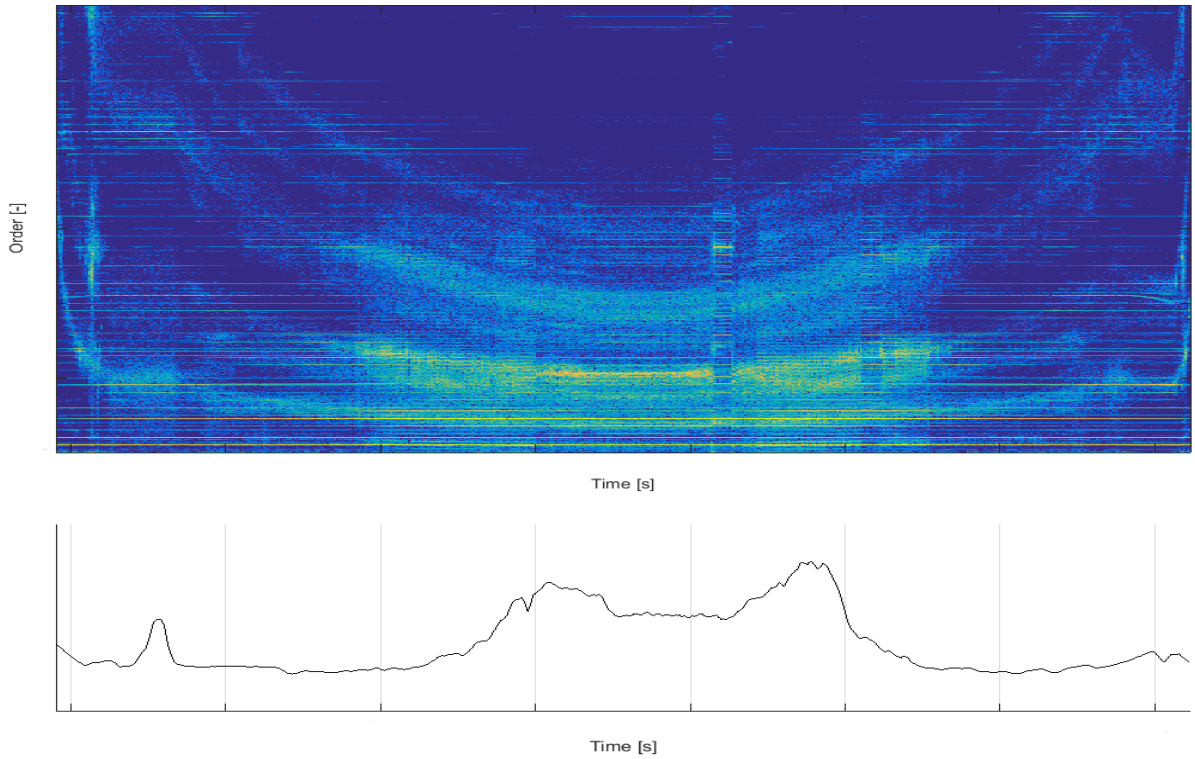


Figure 4.11: Order analysis colormap with default MATLAB colormap function

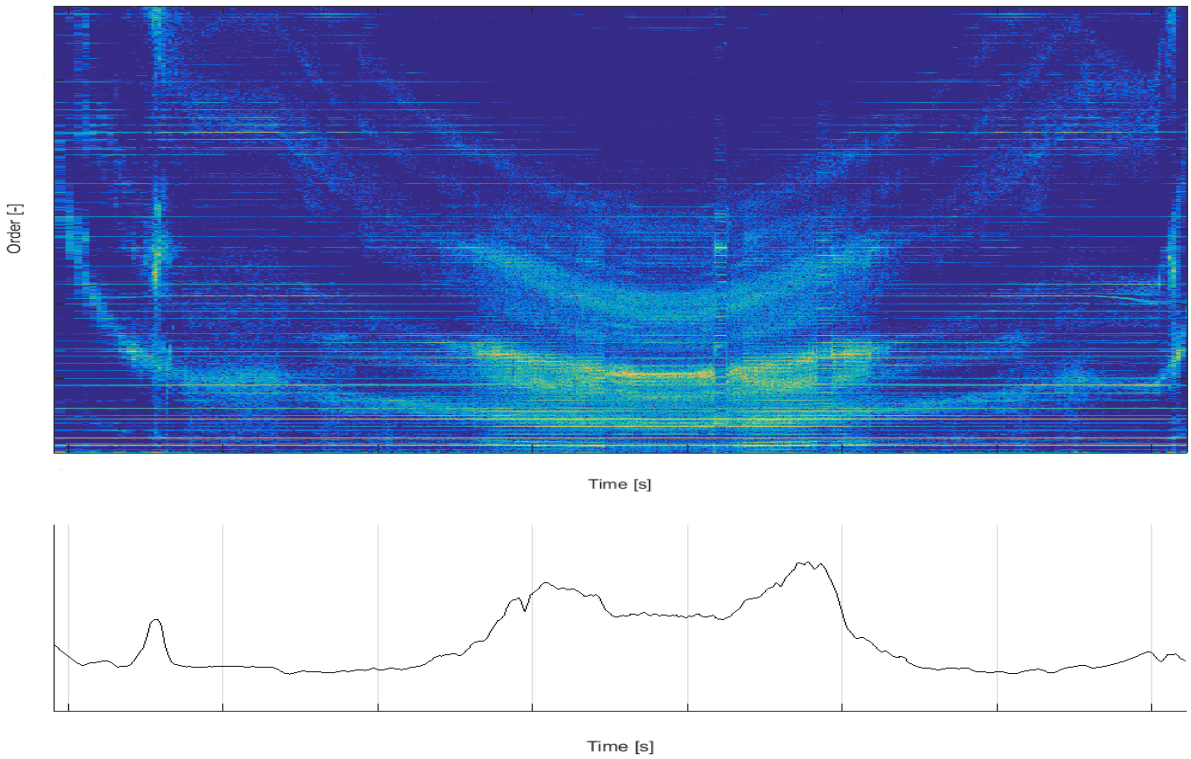
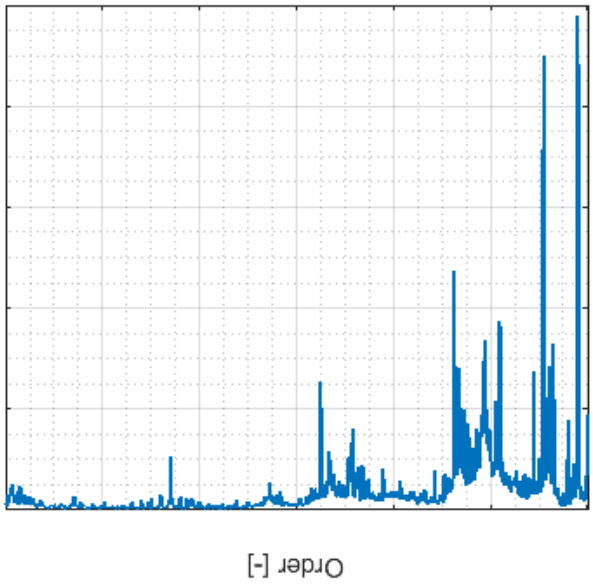
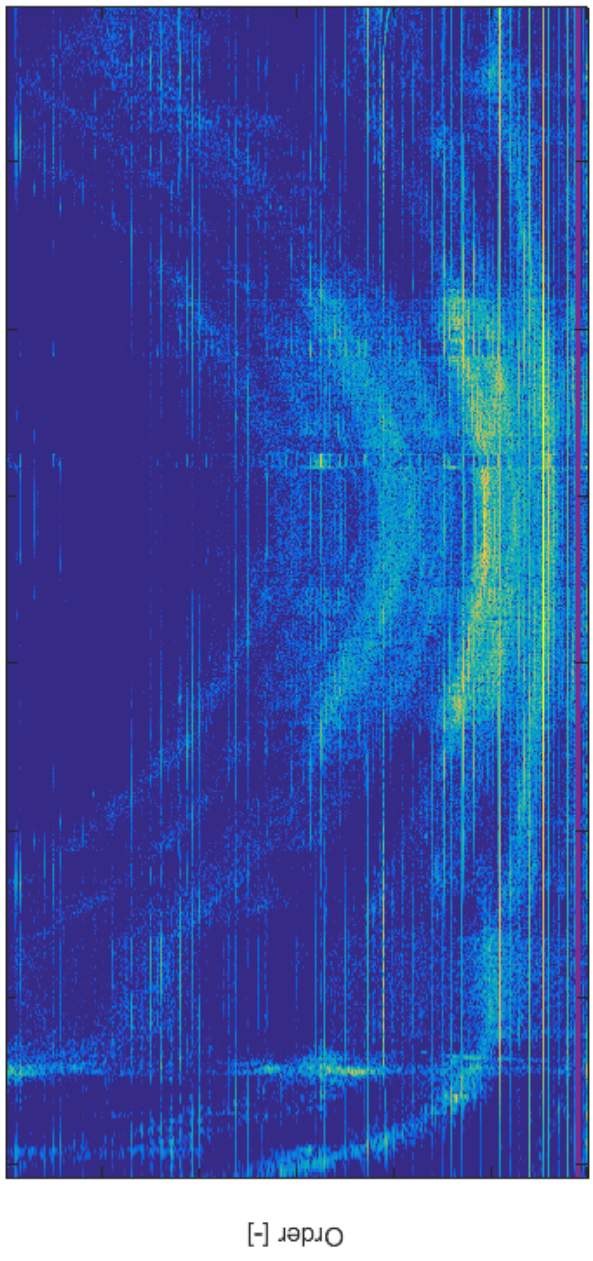


