Biological Signal Analysis

Ramaswamy Palaniappan

Ramaswamy Palaniappan

Biological Signal Analysis

Biological Signal Analysis
1st edition

ISBN 978-87-7681-594-3

Contents

Preface

The aim of this book is to provide readers with a fundamental understanding of signal processing techniques and classification algorithms for analysing biological signals. The text here will allow the reader to demonstrate understanding of basic principles of digital signals; awareness of physiology and characteristics of different biological signals; describe and apply pre- and post- processing techniques, such as conditioning, filtering, feature extraction, classification and statistical validation techniques for biological signals and solve practical biological signal analysis problems using MATLAB.

Final year undergraduates and graduates students in any field with interest in biological signal analysis (and related areas like digital signal processing) are the main target audiences. But the book will also be useful for the researchers in both industry and academia, especially those from non-technical background who would be interested in analysing biological signals – the text does not assume any prior signal processing knowledge and MATLAB is used throughout the text to minimise programming time and difficulty and concentrate on the *analysis,* which is the focus of this book.

I have tried to follow a simple approach in writing the text. Mathematics is used only where necessary and when used (and where possible), numerical examples that are suitable for paper and pencil approach are given. There are plenty of illustrations to aid the reader in understanding the signal analysis methods and the results of applying the methods. In the final chapter, I have given a few examples of recently studied real life biological signal analysis applications.

I hope I have done justice in discussing all four related sections to biological signal analysis: signal preprocessing, feature extraction, classification algorithms and statistical validation methods in this one volume. But by doing so, I had to skip some theoretical concepts which are not mandatory for implementing the concepts and I hope the learned ones will forgive these omissions.

I would like to acknowledge the efforts of my students, John Wilson, Cota Navin Gupta and Tugce Balli for their comments in various parts of the book. For over a decade, I have greatly benefited from discussions with students and fellow colleagues who are too many to name here but have all helped in one way or another towards the contents of this book. I must thank my wife and daughter for putting up with all the weekends and nights that I *disappeared* to complete this book. Finally, I trust that my proofreading is not perfect and some errors would remain in the text and I welcome any feedback or questions from the reader.

Ramaswamy Palaniappan
April 2010

About the author

Dr Ramaswamy Palaniappan
BE, MEngSc, PhD, SMIEEE, MIET, MBMES
School of Computer Science and Electronic Engineering
University of Essex, United Kingdom

Ramaswamy Palaniappan or fondly known as *Palani* among friends, received his first degree and MEngSc degree in electrical engineering and PhD degree in microelectronics/biomedical engineering in 1997, 1999 and 2002, respectively from University of Malaya, Kuala Lumpur, Malaysia. He is currently an academic with the School of Computer Science and Electronic Engineering, University of Essex, United Kingdom. Prior to this, he was the Associate Dean and Senior Lecturer at Multimedia University, Malaysia and Research Fellow in the Biomedical Engineering Research Centre-University of Washington Alliance, Nanyang Technological University, Singapore.

He is an expert reviewer for FWF Austrian Science Fund, Collaborative Health Research Projects Program, Natural Sciences and Engineering Research Council of Canada and Industry Grant Scheme, Malaysia. He founded and chaired the Bioinformatics division in Centre for Bioinformatics and Biometrics in Multimedia University, Malaysia. His current research interests include biological signal processing, brain-computer interfaces, biometrics, artificial neural networks, genetic algorithms, and image processing. To date, he has published over 100 papers in peer-reviewed journals, book chapters, and conference proceedings.

Dr. Palaniappan is a senior member of the Institute of Electrical and Electronics Engineers and IEEE Engineering in Medicine and Biology Society, member in Institution of Engineering and Technology, and Biomedical Engineering Society. He also serves as editorial board member for several international journals. His pioneering studies on using brain signals for brain-computer interfaces and biometrics have received international recognition.

Ramaswamy Palaniappan
April 2010

1 Introduction

Biological signal analysis[1] encompasses several interdisciplinary topics that deal with analysing signals generated by various physiological processes in the human body. These signals could be electrical, chemical or acoustic in origin and an analysis of these signals are often useful in explaining and/or identifying pathological conditions of the human body. However, these signals in their rawest form do not provide much information and therefore, the motivation behind biological signal analysis is to extract (i.e. to reveal) the relevant information. This analysis has become even more important with modern healthcare striving to provide cost effective point-of care diagnosis and personalised treatment. Furthermore, fast computing power in recent years has made much of the more complex analysis methodologies possible. The purpose of this chapter is to provide an overview of biological signal origins and describe commonly encountered biological signals.

1.1 A Typical Biological Signal Analysis Application

Before we delve into the analysis of biological signals, it would be useful to understand that they are often represented as discrete in time. For example, Figure 1.1 shows an example of a sinus rhythm electrocardiogram (ECG), which represents the electrical activity obtained from normal heart. A single measurement of the signal x is a scalar and represents the electrical signals generated by the mechanisms in the heart (the details of this process follow later) at a particular instant of time t (denoted with index n) rather than at all points of time (the involved sampling process to obtain discrete time measurements from continuous signals will be discussed in the following chapter).

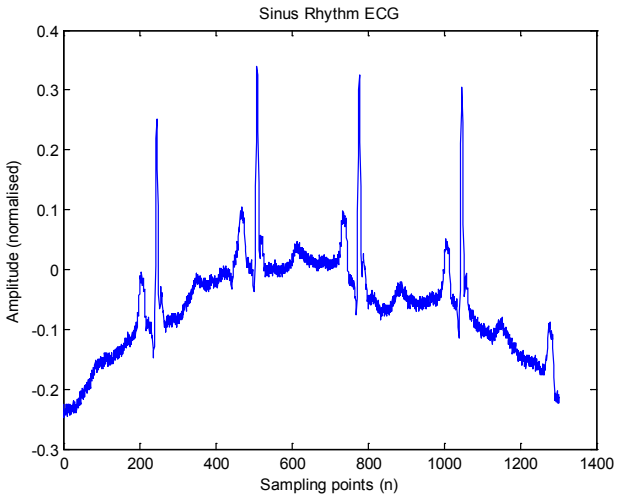


Figure 1.1: Normal sinus rhythm ECG. ECG data used here was obtained from [1].

There are two types of noise inherent in this signal: baseline and powerline interference. Figure 1.2 shows a *cleaned* version of Figure 1.1 obtained through band-pass filtering, which is a subject of discussion in Chapter Four. This figure also shows a zoomed version of the ECG representing one beat of heart. The sharp peaks in the signal denote the occurrence of what is known as the R wave and the time intervals between consecutive R-R peaks would be useful to measure the heart rate, i.e. the number of times the heart beats in a minute.

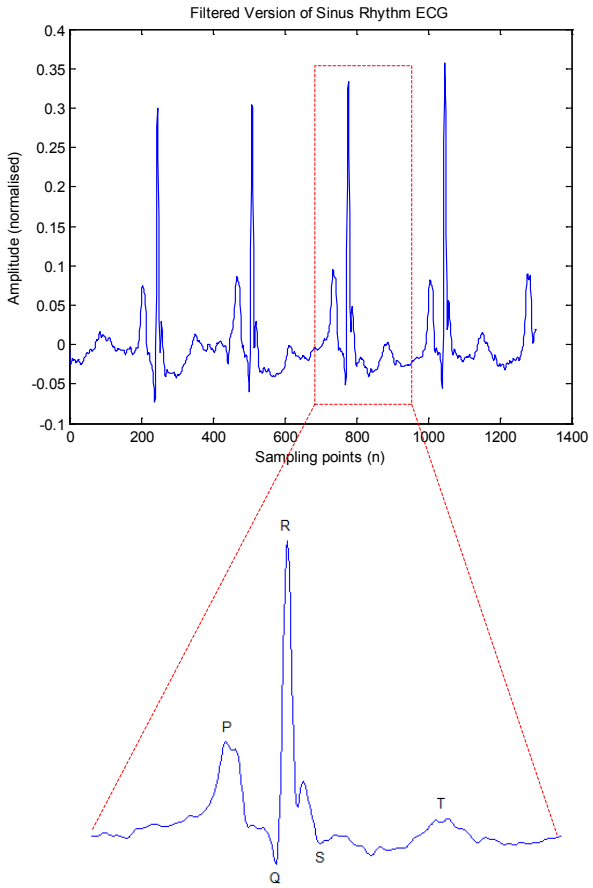


Figure 1.2: Filtered version of sinus rhythm ECG in Figure 1.1.

Similarly, the segment from the Q wave to the S wave (more commonly known as QRS segment), is useful in indicating certain pathological[2] variations in the heart's electrical system. For example, Figure 1.3 shows an ECG waveform from a subject with an ectopic beat of Premature Ventricular Contraction (PVC) type. Ectopic beats are premature beats with a complex waveform that can occur occasionally in most people. However, the presence of frequent ectopic beats (more than six per minute) could indicate a serious fatal problem if left uncorrected. While it is visually obvious that the QRS segment for the PVC beat is different from Figure 1.1, it is impractical to sit and manually detect the occurrences from an ECG chart (or on-screen monitor) of a patient over many hours/days. Rather, a computer that is trained to automatically analyse and detect the occurrence of such beats using signal processing algorithms is employed. This is a typical biological signal analysis application. Figure 1.4 shows the typical signal analysis methodologies involved that would be necessary for such a computer based detection of ectopic beats; the details of which will be discussed generally in the forthcoming chapters and specifically in the final chapter.

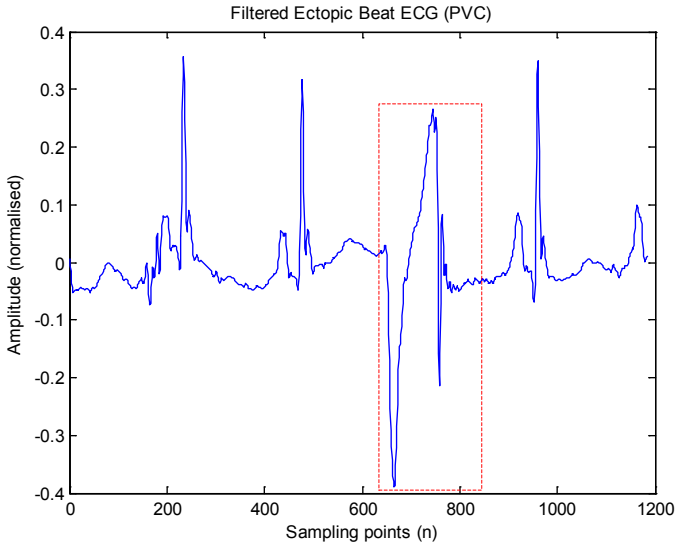


Figure 1.3: ECG which exhibits a PVC ectopic beat (in red block). ECG data used here was obtained from [1].

Figure 1.4: Typical steps for computer based ectopic beat detection.

1.2 Examples of Common Biological Signals

Commonly encountered biological signals that describe the electrical activity of the brain, heart, muscles etc will be introduced in this section. Some of these signals like ECG and electroencephalogram (EEG) are spontaneous activity of the human body while others such as evoked potentials are signals in response to external stimulus, for example the presentation of visual stimuli which results in visual evoked potential (VEP) signals. While some analysis procedures are common (like using filters to extract components in specific frequency range), the different properties of these biological signals do call for the application of widely varying analysis procedures. Hence, it is useful to know the fundamental details of how these signals are generated before studying analysis procedures for extraction of the required information.

1.2.1 Electrocardiogram

The ECG is a representation of the electrical activity of the heart and the cardiac rhythm is controlled by the pacemaker cells known as sinoatrial (SA) node. The PQRST waveform (as shown in Figure 1.5) represents one complete ECG cycle: P wave occurs when SA node fires and the impulse spreads across atria and triggers atrial contraction; PQ interval (isometric segment) is the propagation delay when the impulse travels from atria to ventricles (allowing blood flow to complete in similar direction); QRS complex occurs when the impulse spreads to ventricles and triggers ventricular contraction; ST segment is the period when the ventricles are depolarised and ventricular repolarisation (relaxation) begins and T wave represents the return to resting state by the ventricles [2].

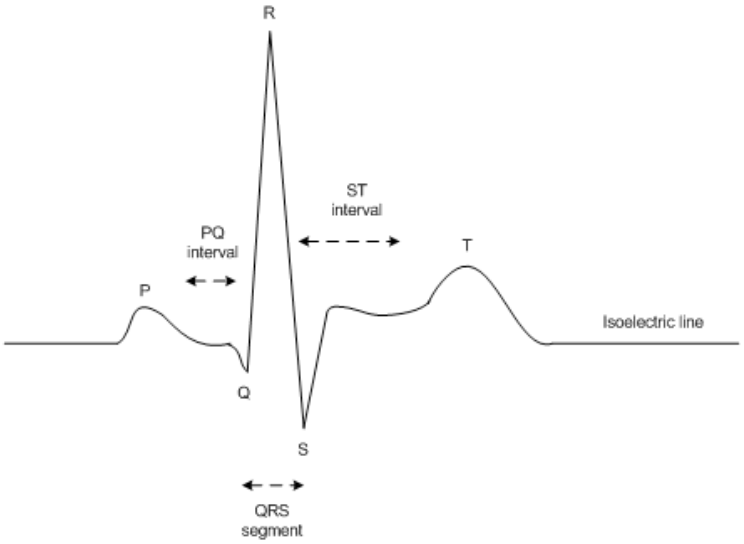


Figure 1.5: ECG waveform.