Figure 4.12: Order analysis colormap with stretched matrix evenly spacing method





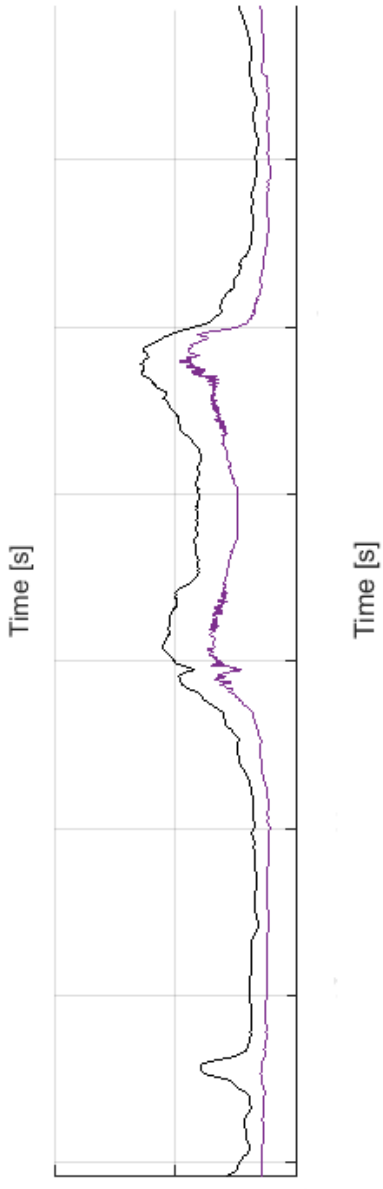


Figure 4.13: Order analysis colormap

Despite the advantages of not having a W_r to calculate, this presents another problem. In chapter 3 it is explained that in case the orders to separate are too close, there can be no available windowing revolutions. If that is the case the function automatically looks for the second smallest $\Delta Order$ and repeats the sequence for that value. This is performed within a loop so that while the condition $W_r < W_l$ holds the function keeps switching $\Delta Orders$ until it finds a $\Delta Orders$ breaking the condition. Thanks to this procedure, which inherently requires some computations, the windowing revolutions parameter is always a reasonable value yielding both amplitude precision and frequency resolution. This is not the case for the order analysis. As a matter of fact, in case the $\Delta Orders$ is made up of two really close orders the resulting windowing revolution can be extremely high since the upward limit previously posed by the W_r does not exist in this case, and this can strongly affect the amplitude precision of the resulting colormap. As a matter of fact in chapter 1 it is explained how the longer the windowing revolution is the lower the amplitude precision is.

Parameter	**Parameter meaning**	**Auto value**	**Condition to be satisfied**
Analysis focus	Specifies whether the following automatic choices will favor amplitude precision or frequency resolution	Amplitude	NA
Windowing function	Either Hanning or FlatTop windowing, depends on the analysis focus	Depends on the analysis focus	NA
Overlap	What percentage of truncated signal gets overlapped while performing the FFT	Depends on the windowing function	NA
Windowing period	Temporal length of the segments in which the signal gets truncated	Default value is $0.25\,s$	$W_l \leq W$
X-axis	Colormap x-axis, either speed or time	Time	NA
ppr	Points per revolution of the resampled signal	$4 \cdot 360$	NA
O_{max}	Maximum order whose amplitude will be displayed in the colormap	$\frac{ppr}{2.56}$	$O_{max} < \frac{ppr}{2.56}$
x-axis resolution	Resolution of the x-axis, only if speed is selected, for time x-axis this is $1/T_w$	$,/[rpm]$	Limited according to order range to prevent colormap being too heavy
Reference speed	Speed used as reference for resampling	Input speed	NA
Orders tolerance	Tolerance used to extract orders from colormap	$tol = \frac{O_2 - O_1}{O_2 + O_1} \cdot 100$	$tol \leq \frac{O_2 - O_1}{O_2 + O_1} \cdot 100$

Table 4.3: Complete list of parameters handled by the automatic parameter selection function for the order analysis

Chapter 5

Other functions

5.1 Resampling code implementation

The order analysis function described in chapter 4 calls several other subfunctions. Some of them only perform the resampling, others generate the frequency spectrum, and some are in charge of the output production. According to the nature of the raw data to be analyzed, the resampling could be useful even outside of an order analysis procedure. For example, in the following section, the signal coming from a strain gauge mounted on a gear tooth is resampled to visualize the strain peak resulting from meshing with another tooth.
To perform this task the same functions which resample the signal before the order analysis takes place are employed. This time they are not followed by the frequency analysis steps, but they pass the resampled data to a new dedicated function which has the role of rearranging the data and plot it. In particular the user has to specify the following input:

- directories concerning the raw data, and the position where outputs are saved
- in case averaging is required, the amount of revolutions to perform it
- speed and order table
- output specifications, which include the specification of which revolutions to plot as output

while on chapter 4 it is explained how upsampling the signal, and low pass filtering it prior to the actual resampling, can reduce the error of the whole process, it is also shown that the increase in computing time is more relevant than the reduction of the error [5]. Hence, the resampling function does not offer the possibility for those operations.

5.1.1 Code architecture

The code architecture is fairly simple if compared to the order or frequency analysis one, and is shown in the work flow in figure 5.1. The synchronous resampling box is composed of the passages indicated in chapter 4, and is performed by the same function used in the order analysis function.

5.1.2 Code output

The output produced by the function is a plot showing the raw data resampled concerning specific revolutions. The user cannot directly request for a specific revolution to be plotted

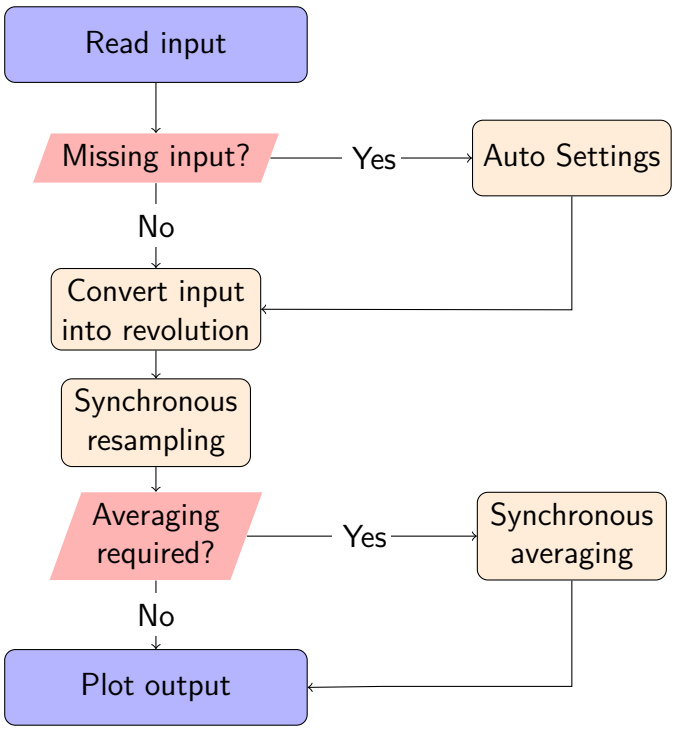


Figure 5.1: Resampling function work flow

though, since it would be not practical as the data are usually shown as function of time or speed. Rather, the input of this function is either time or speed. The code then finds the closest revolution to either the time instant or speed requested, and plots the data. In case the user inputs a time instant, the associated revolution is easily found, since the resampling function generates a vector called *synchronizing pulse* associating the time at which every revolution begins. Instead, if the user inputs a rotational speed, depending on the run test speed profile, the reference shaft can achieve the indicated value for several times. In that case the function warns the user that for such speed more than one revolutions are found, and plots the first one in a time order.
The user can require several revolutions to be plotted, this can be done both in the same plot or in separate plots, according to the user preference. The function also applies synchronous averaging if requested, which will be discussed in the following section in relation to the test case. The x-axis resolution of such plots is controlled by the points per revolutions (*ppr*), which is not bound to any other parameter. The only concern deriving from extremely high *ppr* resides in the increase of computational time and weight of the figure file.
In case of multiple revolutions being plotted in the same figure, the code associates the visibility toggling callback to the figure, to allow for an easier interpretation of the figure. An example of the output produced by the function is shown for the test case in figure 5.7.

5.1.3 Resampling of a meshing tooth

As an example of the application for the resampling function, let us consider a strain gauge mounted on the base of a gear tooth which meshes once per revolution with another gear tooth. The signal registered by the strain gauge is the one in figure 5.2. For instance, at the beginning of the test run it is possible to see a time period in which the strain gauge measures extremely high average strain levels which, if they were true, would certainly result in the failure of the tooth. Such data are obviously the result of some kind

of measuring error. It is not unusual to find measurement errors in data obtained through operative testing, handling of such cases is troublesome, since it is difficult to instruct a code to recognize whether data are just out of ordinary, but still plausible, or an obvious error. In order to handle such cases, the code has a cut function, which simply removes the data outside of a specified time interval. Obviously this requires the user to be aware of such portions to be removed, but generally it is quite easy even for an inexperienced user to identify those regions.
Whenever a test run is conducted at a variable angular speed the only way of visualizing the cyclic deformation of such tooth is to resample the signal with respect to the gear shaft rotational speed, and sometimes, according to the phenomenon to visualize, perform some averaging over a number of revolutions to filter non-synchronous noise affecting the signal. To perform a resampling, it is required to have the speed data since from such