These electrical impulses are normally recorded through electrodes placed on particular areas of the body either using 12-channel ECG in a hospital or 3-channel ECG in the field. Figure 1.6 shows the hypothetical Einthoven's triangle that is commonly used for the electrode setup. The setup requires four limb electrodes placed on the right and left arms and legs; the fourth limb electrode that is not shown in the diagram is placed on the right leg and serves as reference channel. The top of the Einthoven's triangle forms Lead I, while the left forms Lead II and right forms Lead III. Each lead represents a different aspect of the heart's electrical system. The voltage differences between the limb electrodes: left arm (LA), right arm (RA), and left leg (LL) are used to obtain Leads I, II and III [3, 4]:

$$I = V_{LA} - V_{RA}, \tag{1.1}$$

$$II = V_{LL} - V_{RA}, \tag{1.2}$$

$$III = V_{LL} - V_{LA}. \tag{1.3}$$

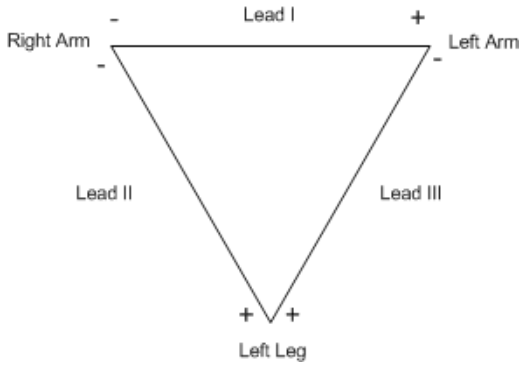


Figure 1.6: Einthoven's triangle.

It should be obvious that the three leads have the relationship:

$$II = III + I. \tag{1.4}$$

These three leads are normally sufficient to detect life threatening abnormal ECG rhythms (arrhythmias) and will often be used in this book. Einthovan's triangle electrode setup does also give an additional three leads and these augmented limb leads (aVF, aVL, and aVR) can be computed by [3, 4]:

$$aVR = V_{RA} - \frac{V_{LA} + V_{LL}}{2}, \tag{1.5}$$

$$aVL = V_{LA} - \frac{V_{RA} + V_{LL}}{2}, \tag{1.6}$$

$$aVF = V_{LL} - \frac{V_{LA} + V_{RA}}{2}. \tag{1.7}$$

The six precordial leads provide a more detailed view of the heart's electrical activity compared to the six limb leads. These six precordial leads can be obtained by placing electrodes on the chest as shown in Figure 1.7.

The obtained voltages are referenced to hypothetical Wilson's Central Terminal, which is the average of the voltages V_{LA}, V_{RA} and V_{LL} [3]:

$$V_{WCT} = \frac{V_{LA} + V_{RA} + V_{LL}}{3}. \tag{1.8}$$

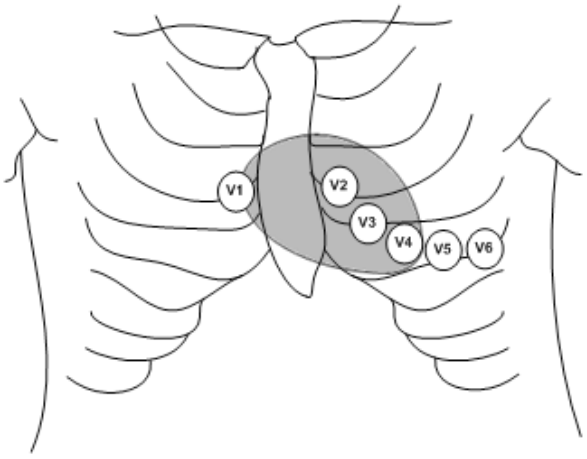


Figure 1.7: Precordial (chest) electrode positions [3]. Figure used with permission from Elsevier.

1.2.2 Electroencephalogram

EEG which represents the electrical activity of the brain has become very useful in the clinical diagnosis and electrophysiological analysis related to functions of the brain. The basic functional unit in the brain is the neuron, which is found in the cerebral cortex. Four different areas of the cortex (frontal, parietal, temporal and occipital) are responsible for varying functions, for example the occipital lobe processes visual information and auditory perception is processed in temporal lobe.

Figure 1.8 shows the neuron and its interconnections. The brain is made of billions of such connections. The cell body (soma) of a neuron receives neural activity inputs though dendrites and outputs its neural activity through an axon. The axon is covered with a myelin sheath that acts as an insulator (just like rubber covering of copper electrical wires). The axon also contains Ranvier nodes at intervals that act to amplify the signals [5].

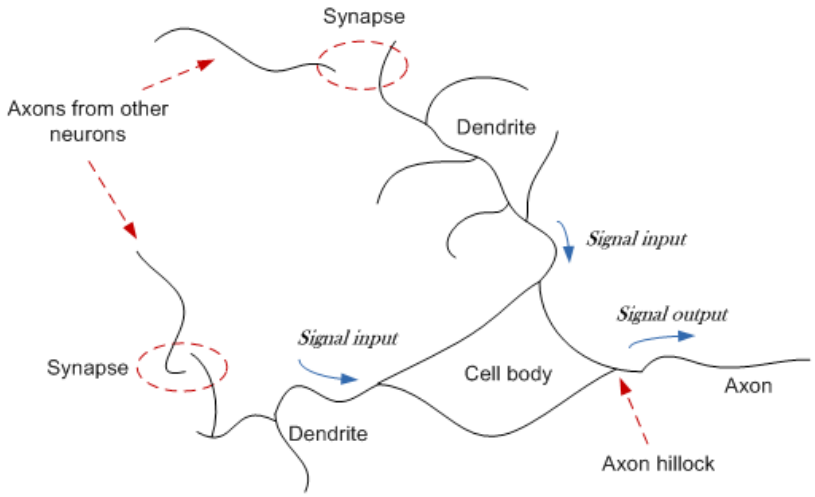


Figure 1.8: Neuron and its interconnections.

The synapse (shown in Figure 1.9, junction of an axon terminal with another neuron's dendrite) is where the actual transmission of information from one neuron to another takes place. The summation of the received signals in soma travels to the axon if it exceeds a certain threshold at axon hillock. At the end of axon (presynaptic membrane), the electrical potential causes chemicals known as neurotransmitters (released by synaptic vesicles) to diffuse across the synaptic cleft and the amount of neurotransmitters that are diffused at the postsynaptic membrane cause proportional electrical potential at the dendrite of the connecting neuron causing an input signal to the connecting neuron's soma[3] [5, 6].

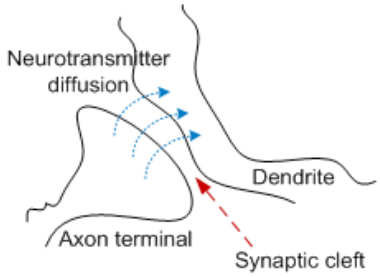


Figure 1.9: Synapse.

This EEG activity is recorded through electrodes placed on the scalp. The cumulative electrical activity of thousands of neurons (sometimes know as rhythms) as they propagate through the skull and scalp are significantly attenuated but nevertheless can still be captured and amplified to a level that is sufficient for analysis. EEG rhythms are conventionally categorised into five different rhythms based on their frequency ranges (EEG components with frequency less than 0.5 Hz are normally considered to be baseline wander noise): Delta (0.5–4 Hz) rhythm, which only appears during deep sleep stages and in infants as irregular activity; Theta (4–7 Hz) rhythm which is encountered in early sleep stages and drowsiness; Alpha (8–12 Hz) rhythm which is the typical rhythm during relaxed state with eyes closed (it is suppressed with eye opening); Beta (13–30 Hz) rhythm which is prominent during stressful situations and Gamma (> 30 Hz) rhythms, which are believed to be involved in higher order functions of the brain.

There are also other transient components in EEG that are encountered during sleep (like K complexes), movements (mu rhythm) and epilepsy (ictal component). Specific components that are in response to external stimuli (i.e. evoked potentials (EP)) are yet another type of EEG and will be discussed in the next section.

In usual practice, EEG signals are recorded on multiple locations on the scalp using electrodes placed at specific points. The International Federation of Societies for Electroencephalography and Clinical Neurophysiology has recommended the 10–20 system of electrode placement [7], which consists of 19 actives electrodes plus two reference (linked to earlobes or mastoids). The distance between each electrode is either 10% or 20% of the total edge distances (e.g. nasion-inion), hence the name 10–20.

Figure 1.10 shows the 10–20 system of electrode placement, where the letters A, C, F, O, P, and T denote auricle, central, frontal, occipital, parietal, and temporal, respectively[4]. Odd and even numbering are used for the left and right sides, respectively. Midline electrodes are numbered with Z, representing zero. Nowadays, it is common to extend this 10-20 system by placing electrodes in between thus arriving at 32, 64, 128 and even 256 channels!

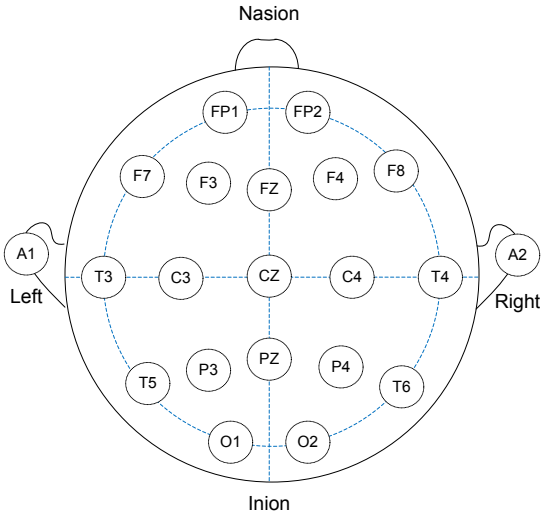


Figure 1.10: The 10–20 system of electrode placement for EEG recording.

Figure 1.11 shows examples of EEG signals extracted from a subject while resting and performing an active mental task (non-trivial computing task). Visually, both these signals appear to be just noise but in later chapters, it will be shown that both these signals can be discriminated effectively for use in a brain-computer interface application.

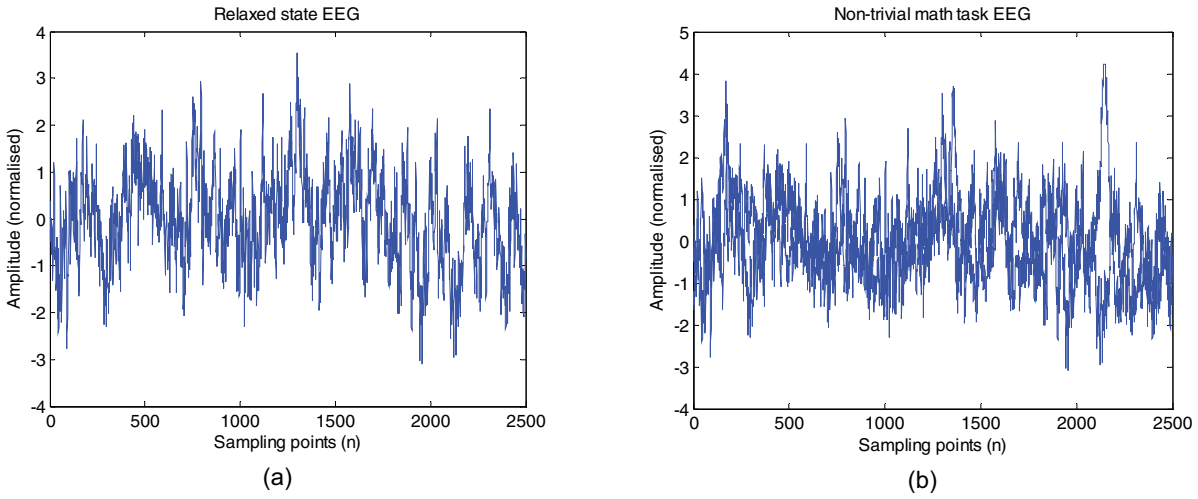


Figure 1.11: EEG signals during: (a) relaxed state (b) non-trivial math task. EEG data used here was obtained from [8].

1.2.3 Evoked Potential

Evoked potential or sometimes known as event related potential includes EEG that occurs in response to external stimulation such as visual, auditory, or somatosensory. The responses to these stimuli are known specifically as visual evoked potential (VEP), auditory evoked potential (AEP) and somatosensory evoked potential (SEP), respectively. The early responses are normally exogenous (i.e. a spontaneous response to external stimulus and is dependent on brightness, volume etc of the stimulus) while the late occurring endogenous components are due to psychological processing of the stimulus. The responses can be transient, which is the response of single stimulus or steady state, which is the response of repetitive stimuli. Figure 1.12 shows a typical morphology of a brainstem AEP signal in response to a short duration click stimulus that would be obtained after averaging the EEG data from many trials [6]. Most EP signals are very low in amplitude compared to the ongoing, background EEG and hence techniques such as time locked ensemble averaging is required to extract the EP component that is related to the external stimulus (this will be dealt in detail in later chapters).

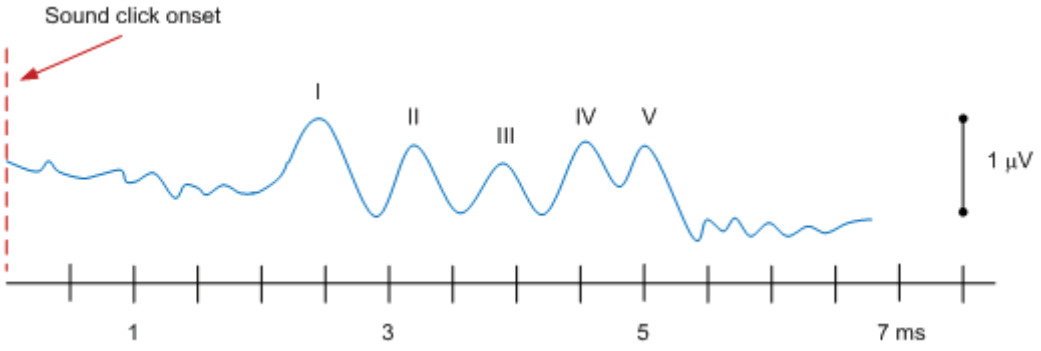


Figure 1.12: Brainstem AEP, consisting mainly of 5 waves in first 10 ms after the sound.