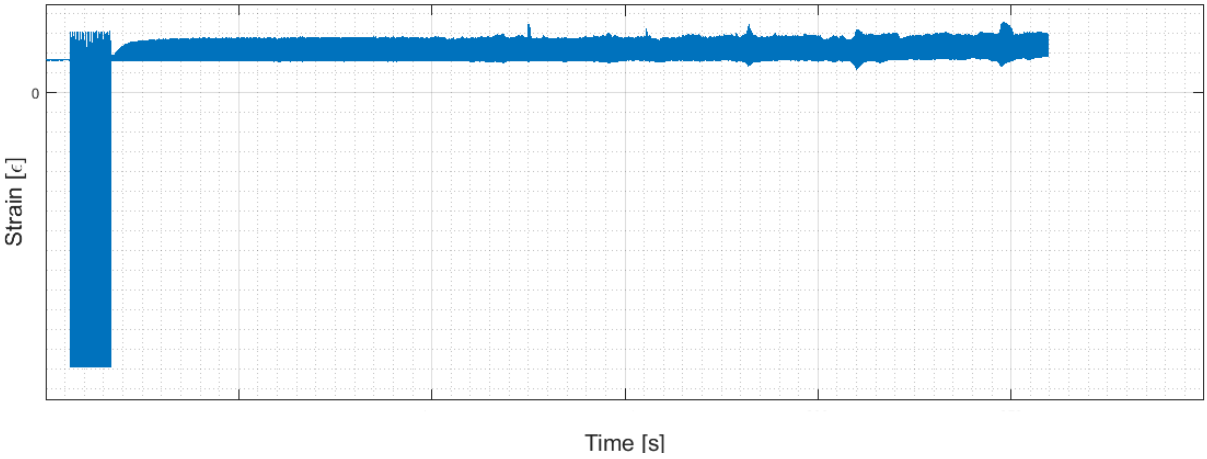


Figure 5.2: Raw strain gauge signal

data the *synchronizing pulse*, the vector indicating the start of every revolution of the test run, is obtained. As stated in section 4.1 there can be some errors affecting the conversion from voltage data to speed data (or tacho), in this section the effect and the nature of one of those possible errors is discussed.
Despite the velocity data not being generated by a dedicated key-phasor, the signal resulting has a fairly low noise level, and can be analyzed in order to yield the needed information about the angular speed. In figure 5.3 a zoom of the instant between 20 and 20.1 seconds, the uppermost plot displays the raw tachometer data, which is not regularly shaped, like the signal in figure 4.5. In particular, neither the shape of the signal nor the actual amplitude are related. In this test case the signal probably is not coming from an optical device since the amplitude oscillates between both positive and negative values. In order to extract the speed information from the raw velocity data the latter has to be converted from analogical to digital, therefore the final signal only portrays an amplitude of either 1 or 0. In order to perform a correct resampling, the tacho signal goes through a Schmitt trigger which has been illustrated in section 4.1. The correct selection of the two thresholds of the filter are of paramount importance for a successful digitalization of the signal. In figure 5.3 the upper threshold is set to a value of 2000, and the lower is simply $th_l = 0.8 \cdot th_h$. The reason behind the choice of the threshold value resides in finding the cleanest crossing between signal and threshold lines. To better explain this concept let us suppose to set the threshold for the same signal to a much lower value such as 1. The resulting tachometer signal with relative digital signal are shown in figure 5.4 (the two threshold lines have merged together because of the scale of the ordinate axes). At first glace it is clear that, despite depicting the same time interval, the two plots for the digital signal are visually different, for instance the signal in figure 5.4 spends more time

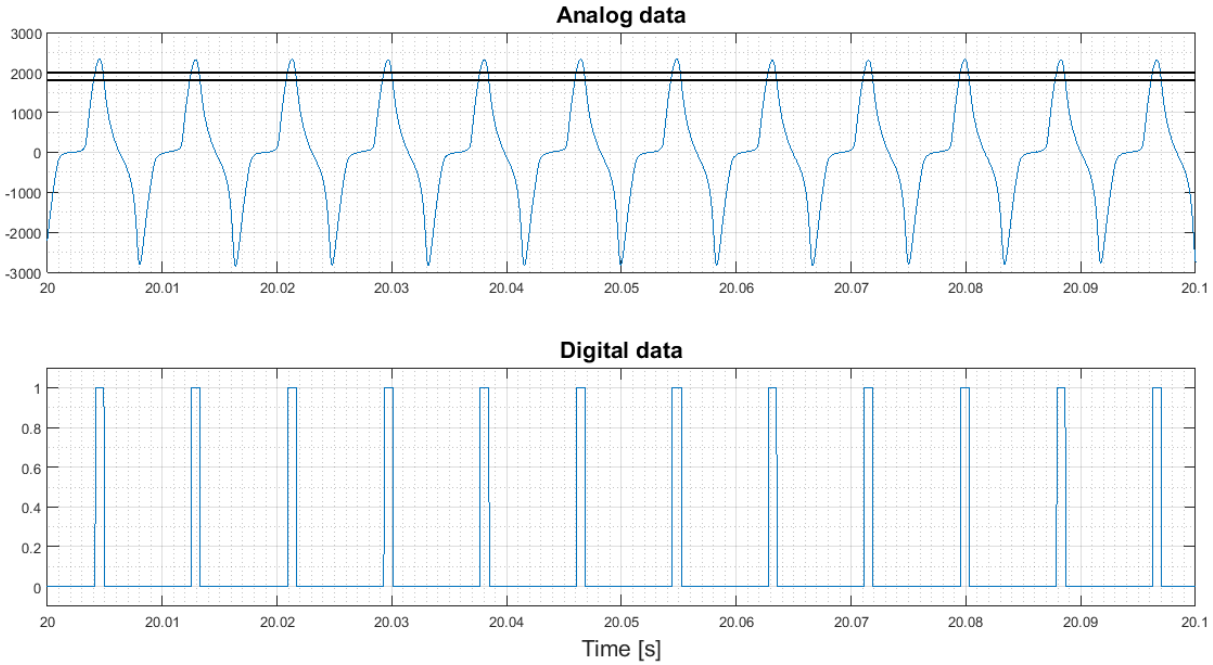


Figure 5.3: Tachometer analog data converted into digital with Schmitt trigger with threshold set to 2000

being null than the one in figure 5.3, that is due to the difference in threshold value. In particular, since the analogue signal becomes a digital 1 only if its amplitude is bigger than the upper threshold th_h, raising the threshold restricts the interval of samples for which said relation is true, hence the smaller size of the time region where the digital signal is 1. Besides, in the interval of time between 20.08 and 20.09 of figure 5.4 the signal