Figure 1.13 shows the important components that are likely to be encountered in a transient EP signal. It should be noted that these components would only be evident after averaging across many trials. N100[5] is the negative peak that is easily evoked by auditory, visual and tactile stimuli (for example, clapping a hand or showing a flash of light). It is believed to be the response to stimulus perception, i.e. when the brain recognises that a stimulus has been presented. Hence, it could be used to diagnose damages to neural pathways in vision or auditory. P300 is the third positive peak that is evoked around 300 ms after a target stimulus appears in random presentation of targets and non-targets, with the frequency of target stimuli smaller than non-targets (this is known as oddball paradigm). P300 component has found many uses, the recent being in the design of brain-computer interface technologies. N400 is the negative component evoked with latency about 400 ms. It relates the subjects ability in processing semantic information. For example, a word in a sentence that is an outlier will evoke N400: “It was nice seeing you eating a cake” may not evoke N400, but “It was nice seeing you eating a cage” will evoke N400. So, N400 has been used in assessing language capabilities.

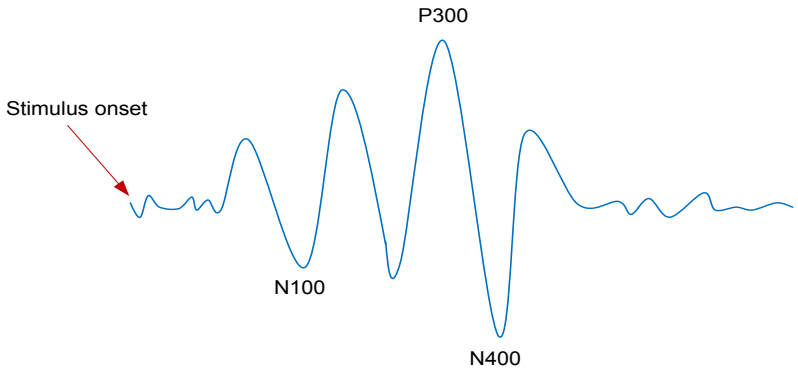


Figure 1.13: Typical components in a transient EP signal.

1.2.4 Electromyogram

Electromyogram (EMG) is recorded by an electromyography device, which measures the muscle's electrical potential. The central nervous system consisting of the brain, spinal cord and peripheral nerves controls the action of the muscle fibres that typically results in movements. Muscle is composed of specialised cells that are capable of contraction and relaxation and is controlled by simulations from innervated motor units (neurons).

EMG can be recorded by two methods: surface EMG (SEMG, which records EMG using electrodes placed on the skin) which is more popular than intramuscular EMG (a needle electrode is inserted in the muscle) as it is non-invasive. SEMG measures the muscle fibre action potentials of a single (or more) motor unit, which are known as the motor unit action potentials (MUAPs). The actual potential is about 100 mV but due to the layers of connective tissue and skin, the SEMG is a complex signal with much less amplitude (typically about 5 mV). Figure 1.14 shows an example of a SEMG signal.

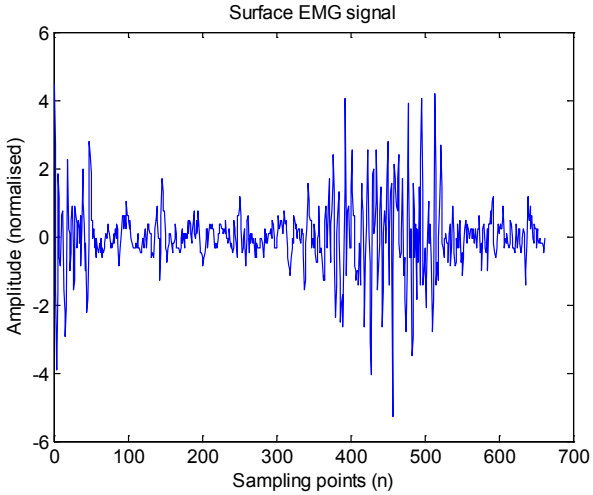
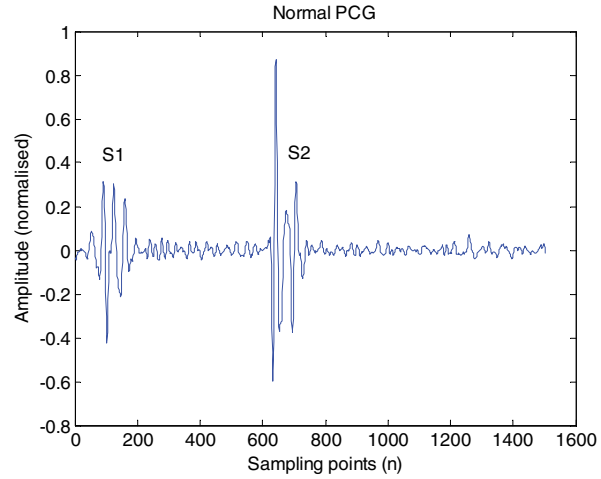


Figure 1.14: Typical surface EMG signal.

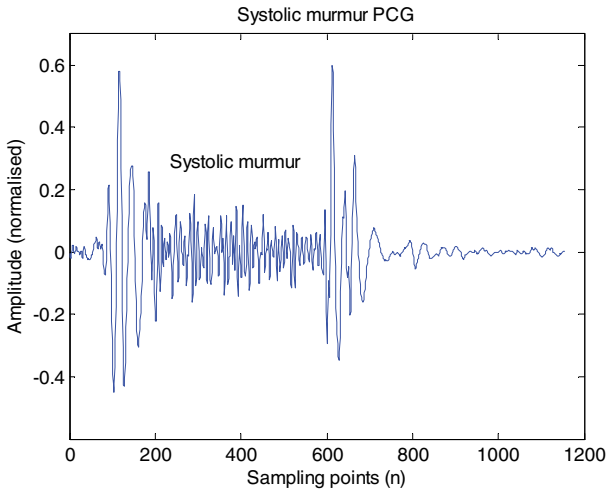
1.2.5 Phonocardiogram

Phonocardiography is the vibration or sound of the heart when it pumps blood [4]. Though the common cardiac analysis centres on ECG, phonocardiography can provide complementary valuable information concerning the function of heart valves and the hemodynamics of the heart as ECG can only detect faults in heart's electrical system.

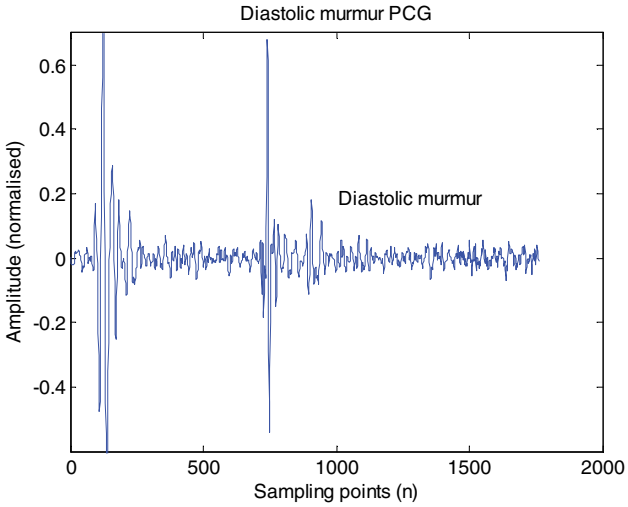
The phonocardiography of a normal heart comprises of two distinct activities namely the first heart sound, S_1 and the second heart sound, S_2, which correspond to the 'lup' and 'dup' sounds, respectively. An abnormal heart, on the other hand, includes several other activities between S_1 and S_2 sounds. These abnormal sound activities (like S_3, S_4, murmurs, clicks and snaps) are useful in diagnosing heart diseases. Nowadays, the sound waves produced by the heart are not only heard using a stethoscope but also observed as phonocardiogram (PCG) signals on a monitor screen. Figure 1.15 shows PCG signals for a normal heart, systolic murmur (SM) and diastolic murmur (DM).



(a)



(b)



(c)

Figure 1.15: PCG signals (a) Normal heart (b) Systolic murmur (c) Diastolic murmur.

1.2.6 Other Biological Signals

There are many other biological signals such as arterial blood pressure signals (ABP, shown in Figure 1.16) generated by changes in blood pressure which are recorded on the upper arm (units-mmHg); electroculogram (EOG) signals, which measure the eye movements and oxygen saturation (SpO_2) signals which measures the level of oxygen in blood. It is common to perform a multimodal signal analysis, where more than one signal modality is recorded. This is useful to perform a more thorough diagnosis. Figure 1.17 shows a typical example where three ECG leads, arterial pressure, pulmonary arterial pressure, central venous pressure, respiratory impedance, and airway CO2 signals are recorded from a subject.

1.3 Contents of this book

This book is concerned with the aim of analysing biological signals to extract useful information. This chapter has introduced some of these biological signals and discussed their origins. The methods for signal analysis will depend on the type of the signal and nature of the information being carried by the signal – there are some general methodologies and some specific ones for particular signals. The following five chapters will deal with some of the popular methodologies while the final chapter is dedicated for discussion of several real world applications that use the methodologies described in the previous chapters. MATLAB[6] software is used throughout the text here. While some would argue that other specific software packages could have been used, the decision to use MATLAB was taken as it is commonly used in signal analysis and requires much smaller effort in learning to use it.

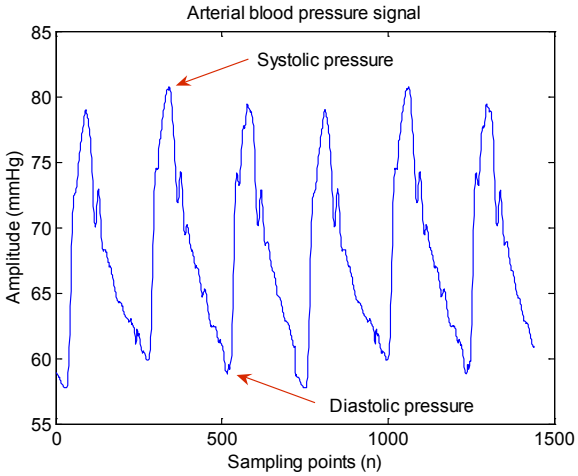


Figure 1.16: Arterial blood pressure signal. ABP data shown here was obtained from [1].

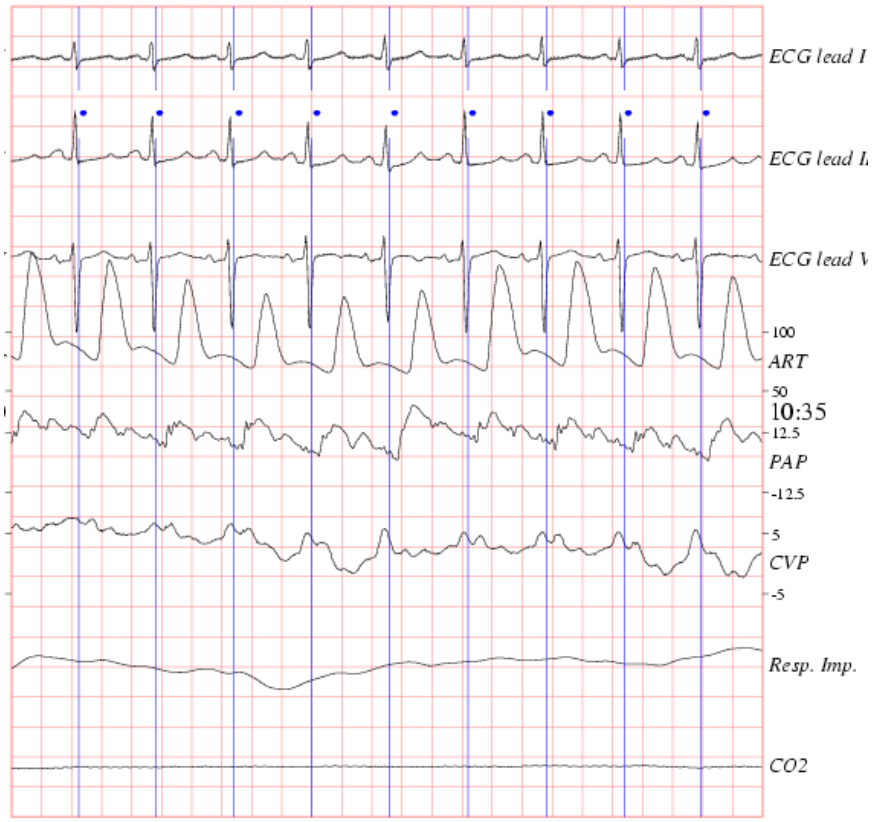


Figure 1.17: Multimodal data from MGH/MF database. Figure from [1].

The next chapter will be on discrete-time signals and systems where the basic principles of signals and systems will be discussed. The important process of converting analogue to discrete-time signals involving sampling will be described in this chapter.

Classical spectral estimation using Fourier analysis will be dealt in chapter 3. Basic properties of discrete Fourier transform are discussed before moving to its utilisation for estimating the spectral components in a signal.

Chapter four starts with an introduction to digital filtering and moves on to filter design, where both finite impulse response and infinite impulse response filters are discussed. Chapter five will focus on feature extraction methods such as power spectral density, asymmetry ratio and autoregressive models.

Chapter six will look at decision making models where both a simple classifier (k-nearest neighbour) and a more advanced one (artificial neural network) are discussed. Cross validation measures to improve the reliability of the classification results and statistical validation (such as t-test) will also be described in this chapter. As mentioned earlier, the final chapter will discuss a few recently studied real world applications using common biological signals.

1.4 References

[1] The Massachusetts General Hospital/Marquette Foundation (MGH/MF) Waveform Database: Available: http://www.physionet.org/pn3/mghdb/.

[2] B.M. Beasley, *Understanding EKGS: A Practical Approach, 2nd ed.*, Pearson Education, 2003.