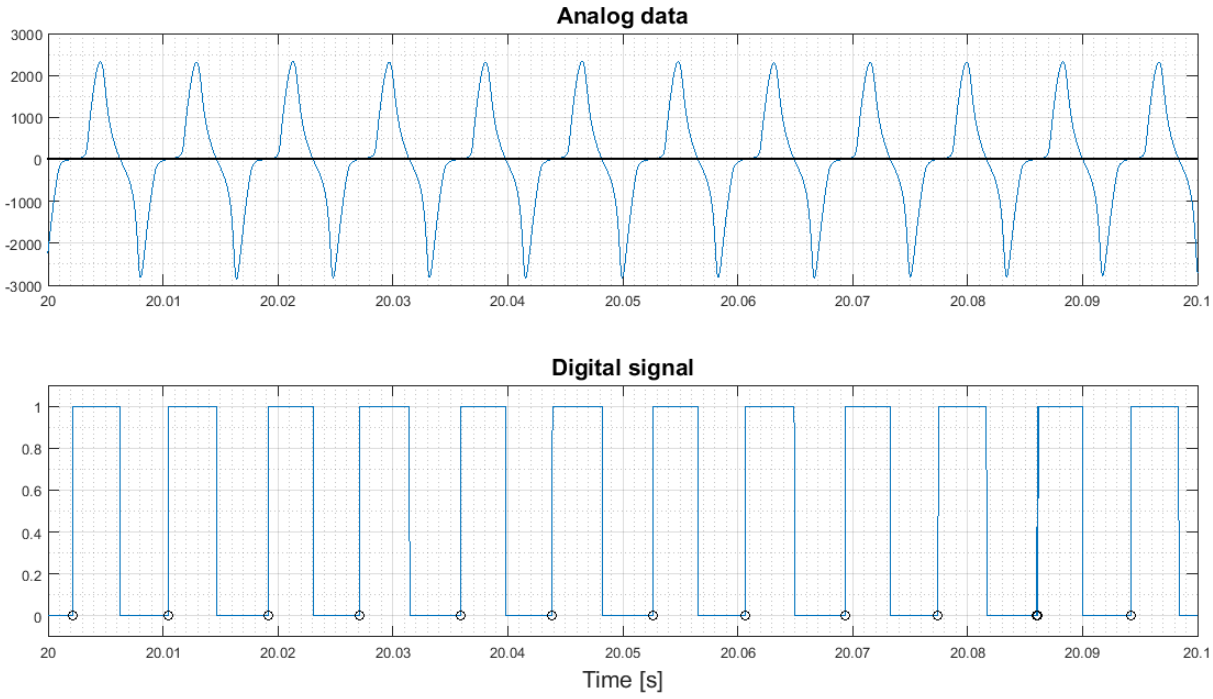


Figure 5.4: Tachometer analog data converted into digital with Schmitt trigger with threshold set to 1. The circles indicate the beginning of the digital signal regions converted to 1 by the Schmitt trigger

seems to quickly raise to one, descend to zero, and climb back up to one. Upon closer

inspection (figure 5.5) it is possible to see that the low difference value of upper and lower threshold allows for random signal oscillations to cross both of them resulting in a wrong conversion by the Schmitt trigger which registers a crossing as if a phonic wheel tooth crossed the optical sensor. Considering that this was found in an interval of $0.1\,s$ in a test run of several minutes this error can prevent the resampling from being successful by producing a *synchonizing pulse* vector not representative of the true times at which the revolutions begin. In this particular case of a strain data of a meshing tooth, this kind of error causes the output plot to show several peaks, corresponding to the meshing, for every revolution.
The higher threshold value displayed in figure 5.3 allows for the crossing to happen in a region where the signal is steeper and therefore has less of a chance of behaving similarly to figure 5.4. Moreover the higher threshold value increases the difference between the upper and the lower threshold (with an upper threshold of 2000 the lower becomes $0.8 \cdot 2000 = 1600$), forcing the noise to cause an amplitude difference of 400 instead of 0.2. Despite this fact there still is some chance of having some conversion error since the nature of the noise cannot be predicted. For this reason, in addition to selecting the appropriate threshold, some filtering could be done in order to detect and delete faulty crossings. For example the filter could eliminate crossings closer than a given time, or equivalently it could delete crossings with too few samples in between them. This should be performed carefully enough not to delete also valid crossings. It is generally a good idea to at least have a look at the tachometer data before setting any threshold or crossing filters. As a matter of fact signals like the one in figure 4.5 do not need any crossing control since they are clean enough to yield a correct speed reading, which could actually be ruined by applying a crossing filter.
This kind of measures helps in conjunction with an appropriate threshold selection, but they are not enough by themselves. It might also be useful to perform a moving average on the tachometer data, being careful not to pick an averaging window too large which would result in an excessive modification of the tachometer signal. In figure 5.6 the difference between the two speed plots obtained with the two different threshold values. The two plots are very different at the beginning of the test run, that is because before the $10\,s$ mark the tachometer signal is very noisy and a threshold equal to 1 picks up all the noise and converts it in erroneous teeth passages. As stated before in section 4.1 a revolution is defined by z crossings where z is the number of teeth of the phonic wheel (ppr), hence marking a crossing where there should not be one causes wrong speed readings.
Once the strain data are synchronized with respect to the revolution of the gear, it should depict a peak always at the same angle representing the meshing of the tooth with the corresponding one from the other gear, while the actual magnitude of the peak can vary due to variations in input torque and rotational speed. The only variation of the peak position is the one caused by torsional deformation on the shaft, but should be of very low entity. This is true even if the speed varies in the test run as the synchronous resampling removes the df/dt of the signal on which it is performed.

5.1.4 Synchornous averaging

Once the speed data are correctly analyzed and the strain signal is resampled synchronous with the revolutions of the gear, it might be useful to perform a synchronous averaging, especially if the user is trying to visualize a phenomenon masked by noise.
Synchronous averaging consists in performing the average of the acquired samples of data over a window representing the cyclic nature of the phenomenon that is being investigated. For example in the case taken into consideration, the phenomenon is cyclic with respect