[3] L. Sornmo and P. Laguna, *Bioelectrical Signal Processing in Cardiac and Neurological Applications,* Elsevier, 2005.

[4] R.M. Rangayyyan, *Biomedical Signal Analysis: A Case-Study Approach*, IEEE Press, 2002.

[5] K. Gurney, *Introduction to Neural Networks*, Taylor and Francis, 1997.

[6] R. Cooper, J.W. Osselton, and J.C. Shaw, *EEG Technology*, Butterworths & Co, 1980.

[7] H. Jasper, "The ten twenty electrode system of the international federation," *Electroencephalographic and Clinical Neurophysiology,* 10:371-375, 1958.

[8] Mental Task EEG data: Available: http://www.cs.colostate.edu/eeg/eegsoftware.html#keirndata.

2 Discrete-time signals and systems

In this chapter, we'll look at discrete-time signals and systems. As described in the first chapter, a signal is a function of independent variables such as time, distance, position, temperature, pressure, etc. Most signals are generated naturally but a signal can also be generated artificially using a computer and signals can be in any number of dimensions (1D, 2D, 3D, etc). We'll be mainly studying time series signals, which are 1D signals with amplitude, pressure, intensity, etc as a function of time. Now, what about discrete-time systems?

Discrete-time systems operate on an input signal, according to some prescribed function[7] and produce another output signal. For example, the input sequence could be a noisy electrocardiogram (ECG) signal and the system outputs a noise reduced ECG signal. In this example (see Figure 2.1), the discrete-time system is a band-pass filter that removes low frequency baseline noise and high frequency powerline interference[8].

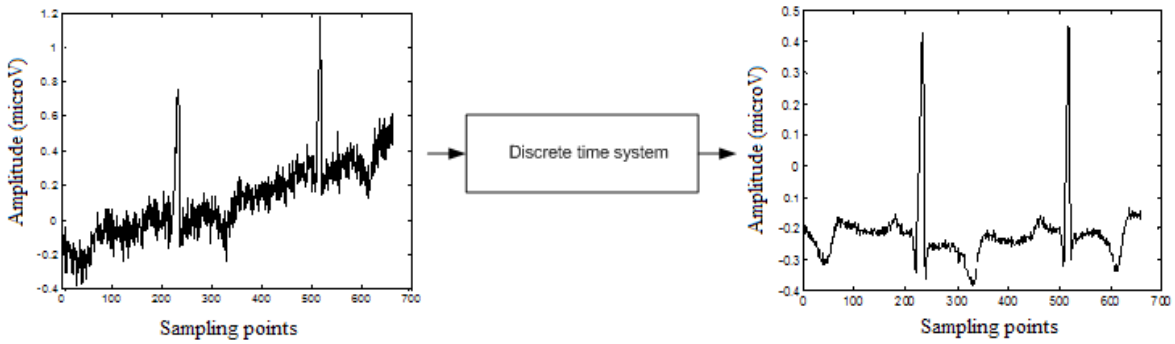


Figure 2.1: Band-pass filter as an example of a discrete time system.

There are some basic operations for discrete-time systems, which we will study in this chapter but before we do that, we need to understand that discrete-time systems operate on discrete-time signals and we need to obtain the latter from analogue signals.

2.1 Discrete-time signal

In general, biological signals that are recorded are analogue and to process analogue signals by digital means (like using a computer), we need to convert them to discrete-time form, i.e. to convert them to a sequence of numbers defined at specific uniform intervals. This process is known as sampling[9].

2.1.1 Sampling

Since most biological signals are 1D time series signals[10], we'll focus our attention to such signals where the independent variable is time. A discrete-time signal, $x[n]$ is developed by uniformly sampling an analogue signal $x(t)$ as indicated in Figure 2.2 below. It is done by taking samples at specific uniform intervals of time (every value of $x[n]$ is called a sample) and the sampling interval is the time of one of these uniform intervals while the sampling frequency is the number of uniform time intervals in one second normally specified in Hz. Then, discrete variable n can be normalised to assume integer values as a representation of t.

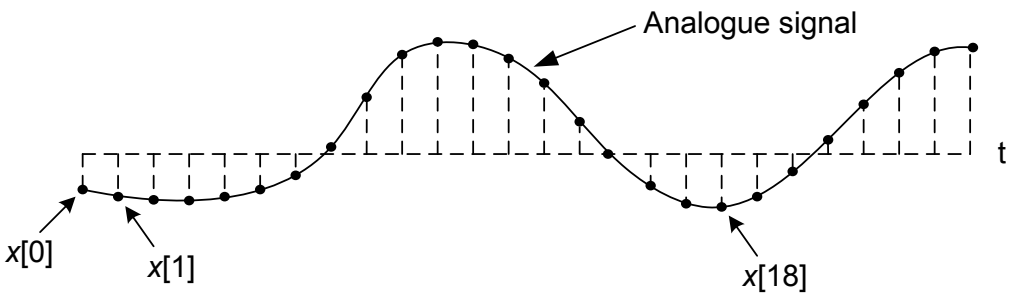


Figure 2:2: Obtaining a discrete-time signal from analogue signal through sampling.

2.1.2 Aliasing

Aliasing is a problem in the sampling process as it causes ambiguities in reconstruction, i.e. distorts the sampled signal unrepresentative of the original signal. To avoid aliasing and be able to reconstruct the original signal without errors, the sampling frequency has to be more than twice of the highest frequency contained in $x(t)$. This is known as Nyquist theorem and the minimum frequency known as Nyquist frequency (rate).

Consider the following sampling example of an analogue signal with sufficient sampling frequency.

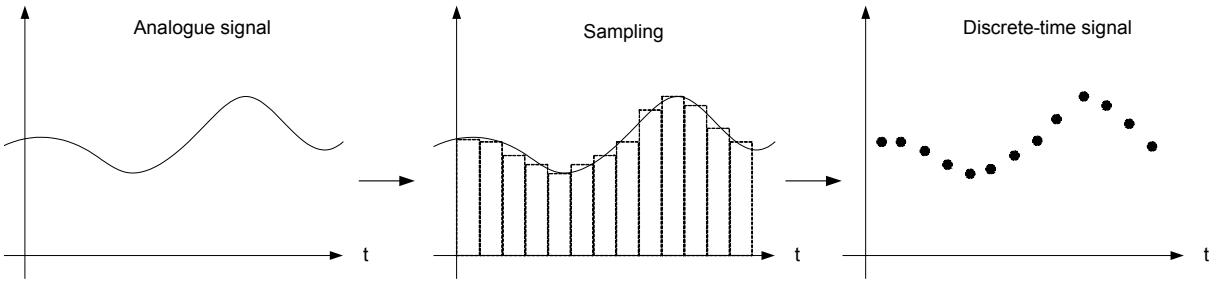


Figure 2.3: Sampling with high enough frequency – the signal is *correctly* represented.

Figure 2.4 shows the problem of aliasing with insufficient sampling frequency. It can be seen that the analogue signal in Figure 2.4 is not represented correctly by the discrete-time version.

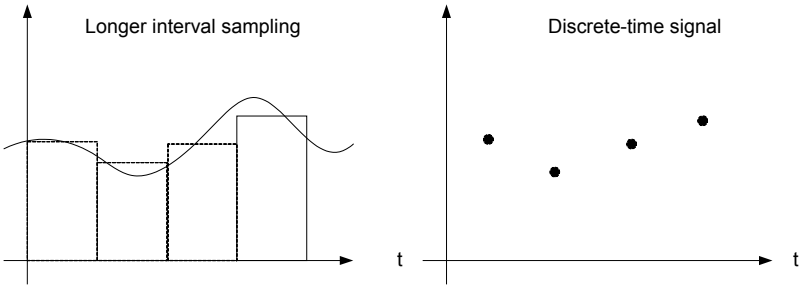


Figure 2.4: Aliasing problem – effects of sampling frequency below Nyquist frequency.

Example: Consider the analogue signal $x(t) = 3\cos 50\pi t + 10\sin 300\pi t - \cos 100\pi t$. What is the Nyquist frequency for this signal?

Answer: Use the generic term, $A\cos 2\pi t$ or $A\sin 2\pi t$ to compute the frequencies present in the signal, which are 25 Hz, 150 Hz and 50 Hz. So, Nyquist frequency is 2* highest frequency= 2*150 Hz = 300 Hz. In practise, we normally sample at a much higher rate than Nyquist frequency.

2.2 Sequences

Sometimes, a discrete-time signal is known as a sequence and vice versa. A discrete-time signal may be a finite length or an infinite-length sequence, e.g. $x[n]=1-2n^4$, $-5\le n \le 2$ is a finite length sequence with length $2-(-5)+1=8$ but $x[n]=\sin(0.1n)$ is an infinite-length sequence.

An important and perfectly valid operation with sequences is zero padding. An N length sequence can be increased by padding with zeros in the beginning or in the end. For example, a sequence with length 3 can be changed to length 7 by padding with 4 zeros:

$$x[n] = n^4, \quad 1 \le n < 4;$$

$$x_{pad}[n] = \begin{cases} n^4, & 1 \le n < 4; \\ 0, & 4 \le n \le 7. \end{cases} \tag{2.1}$$

2.3 Basic Discrete-time System Operations

There are several common operations that we'll frequently encounter for discrete-time systems and we'll look at a few here.

2.3.1 Product (modulation)

Product operation is given as:

$$y[n] = w[n].x[n]\,. \tag{2.2}$$

It is frequently used in windowing, where a discrete-time finite signal $y[n]$ is obtained from a discrete-time infinite signal $x[n]$ using a window, $w[n]$.

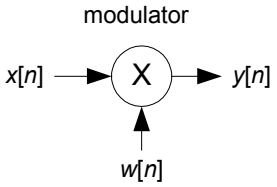


Figure 2.5: Product operation.

2.3.2 Addittion

Addition operation can be performed as given below to add two sequences,

$$y[n] = x[n] + w[n] \tag{2.3}$$

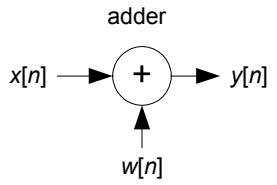


Figure 2.6: Addition operation.

2.3.3 Multiplication

Multiplication operation is used to amplify or to attenuate a signal:

$$y[n] = A.x[n] \qquad or \qquad y[n] = Ax[n]\,. \tag{2.4}$$

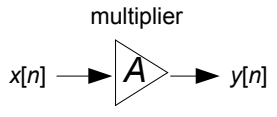


Figure 2.7: Multiplication operation.

2.3.4 Time reversal (folding)

Folding operation is important for filtering and is given by:

$$y[n] = x[-n]. \tag{2.5}$$

The input sequence is flipped at n=0 to produce the output sequence.

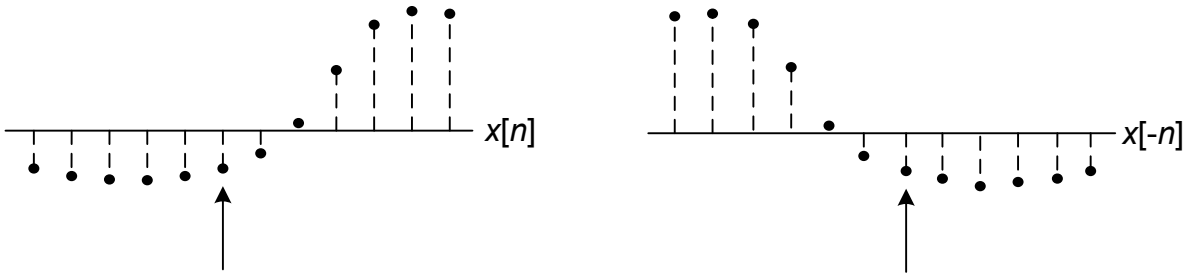


Figure 2.8: Folding operation (arrow points to n=0).

2.3.5 Branching

Branching is used to provide multiple copies of the input sequence:

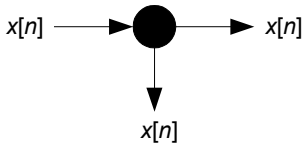


Figure 2.9: Branching operation.

2.3.6 Time shifting

Time shifting denotes delaying or advancing the input by N samples:

$$\begin{aligned} y[n] &= x[n-N] \quad \text{(delay)} \\ y[n] &= x[n+N] \quad \text{(advance)} \end{aligned} \tag{2.6}$$

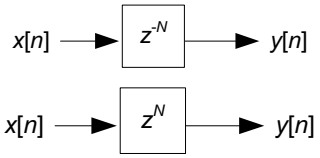


Figure 2.10: Block diagram for time shifting operation.

2.3.7 Time scaling

Time scaling can be categorised into down sampling or up sampling. In down sampling, every Mth sample of the input sequence is kept and M-1 in-between samples are removed. Hence, the number of samples is reduced or in other words, the sampling frequency is reduced. Down sampling can be denoted as

$$y[n] = x[Mn];$$
$$x[n] \rightarrow\downarrow M \rightarrow y[n]. \tag{2.7}$$

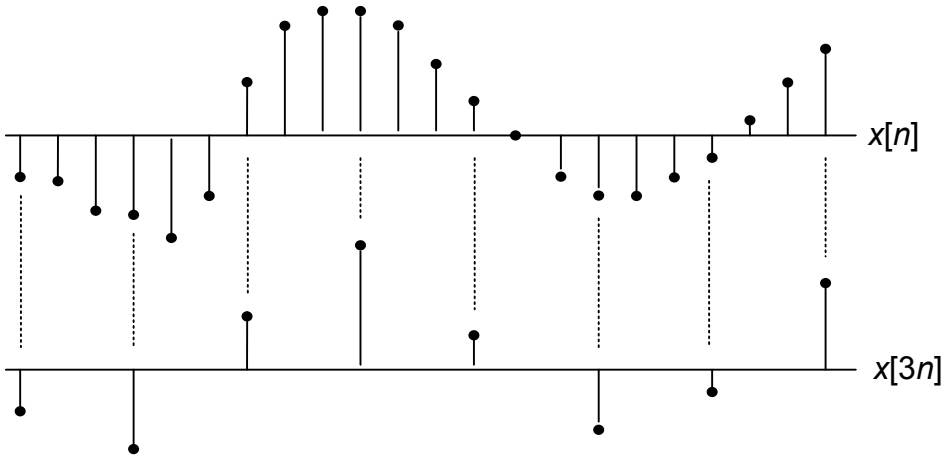


Figure 2.11: Example showing the down sampling process for M=3.

Up sampling is the opposite process of down sampling. In up sampling, N-1 equidistant zero-valued samples are inserted by the up sampler device between each consecutive samples of the input sequence. As a result, the number of samples is increased which effectively increases the sampling frequency.

$$\begin{aligned} &y[n] = x[n/N]; \\ &x[n] \rightarrow \uparrow N \rightarrow y[n]. \end{aligned} \tag{2.8}$$

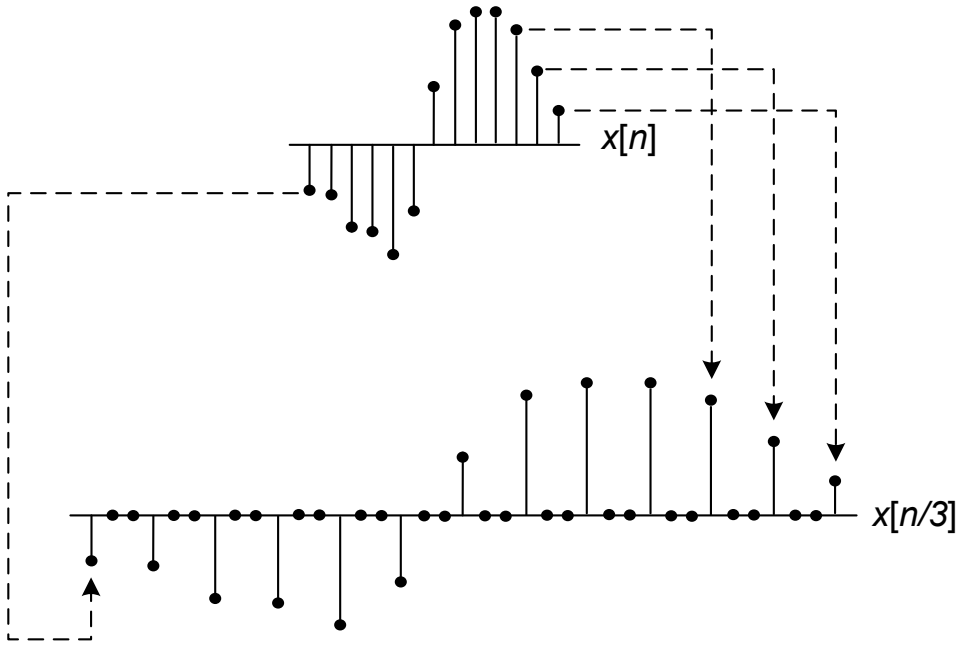


Figure 2.12: Example showing the up sampling process for N=3.

2.3.8 Combination of operations

Often, a system includes a combination of these operations. For example, Figure 2.13 shows a discrete-time system block diagram that includes 3 multipliers, 3 single sample delays and 1 adder operations for $y[n]$:

$$y[n] = \alpha x[n] + \beta x[n-1] + \lambda x[n-3] \tag{2.9}$$

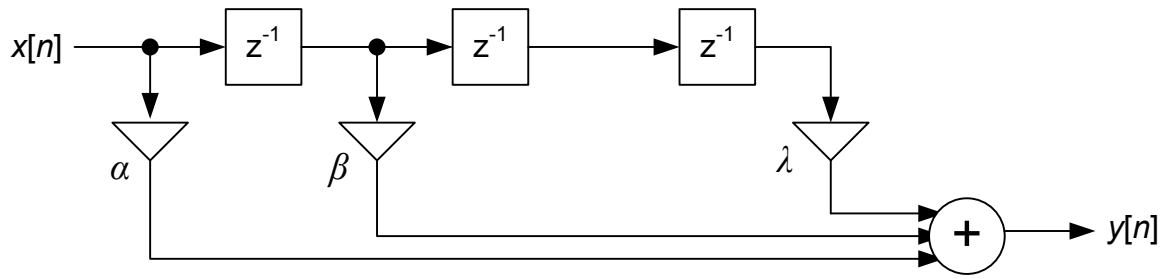


Figure 2.13: Block diagram of a discrete-time system $y[n]$.

2.4 Examples on sequence operations

Several examples are given here to illustrate the concept of combining different sequence operations.

Example 1
For the following sequences, defined for $1 \le n \le 5$ (i.e. length=5),

- $a[n]=\{1\ 2\ 4\ -9\ 1\}$;
- $b[n]=\{2\ -1\ 3\ 3\ 0\}$.

obtain the new sequences

- $c[n]=\{2.a[n].b[n]\}$;
- $d[n]=\{a[n]+b[n]\}$;
- $e[n]=0.5\{a[n]\}$.

Answer

- $c[n]=\{4\ -4\ 24\ -54\ 0\}$;
- $d[n]=\{3\ 1\ 7\ -6\ 1\}$;
- $e[n]=\{0.5\ 1\ 2\ -4.5\ 0.5\}$.

These are easy as both $a[n]$ and $b[n]$ have same length. What if their lengths differ? If the lengths of sequences differ, then pad with zeros (in front or end) to obtain same length sequences and same defined ranges before applying the operations.

Example 2
For the following sequence $f[n]=\{-5\ 2\ -3\}$ defined for $1 \le n \le 3$, what would be $g[n]=a[n]+f[n]$?

Answer
As $f[n]$ has length 3 and $a[n]$ has length 5, pad $f[n]$ with 2 zeros.
$a[n]=\{1\ 2\ 4\ -9\ 1\}$ defined for $1 \le n \le 5$;
$f_{pad}[n]=\{-5\ 2\ -3\ 0\ 0\}$ defined for $1 \le n \le 5$.

$f[n]$ is padded with 2 zeros at the end as to make the defined ranges of $a[n]$ and $f_{pad}[n]$ equal. So, $g[n]=\{-4\ 4\ 1\ -9\ 1\}$.

Example 3
Consider the following sequences:

- $a[n]=\{-3\ 4\ 2\ -6\ -9\}$, $-3 \le n \le 1$;
- $b[n]=\{-3\ -1\ 0\ 8\ 7\ -2\}$, $-2 \le n \le 3$;
- $c[n]=\{1\ 8\ -3\ 2\ -6\}$, $3 \le n \le 7$.

The sample values of each of the above sequences outside ranges specified are all zeros. Generate the following sequences (put an arrow at n=0):

- $d[n]=a[-n+3]$;
- $e[n]=b[-n]$;
- $f[n]=a[n]+b[-n+2]+c[-n]$.

Answer

Making a table will make it easier to obtain the answer.

n	-3	-2	-1	0	1	2	3	4	5	6	7	8
$a[n]$	-3	4	2	-6	-9							
$b[n]$		-3	-1	0	8	7	-2					
$c[n]$							1	8	-3	2	-6	

For $d[n]=a[-n+3]$, the folding operation is performed first and then the shift to obtain

$$d[n] = \{0 \quad 0 \quad -9 \quad -6 \quad 2 \quad 4 \quad 3\}$$
$$\uparrow$$

n	-3	-2	-1	0	1	2	3	4	5	6
a[*n*]	-3	4	2	-6	-9					
a[-*n*]			-9	-6	2	4	-3			
a[-*n*+3]				**0**	**0**	-9	-6	2	4	-3

Similarly, for $e[n]=b[-n]$, we have

$$e[n] = \{-2 \quad 7 \quad 8 \quad \underset{\uparrow}{0} \quad -1 \quad 3\}.$$

n	-3	-2	-1	0	1	2	3	4	5	6	7	8
b[*n*]		-3	-1	0	8	7	-2					
b[-*n*]	-2	7	8	0	-1	-3						

For $f[n]=a[n]+b[-n-2]+c[-n]$, we have

$$f[n] = \{-6 \quad 2 \quad -5 \quad 15 \quad 6 \quad 4 \quad 1 \quad \underset{\uparrow}{-9} \quad 9\}.$$

n	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8
a[*n*]					-3	4	2	-6	-9							
b[*n*]						-3	-1	0	8	7	-2					
b[-*n*]					-2	7	8	0	-1	-3						
b[-*n*-2]			-2	7	8	0	-1	-3								
c[*n*]								**0**	**0**	**0**	1	8	-3	2	-6	
c[-*n*]	-6	2	-3	8	1	0	0	0								
f[*n*]	-6	2	-5	15	6	4	1	-9	-9							

Often, combination of folding and shifting operations causes confusion on which direction to shift the sequence. The following hints will make it easier to remember the direction to make the shift operation:

- $x(n+k)$, then the signal moves $k \leftarrow$
- $x(n-k)$ signal moves $k \rightarrow$
- $x(-n+k)$ signal moves $k \rightarrow$
- $x(-n-k)$ signal moves $k \leftarrow$

From the examples above, we can verify that $a(-3-n)= a(-n-3)$ but it is always easier to if we rewrite the expression with the folding operation first.

2.5 Bibliography

[1] S.K. Mitra, *Digital Signal Processing: A Practical Approach, 3rd edition*, McGraw-Hill, 2006.

[2] J.G. Proakis and D.K. Manolakis, *Digital Signal Processing (4th Edition)*, Prentice Hall, 2006.

3 Fourier transform

Knowledge of cyclic or oscillating activity in signals is very important. Fourier transform is the oldest method for analysing the cycles, i.e. obtaining frequency information. For example, analysing the frequency of variable stars is the oldest application of analysing cycles [1]. But then, what is frequency?

Frequency measures the periodicity (i.e. repetitiveness); it measures the number of cycles per second in Hz. Fundamental period (in seconds) is the inverse of the frequency. For example, see the saw-tooth waveform below.

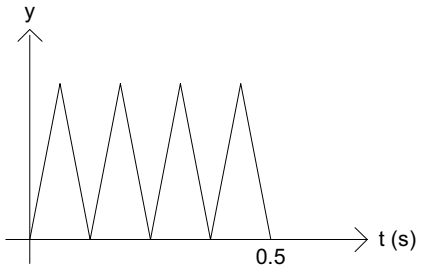


Figure 3.1: Saw-tooth waveform.

In the figure, there are 4 *fundamental* cycles in 0.5 s. Hence, each cycle is completed in 0.125 s and the freq is 1/0.125 = 8 Hz.

Normally, the Fourier method is used to obtain the magnitude of frequency components – from Figure 3.1, we know that the frequency is 8 Hz but how strong is this frequency component? We can use Discrete Fourier Transform for this purpose.

Since we are dealing with discrete-time signals in this book, we will skip discussions on continuous Fourier transform and move into discrete Fourier transform (DFT). But we need to understand the concept of discrete frequency before discussing DFT.

3.1 Discrete frequency

Just like the discrete variable n for time domain, we have discrete frequency number m for frequency domain. Assume N is the number of sampled points, we have the discrete frequency variable (frequency number), m ranging from 0 to N-1. It should be noted that the actual frequency range will be from 0 to one point less than sampling frequency Fs with Fd intervals. Hence, the frequency spacing will be

$$Fd = \frac{1}{NT} \text{ or } Fd = \frac{Fs}{N}. \tag{3.1}$$

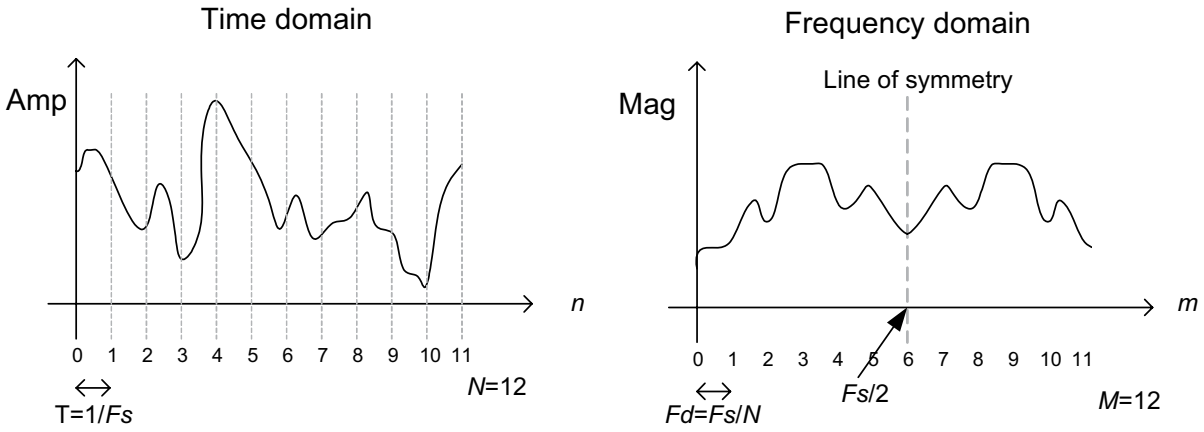


Figure 3.2: Example illustrating the connection between discrete-time and frequency number.

Figure 3.2 shows an example illustrating the discrete-time and frequency number. Here, the discrete-time signal is of length N=12 and as such, when transformed into the frequency domain, the frequency number m will range from 0 to N-1, i.e. 0,1,2,3,…..11. The actual frequency (which is $m^{*}Fd=m^{*}Fs/N$) ranges from 0 to one point less than Fs[11]. It should be obvious from the figure that there is a line of symmetry and due to this symmetry, sometimes we plot spectral magnitude with the values from 0 to (Fs/2) rather than up to Fs.