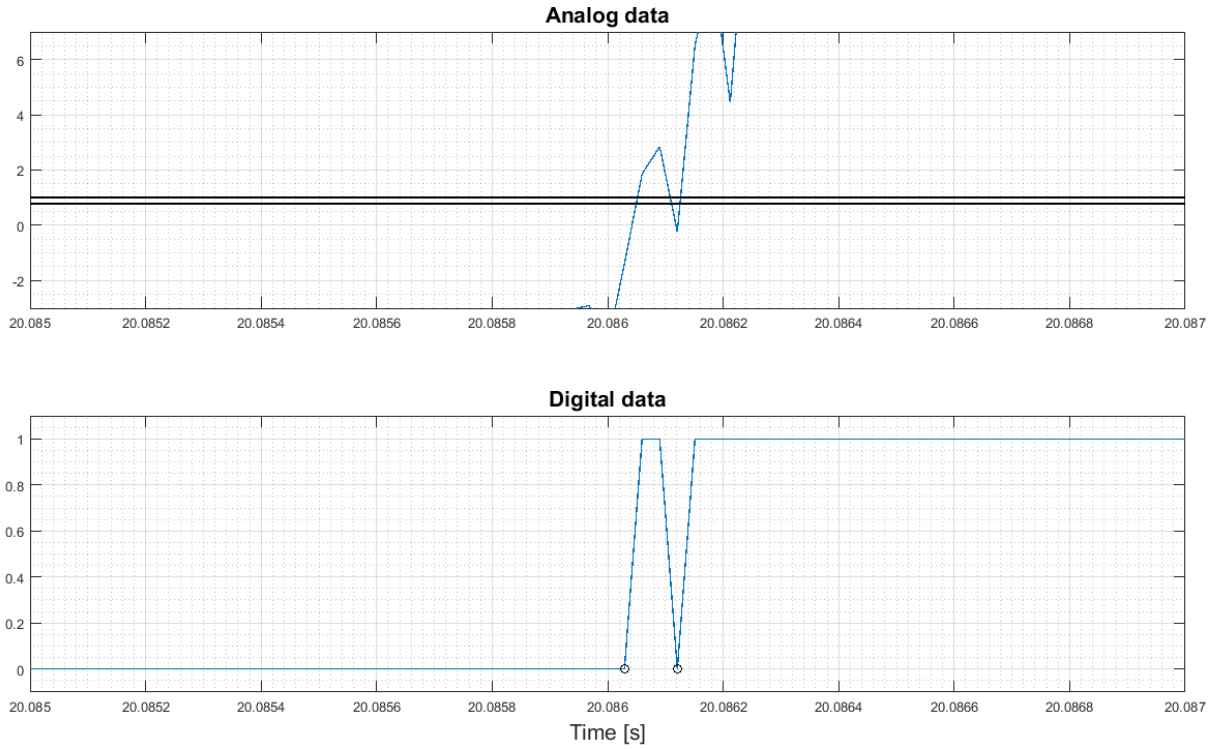


Figure 5.5: Tachometer analog data converted into digital zoomed at time interval 20.085 s to 20.087

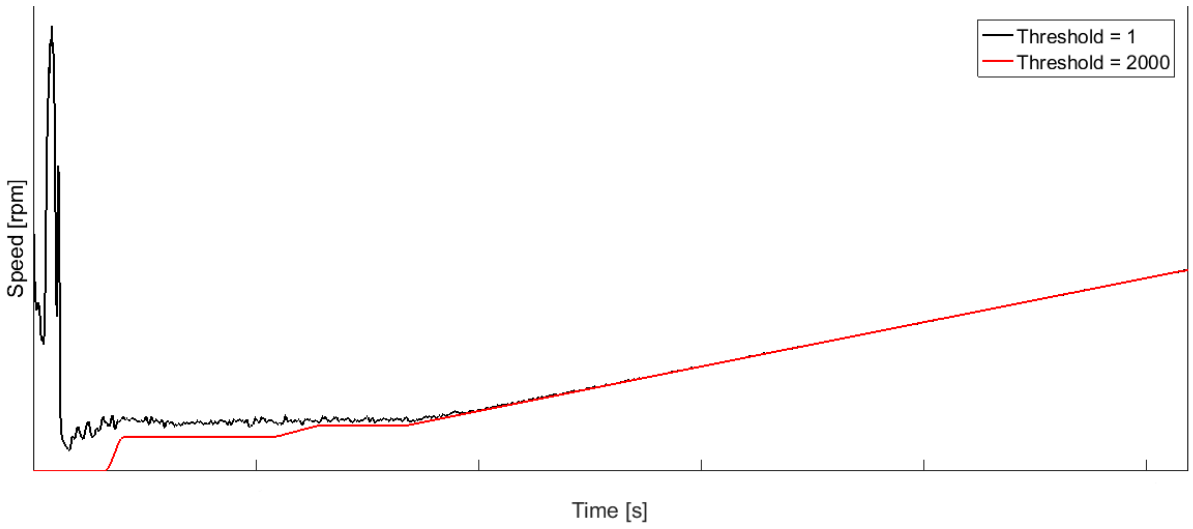


Figure 5.6: Different speed plots resulting from various threshold values

to a single gear revolution, hence the window would be a number of samples representing any integer number of revolutions. Obviously this operation produces significant results only when the signal is resampled so that within the time interval of each revolution there is a constant amount of samples, and if the noise is not synchronous with the phenomenon being investigated.
Averaging yields a smoothing of the final signal, it can be considered a form of lowpass filtering, as a matter of fact, about the filtering properties of synchronous averaging it can be stated that the standard deviation of any non synchronous signals is proportional to the inverse of the square root of the number of averages performed [8]. So increasing the amount of windows used in the averaging increases the smoothing of the signal. To better understand this point let us consider the example so far discussed. Since the sensor is mounted on a gear tooth, the expected plot of its strain as a function of the angular position of the tooth over a revolution should depict one single peak per revolution, corresponding to the meshing as shown in figure 5.7. In the figure the two peaks show significantly different amplitudes due to the fact that the test run was not conducted at constant torque, while the slight difference in angular position is probably due to some error in the tachometer signal digitalisation and also to different angular deformation due to different torques transferred by the two teeth. Because of the synchronous resampling the number of samples plotted remains constant varying the revolution, also the peak angular location remains constant as the relative position of the two meshing gears does not change over time. This means that averaging a number of samples relative to different revolutions will still display a single peak at the same angular position. On the other hand, any random noise or non synchronous signals in general are filtered as they do not show the same amplitude and angular position in every revolution considered while averaging. In figure 5.8, it is shown the strain data of the same revolution than the ones in figure 5.7 averaged over 500 revolutions, which corresponds to a time interval of about 10 seconds. The noise of the orange plot has a noticeably lower amplitude, moreover the averaged plot seems to show another peak of smaller amplitude compared to the one caused by the meshing at around $200\,deg$. This smaller peak is masked by noise in figure 5.7 and has been made visible thanks to the synchronous averaging filtering. In addition, while the orange plot in figure 5.7 shows some irregularities, after the filtering the peak becomes regular, meaning that the source of that irregularity was not synchornous with the teeth meshing, thus was likely some sort of noise.
Performing this operation on code is fairly simple once it is defined the vector of synchronously resampled data. As a matter of fact, once the point per revolution (ppr) is defined, the code performs a moving mean with a window consisting of ppr samples and repeats the operation for as many windows as they are requested by the user.
This was just one example of the resampling applications. Many more exist, for example by measuring an acceleration data of a meshing gear shaft, and resampling it, it is possible to see meshing irregularities due to teeth defects by averaging over a number of revolutions equal to the lowest common denominator of the two gears number of teeth. Example of this procedures can be found in literature [8].