If Fs = 6 Hz for the discrete-time signal shown in Figure 3.2, the frequency abscissa will be from 0 to 3 Hz. Ignoring symmetrical parts, we will have Figure 3.3 where the frequency number m=0,1…..(N/2) (the rest of N/2 points are not plotted due to symmetry in frequency domain). Thus, m = 0,1,2,3,4,5,6 and actual frequency is ($m^{*}Fs$)/N: 0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0.

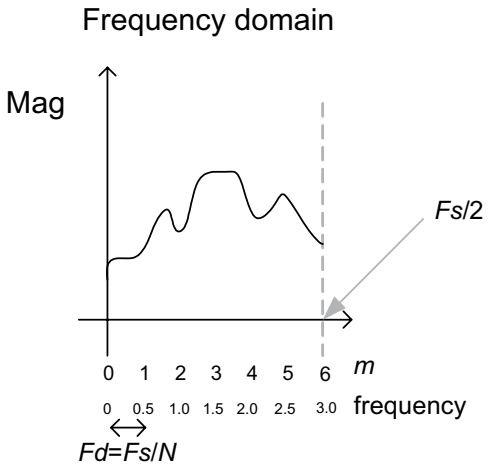


Figure 3.3: Frequency plot (from Figure 3.2) showing range up to Fs/2.

3.2 Discrete Fourier transform

DFT can be used to perform Fourier analysis of discrete-time signals. For discrete-time signal $x[n]$, it is defined as[12]

$$X_{DFT}[m] = \sum_{n=0}^{N-1} x[n] e^{-j\omega n} . \tag{3.2}$$

Since $\omega = 2\pi f / F_s$ and $f = mF_s / N$, we have

$$X[m] = \sum_{n=0}^{N-1} x[n] e^{-\frac{j2\pi mn}{N}} , \tag{3.3}$$

while inverse DFT (IDFT) is defined as

$$x[n] = \frac{1}{N} \sum_{m=0}^{N-1} X[m] e^{\frac{j2\pi mn}{N}} . \tag{3.4}$$

Using Euler's relation,

$$e^{jx} = \cos(x) + j\sin(x), \tag{3.5}$$

DFT can now be expressed as

$$X[m] = \sum_{n=0}^{N-1} x[n]\cos\left(\frac{2\pi mn}{N}\right) - j\sum_{n=0}^{N-1} x[n]\sin\left(\frac{2\pi mn}{N}\right). \tag{3.6}$$

As can be seen, DFT now consist of a real part and an imaginary part for every frequency number *m*. The magnitude and phase spectra can be expressed as

$$\begin{aligned} \left|X[m]\right| &= \sqrt{\mathrm{Re}(X[m])^2 + \mathrm{Im}(X[m])^2}\,, \\ \theta(m) &= \arctan\frac{\mathrm{Im}(X[m])}{\mathrm{Re}(X[m])}, \end{aligned} \tag{3.7}$$

where

$$\begin{aligned} \mathrm{Re}[m] &= \sum_{n=0}^{N-1} x[n]\cos\left(\frac{2\pi mn}{N}\right), \\ \mathrm{Im}[m] &= -\sum_{n=0}^{N-1} x[n]\sin\left(\frac{2\pi mn}{N}\right). \end{aligned} \tag{3.8}$$

The following MATLAB program can be used to compute magnitude and phase spectra using DFT. Alternatively, these can also be computed efficiently using the built-in *fft* function. It should be noted that array indexing for MATLAB starts at 1 and not 0 as in other languages such as C.

```
%Computation of magnitude and phase spectra using DFT
%Input the values for N and x

for m=1:N,
sumn=0;
    for n=1:N,
    sumn=sumn + x(n)*(cos (2*pi*(m-1)*(n-1)/N));
    end
dftreal(m)=sumn;
end

for m=1:N,
sumn=0;
    for n=1:N,
    sumn=sumn + x(n)*(sin (2*pi*(m-1)*(n-1)/N));
    end
dftimag(m)=-sumn;
end

for m=1:N,
dftmag(m)=sqrt(dftreal(m)*dftreal(m)+dftimag(m)*dftimag(m));
dftphase(m)=atan(dftimag(m)/dftreal(m));
end
```

Consider the following example of central processing unit (CPU) usage of a computer in a typical working day. Table 3.1 shows the usage starting from midnight.

Time (hours)	0	2	4	6	8	10	12	14	16	18	20	22
Usage (%)	5	3	4	8	60	80	25	75	60	90	15	20

Table 3.1: CPU usage

Figure 3.4 shows the plot of the data in Table 3.1 obtained using MATLAB codes:

```
time=[0 2 4 6 8 10 12 14 16 18 20 22];
usage=[5 3 4 8 60 80 25 75 60 90 15 20];
plot(time,usage,'+-');
axis([0 24 0 100]);
title('CPU Usage Data');
xlabel('Time (h)');
ylabel('Usage (%)');
```

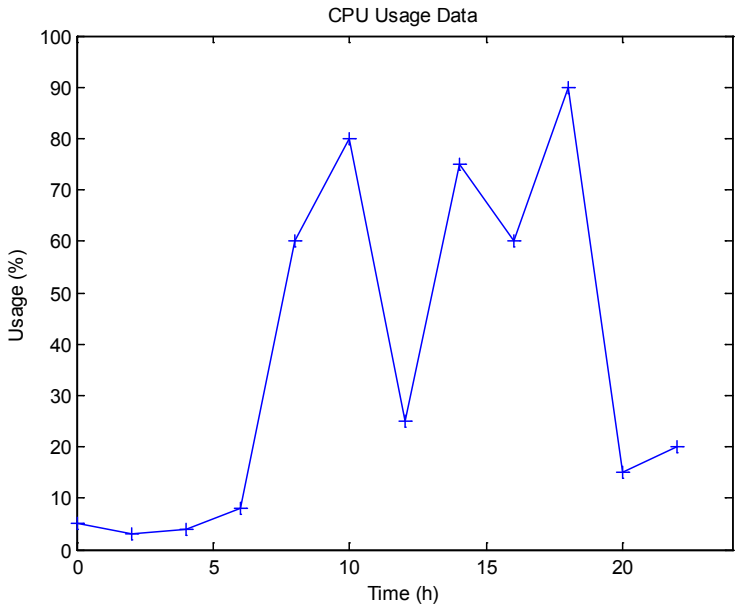


Figure 3.4: CPU usage data.

Next, using the DFT program given earlier, we obtain the real and imaginary parts of DFT for m=0,….11 and plots for these are given using the MATLAB codes below. Assume Fs=12 (since we have 12 readings for a day).

```
plot(0:11,dftreal,'+-');
hold on;
plot(0:11,dftimag,'r*-');
xlabel('Frequency number (m) ');
title('Real and imaginary DFT parts');
```

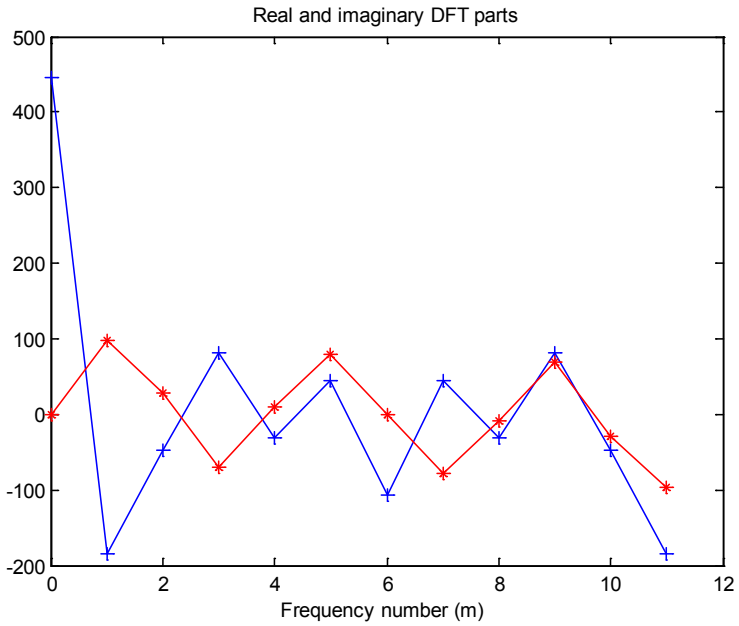


Figure 3.5: Real (in blue) and imaginary (in red) DFT values for CPU usage data.

The point of symmetry is at frequency number 5 for the real DFT values. It should be obvious that DFT imaginary values are conjugate symmetric at this point of symmetry.

Figure 3.6 shows the magnitude and phase spectra obtained using MATLAB codes:

```
plot(0:6,dftmag(1:7),'+-');
xlabel('Scale is in m but note: frequency in cycles/day is also m');
ylabel('Magnitude');
title('CPU usage magnitude spectrum');
figure;
plot(0:6,dftphase(1:7),'r*-');
xlabel('Scale is in m but note: frequency in cycles/day is also m');
ylabel('Phase (rad)');
title('CPU usage phase spectrum');
```

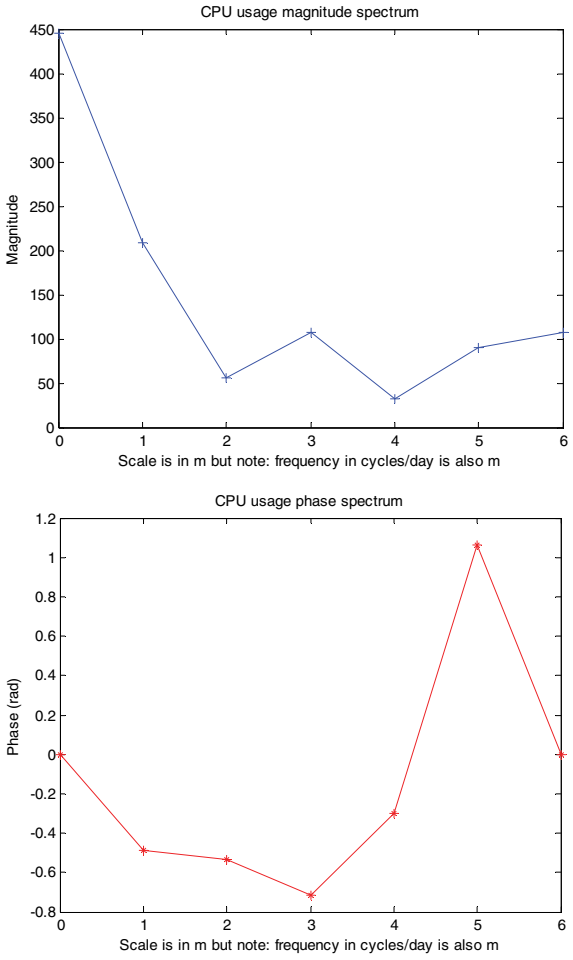


Figure 3.6: CPU usage spectra.

Frequently, the magnitude spectrum is the output that we want from DFT; it measures the *strength* of the signal at the specific frequencies. As explained earlier due to symmetry, magnitude spectrum is normally plotted from f=0 to $Fs/2$ with $N/2+1$ points, i.e. with Fs/N frequency spacing. In the example, Fs=12, N=12, so the actual abscissa (frequency scale) also ranges from 0,1,2,3,4,5,6 cycles/day.

3.3 DFT computation using matrix relation

There is another method to compute DFT using matrix relation [2]:

$$\mathbf{X} = D_N \mathbf{x}, \tag{3.9}$$

where $\mathbf{X}$ is the vector composed of the N DFT samples and D_N is the N by N DFT matrix given by

$$\mathbf{D}_N = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & W_N^1 & W_N^2 & \cdots & W_N^{N-1} \\ 1 & W_N^2 & W_N^4 & \cdots & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & \cdots & W_N^{(N-1)(N-1)} \end{bmatrix}, \tag{3.10}$$

where

$$W_N = e^{-j2\pi/N} \text{ and } X_{DFT}[m] = \sum_{n=0}^{N-1} x[n] W^{mn}{}_N, \quad 0 \le m \le N-1. \tag{3.11}$$

For example, compute the DFT for x=[9 8 3 4] using matrix relation method. Using polar coordinates to get W_N:

$$W_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}, \text{ hence we have } \begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 9 \\ 8 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 24 \\ 6-j4 \\ 0 \\ 6+j4 \end{bmatrix}.$$

3.4 Picket fence effect

One of the problems with DFT is that it provides values of $X[m]$ only at a specific set of frequencies. What if an important value exists at the frequency, $f \neq m$? In our example, we have spectral values for frequencies 0,1,2,3,4,5,6 but what if there was an important cycle at frequency = 1.5? This problem is known as the picket fence effect and is due to discretising the frequency.

The solution is to increase the signal size (length) by zero padding as this will increase N and hence reduce the frequency spacing (i.e. hypothetically increase the resolution of DFT). For example, assume we pad 12 zeros to the CPU usage data and compute the spectra as earlier:

```
usage=[usage, 0 0 0 0 0 0 0 0 0 0 0 0].
```

The results are shown in Figure 3.7.

Now N=24 and frequency spacing =Fs/N=12/24=0.5. From the figure, it can be seen that the spectral values at 0,1,2,3,4,5,6 has not changed but there are values for the in-between frequencies. So, there are frequency values for 0,0.5,1.0,1.5,2.0,……,6.0.

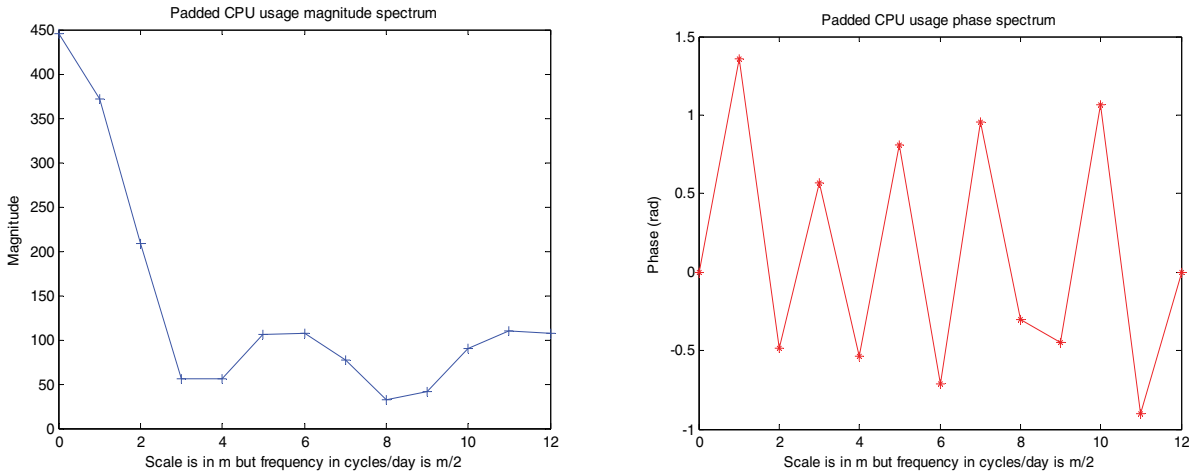


Figure 3.7: Padded CPU usage spectra.

It is important to note that zero padding isn't magic and does not actually improve the resolution of the DFT associated with the original signal, the only way to do that would be to provide a longer signal (i.e. more samples). In this example, as we were interested in an accurate value for frequency = 1.5, we had to double N.

3.5 Effects of truncation

When we compute DFT, we assume that the signal is one complete cycle extracted from an infinite signal that is *periodic* and this is a basic assumption of DFT, i.e. we truncate the signal (though in actual fact, we do not). If the signal was indeed periodic, it would have the same starting and ending points after truncation. But for the CPU usage example, we would have discontinuity, so this creates a lot of noise, which is technically known as spectral leakage and can be solved using *windowing*.

In the CPU data example shown in Figure 3.8, the discontinuity causes spectral leakage and windows such as given in Equations[13] 3.12 to 3.15 [3] can be used to *force* the signal values at the beginning and end to be the same. For most windows, this value is zero though not always.

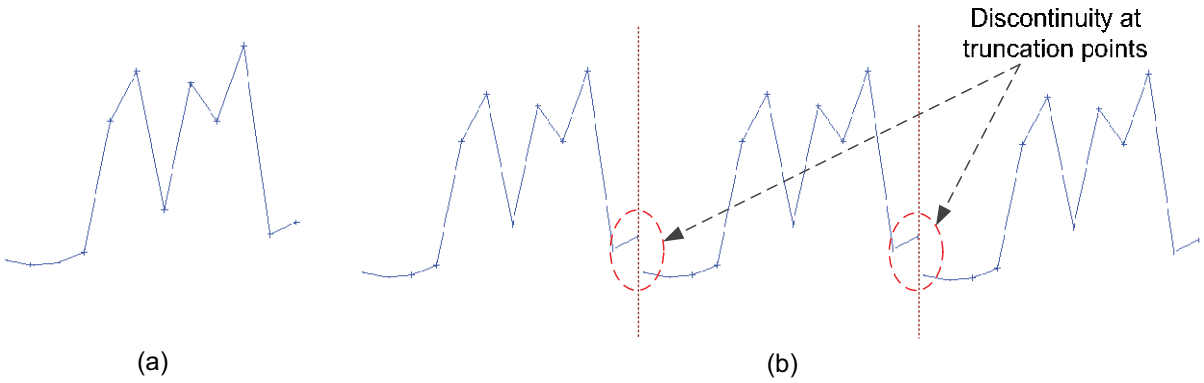


Figure 3.8: (a) Actual CPU usage signal (b) assumption that the signal is obtained from longer periodic signal through truncation.

```
%Blackman spectral window
N=101; %window size/order
plot(blackman(N), 'r-'); xlabel('n'); ylabel('w(n)');
title('Blackman window'); axis([0 105 0 1]);
```

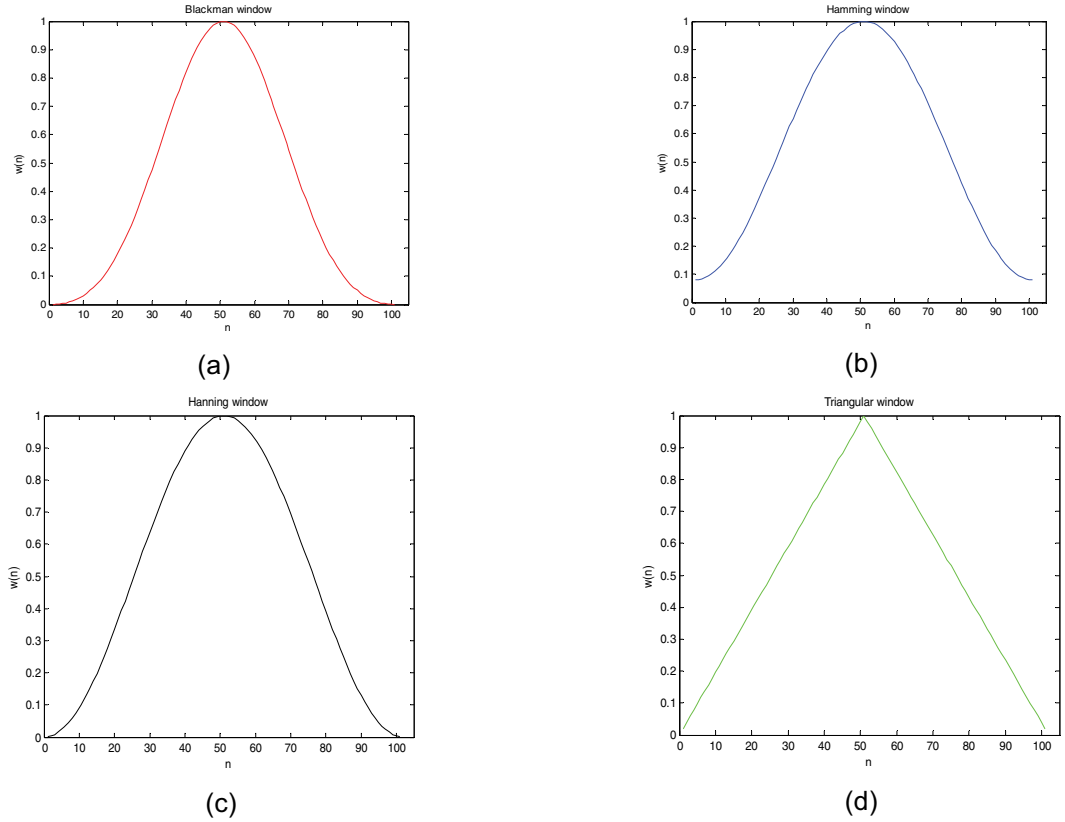


Figure 3.9: Examples of common spectral windows: (a) Blackman (b) Hamming (c) Hanning (d)Triangular.

$$w[n]_{triangular} = 1 - \frac{2\left|n - \frac{N-1}{2}\right|}{N-1}, \qquad 0 \le n \le N-1; \tag{3.12}$$

$$w[n]_{blackman} = 0.42 - 0.5\cos\left(2\pi\frac{n}{N-1}\right) + 0.08\cos\left(4\pi\frac{n}{N-1}\right), \qquad 0 \le n \le N-1; \tag{3.13}$$

$$w[n]_{hanning} = 0.5\left(1 - \cos\left(2\pi\frac{n}{N-1}\right)\right), \qquad 0 \le n \le N-1; \tag{3.14}$$

$$w[n]_{ham\,\min g} = 0.54 - 0.46\left(2\pi\frac{n}{N-1}\right), \qquad 0 \le n \le N-1. \tag{3.15}$$

The new windowed signal for every point n is $y[n]=x[n].w[n]$. For example, let us apply the Hamming window to the ECG signal discussed in Chapter 1 and compute the magnitude spectrum (shown in Figure 3.10). It can be seen that the use of windowing gives a smoother spectrum (i.e. with less noise). The choice of windowing is outside the scope of this book as it involves ratio analysis of spectral magnitudes of main lobes and side lobes of the window and hence, we'll skip them.

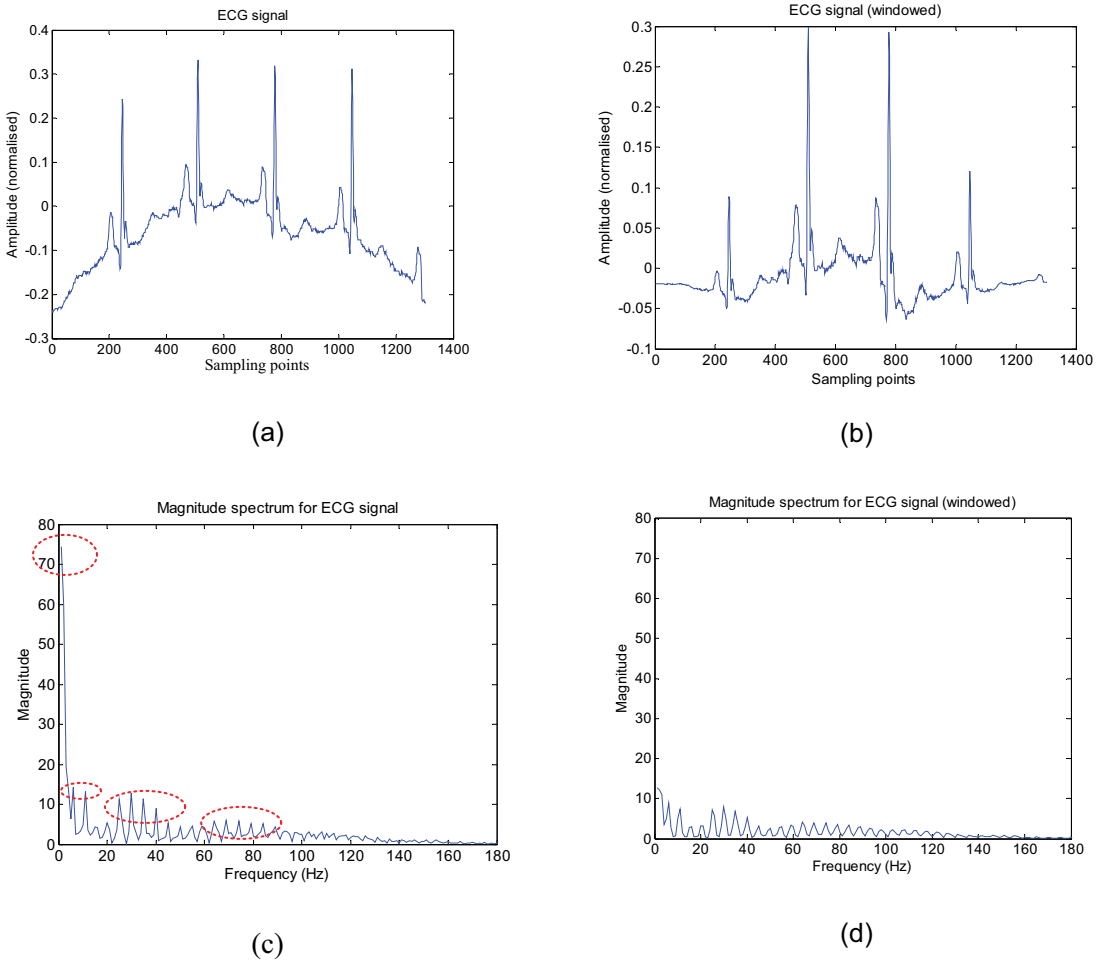


Figure 3.10: Example of spectral leakage: (a) Original ECG signal (b) windowed ECG signal (c)magnitude spectrum for the signal in (a) with noise from spectral leakage denoted with red ellipses (d) magnitude spectrum for the signal in (b).

3.6 Examples of using DFT to compute magnitude spectrum

Example 1

Consider a sinusoidal wave with frequency of 10 Hz generated with Fs=200 Hz using the MATLAB codes:

```
N=200; Fs=200;
x(1:N)=sin(2*pi*(0:N-1)*10/Fs); %Note, MATLAB index starts from 1
```

The magnitude spectrum is computed for this signal and is shown in Figure 3.11.

```
plot(0:N-1,x(1:N)); %Note, MATLAB index starts from 1
xlabel('Sampling points')
ylabel('Amplitude')
title('10 Hz sine wave (N=200)')
figure;
plot(0:N/2,dftmag(1:N/2+1)); %Note, MATLAB index starts from 1
xlabel('Frequency number (m)')
ylabel ('Magnitude')
title ('Magnitude spectrum of a 10 Hz sine wave (N=200)')
```

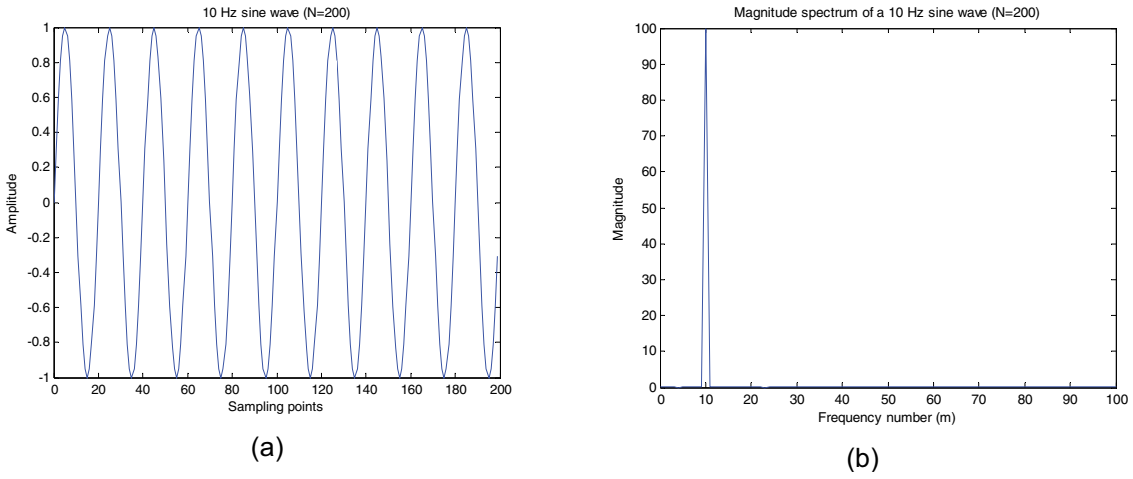


Figure 3.11: A 10 Hz sine wave (a) for length N=200 (b) spectral magnitude.