5.2 Averaging code implementation

In some cases looking at the average signal of the raw input can be beneficial. For instance from no other code function the average value of the raw signal is generated. In both the functions dedicated to frequency and order analysis, the overall is produced, representing the energy content of the signal, but in both cases, before generating the overall, the raw signal gets its mean value removed. This operation is required to remove the zero

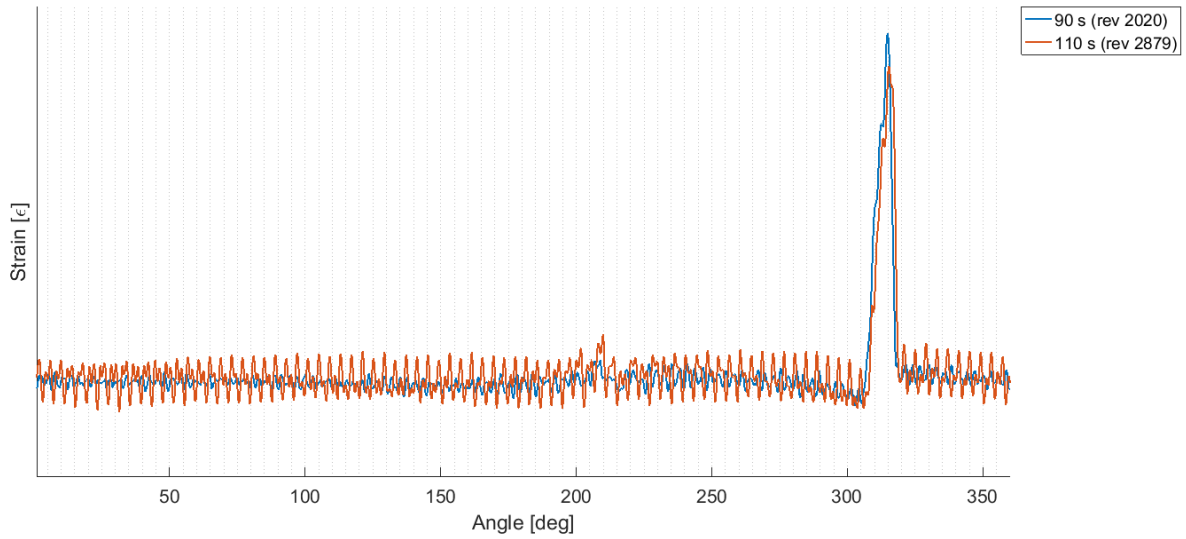


Figure 5.7: Resampled data at two different revolutions

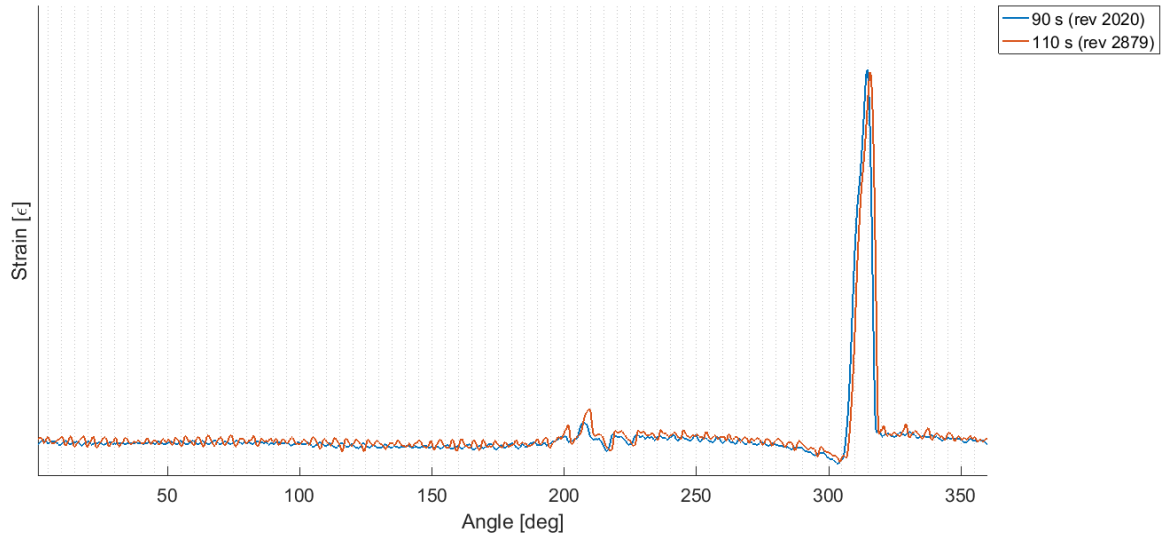


Figure 5.8: Resampled data at two different revolutions, averaged over 500 revolutions

frequency components from the signal, which are not interesting from a frequency analysis standpoint, and, due to its typically high amplitude, could mask some other frequencies amplitudes. Thus, a function dedicated to implementing the raw signal average has been written.
The function performs the moving average of the input signal with a window which can be specified or left black to be automatically set by the dedicated function. The input requested by the function are:

- directories concerning position of input data, and position where output plot is saved
- averaging parameters
- output specifications, such as x-axis unit

The averaging parameters mentioned in the list are the same parameters required to window the signal (windowing period and overlap), extraction, interpolation methods (in case of a speed x-axis), and averaging window size. To perform the averaging the function divides the raw signal into many segments of time length equal to T_w, performs the average of the segment elements, and saves the number obtained into a newly defined output vector. If needed the segments can be overlapped by the specified value. This passages are the same passages performed to obtain the x-axis vector of the colormap produced by the frequency analysis function described in chapter 3. Instead of performing the average of the signal segments elements, the function can perform the root mean square of the signal.

5.2.1 Code architecture

In figure 5.9 the work flow of the averaging function is shown. The function is fairly simple, however, with the wrong settings it can result quite time consuming, since both MATLAB average and interpolation function are not particularly efficient.

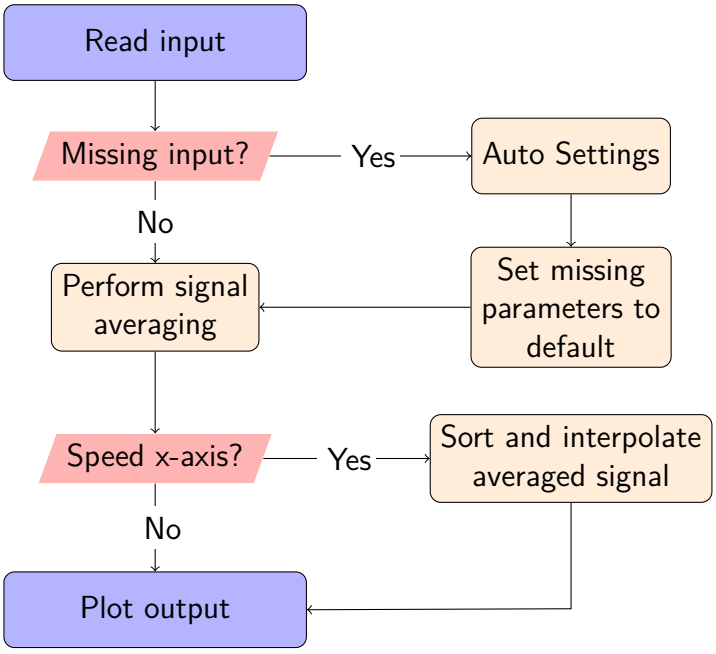


Figure 5.9: Signal average workflow

5.2.2 Code output

Once the output vector has been generated, it is passed to the dedicated output function. The output function creates a figure depicting the output vector as function of either time or speed, according to the user request. In particular, in case of a speed x-axis, the interpolation described in chapter 3 is not required. As a matter of fact, since MATLAB 2D plot function supports vector unevenly spaced, only the sorting by speed is required. This inherently is a positive aspect, which makes the averaging function really fast and light from a computational standpoint.

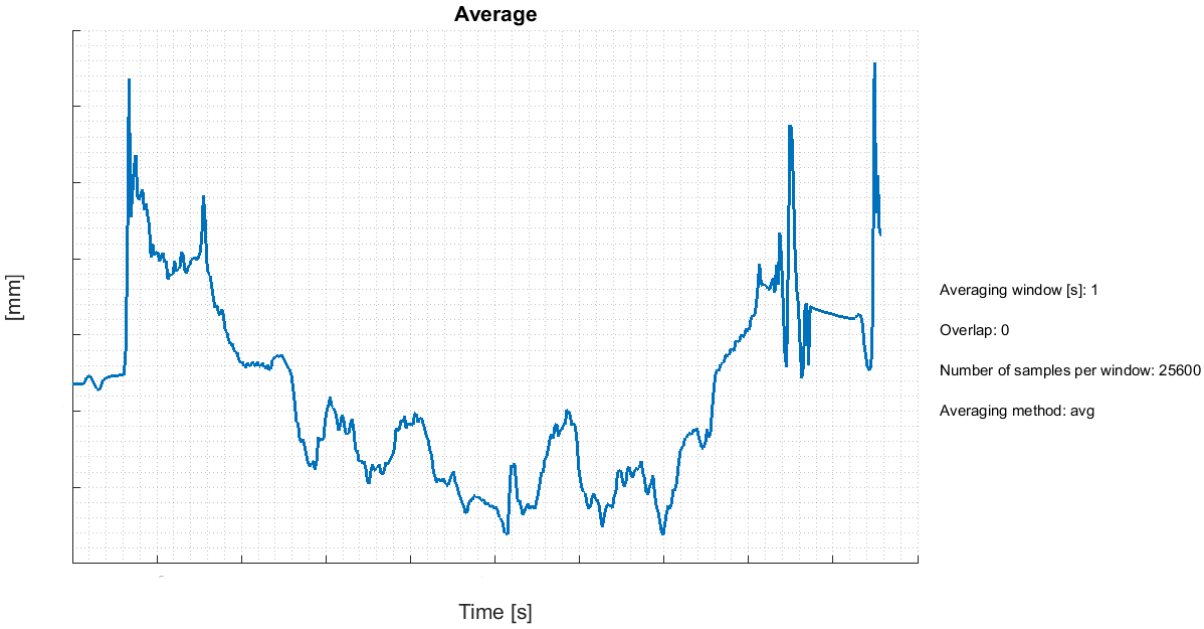


Figure 5.10: Average function output example

Conclusions

In this work the main tools to analyze vibration experimental data are exposed, such as frequency and order analysis. Particular emphasis has been given to explaining how such analysis can be carried from a practical standpoint by means of a MATLAB code. Such code has been written following two guidelines:

1. time optimization
2. ease of use

Concerning the first point, the code has achieved an almost full time optimization. It is able to recognize when different jobs require the same data and sorts them correctly in order to save memory and time while executing. The most time consuming instructions are by far the data reading from the files and the output saving, hence from a time optimization point of view not much more can be done. To give some examples of the time duration of the analysis, a single frequency analysis can take from a few seconds for raw data lighter than 1 GB, to around 20 minutes for files weighting several GB. Regardless of the file size, data reading is always the longest operation to perform in the analysis.
As stated in the second point, the code has been written for an experienced user to be able to obtain the Campbell diagram of the raw data. The code is able to set the main analysis parameters such as windowing period, windowing function, overlap, points per revolution and many others required to perform the analysis. In particular to choose the windowing period the code must gain knowledge of the reference shaft acceleration, which is not a simple task to be carried out automatically by an algorithm. As a matter of fact, future works should address the stability of such procedure with the focus of finding a more robust algorithm for filtering the speed data.
Despite the code has been developed for inexperienced users, it also allows for expert users to set manually every parameter. The code however still verifies that such values are in compatible with a successful analysis (e.g. it checks whether the colormap will include frequencies beyond Nyquist).
Future developments should include the creation of a GUI (graphical user interface) able to speed up the input generation part, which is currently fairly lengthy since it includes both analysis parameters and colormap options.

Bibliography

[1] S. Marchesiello and A. Fasana. *Meccanica delle vibrazioni*. Ed. by Clut. Clut, 1992.

[2] Paresh Girdhar. *Practical Machinery Vibration Analysis and Predictive Maintenance*. Ed. by Cornelius Sheffer. Elsevier, 2004.

[3] Audrey F. Harvey and Michael Cerna. "The Fundamentals of FFT-Based Signal Analysis and Measurement". In: 1993.

[4] Letizia Lo Presti. *L'analisi dei segnali*. Ed. by Clut. Clut, 1992.

[5] Enrico Geninatti. "Analisi di segnali sperimentali in scatole ingranaggi per motori aeronautici". MA thesis. Politecnico di Torino.

[6] S. Gade et al. *Order Tracking Analysis*. Tech. rep. BrÃijel Kjaer, 1995.

[7] Jason R. Blough. "Adaptive Resampling - Transforming From the Time to the Angle Domain". In: *Michigan Technological University* (2006).

[8] Jiri Tuma. *Vehicle Gearbox Noise and Vibration*. Willey, 2014.