Now let us assume that the signal is available only up to N=20. We obtain the magnitude spectrum as shown in Figure 3.12. What has happened? The peak is at frequency number, m=1 which still corresponds correctly to 10 Hz as the actual frequency is given by mFs/N.

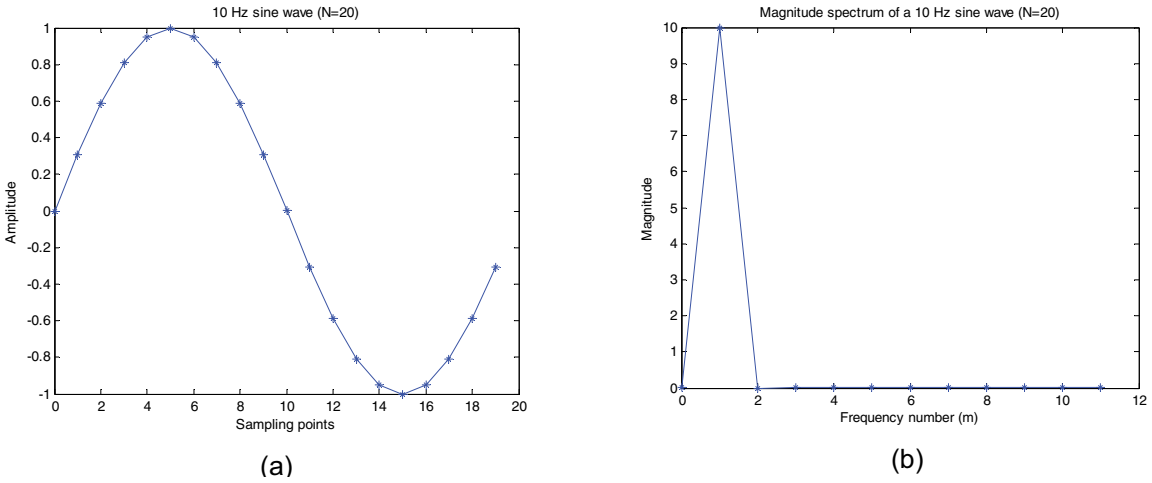


Figure 3.12: A 10 Hz sine wave (a) for length *N*=20 (b) spectral magnitude.

Example 2

Consider a combination of sinusoidal waves with frequencies of 5 Hz and 20 Hz with *Fs*=200 Hz:

```
N=20;fs=200;f1=5;f2=20; A=1; B=1;
x(1:N)=A*sin(2*pi*(0:N-1)*f1/fs)+B*sin(2*pi*(0:N-1)*f2/fs);
plot(0:N-1,x(1:N));
xlabel('Sampling points')
ylabel('Amplitude')
title('Combination of two sine waves (N=20)')
```

The magnitude spectrum is shown in Figure 3.13. As the frequency spacing is 10 Hz[14], to avoid the picket fence effect, we pad the signal with 20 zeros and compute the spectrum. The magnitude spectrum shows a peak at 20 Hz and there is a value at 5 Hz but with a magnitude different to 20 Hz, so there is a problem as both the generated signals are of the same amplitude (i.e. A=B=1).

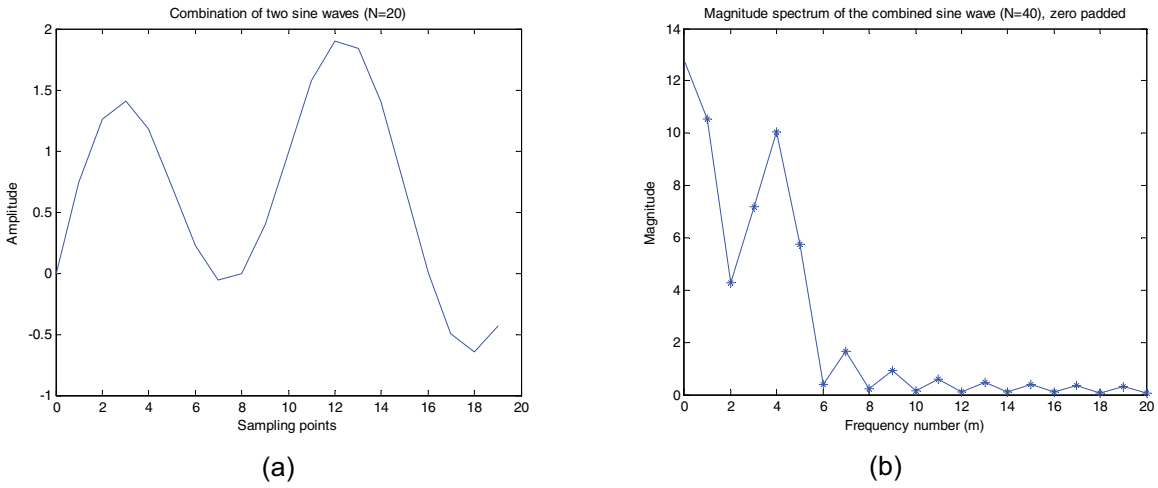


Figure 3.13: Combined sine wave (a) for length *N*=20 (b) spectral magnitude with 20 padded zeros.

DFT generated spectrum will work properly only if there is at least one complete cycle of the signal. Hence, we need to use at least 40 points (since 5 Hz signal completes a cycle in 40 points with *Fs*=200 Hz). See Figure 3.14 that has been generated with *N*=40 which shows both the peaks correctly at frequency numbers 1 and 4. But, in reality, we need to include multiple cycles in order to obtain reliable DFT information.

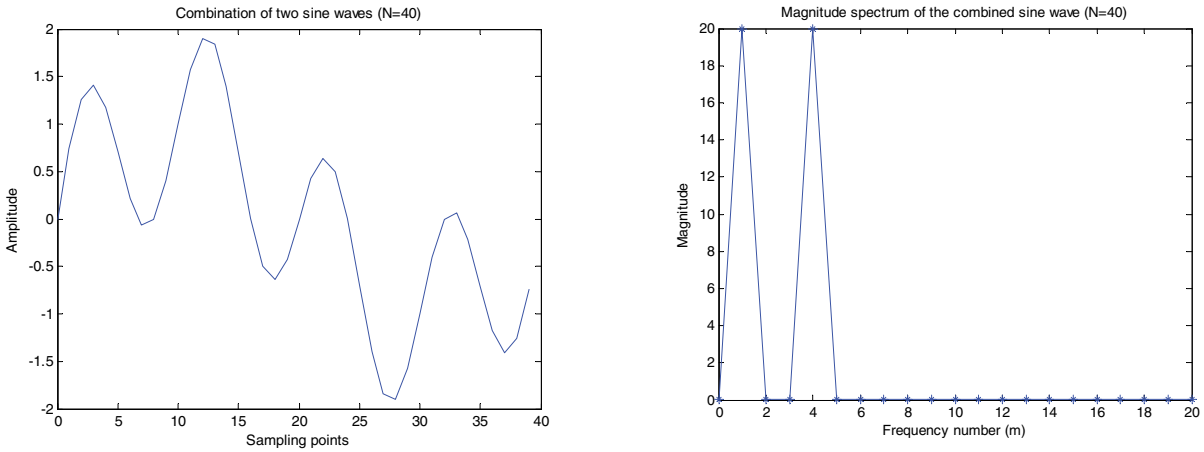


Figure 3.14: Combined sine wave (a) for length N=40; (b) spectral magnitude.

3.7 Periodogram

Periodogram is a measure of power spectrum and is similar to the square of magnitude spectra- actually, the only difference is the scale factor. If we compute magnitude spectra using DFT, square, scale and then convert to decibels (dB); we will have periodogram.

The following code can be used to obtain the periodogram of a sinusoidal wave:

```
N=200; Fs=100; f1=5;
x=sin(2*pi*(1:N)*f1/Fs);
plot(x);
y=fft(x,N);
py = power(abs(y),2);
pdgrm=10*log10(py/N); %normalise by N
plot(pdgrm(1:N/2));
```

Windowing is not necessary for this signal[15] but when a window is used, the normalisation is no longer *N* but *CN* where *C* is given by

$$C = \frac{1}{N}\sum_{n=0}^{N-1}\left|w[n]\right|^2 \tag{3.16}$$

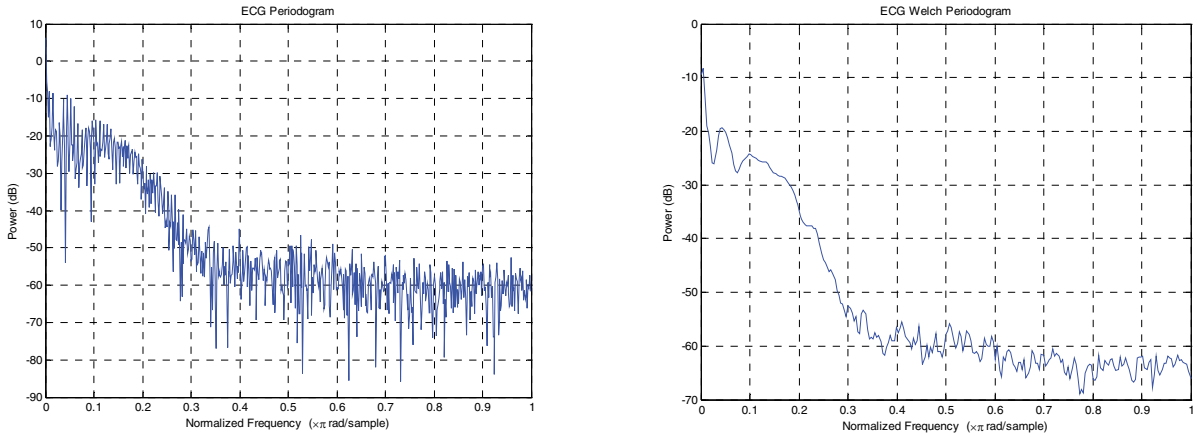
and $w[n]$ is the used window. This is to avoid any bias in the estimate due to the use of the window. The obtained spectrum is known as modified periodogram [2].

3.7.1 Welch method

Welch method is an improved method of computing the periodogram. The first step in this method is to divide the data into *K* overlapping segments each containing *M* sample points. The non-overlap length is *D*, i.e. the segments overlap with *M-D* samples. Thus *K=(N-M)/D +1* and the percentage overlap is 100**(M-D)/M*.

Next, we apply the conventional method to compute periodogram (or modified periodogram) for each segment and finally average all the periodograms. In MATLAB, *pwelch* function can be used where the percentage overlap is 50% with *K*=8 and Hamming window is used by default.

As an example, let use revisit the ECG signal used earlier. The first plot is obtained by using the periodogram code given earlier, while the second is obtained by using *pwelch* function. From the plots, we can see that the Welch method gives a smoother power spectrum than conventional periodogram. A note about the *abscissa* scale when using *pwelch* function in MATLAB: the frequency scale is normalised from 0 to 1 representing the range 0 to Fs/2.



(a) (b)

Figure 3.15: Power spectrum for ECG signal (a) using periodogram; (b) using Welch periodogram.

3.8 References

[1] R. Shiavi, *Introduction to Applied Statistical Signal Analysis (2nd edition)*, Academic Press, 1999.

[2] S.K. Mitra, *Digital Signal Processing: A Practical Approach (3rd edition)*, McGraw-Hill, 2006.

[3] J.G. Proakis and D.K. Manolakis, *Digital Signal Processing (4th edition)*, Prentice Hall, 2006.

4 Digital Filtering

In this chapter, we'll study digital filtering methods. Specifically, we'll look into the following:

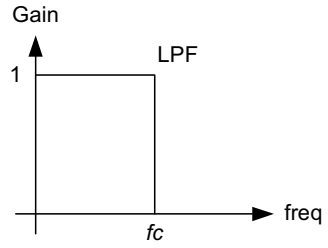
- Filter specifications
- Filtering in frequency domain
- Filtering in time domain
- Simple filter design – Sum and difference (SD) filters
- Finite Impulse Response (FIR) filters
- Infinite Impulse Response (IIR) filters (using MATLAB functions)

4.1 Filter Specifications

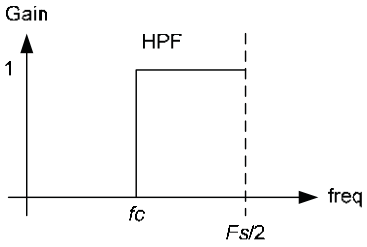
Filtering is the process of keeping components of the signal with certain desired frequencies and removing components of the signal with certain undesired frequencies. Very often, we keep the gain of the required frequency components to 1 or close to 1 and the gain of the undesired frequency components will be 0 or close to 0. In general, there are 4 types of filter: low-pass filter (LPF), high-pass-filter (HPF), band-pass filter (BPF) and band-stop filter (BSF). Each filter will have specific characteristics:

- Passband – the range of frequency components that are allowed to pass
- Stopband – the range of frequency components that are suppressed
- Passband ripple – ripples in the passband, the maximum amount by which attenuation in the passband may deviate from gain (which is normally 1)
- Stopband ripple – ripples in the stopband, the maximum amount by which attenuation in the stopband may deviate from gain (which is normally 0)
- Stopband attenuation – the minimum amount by which frequency components in the stopband are attenuated
- Transition band – the band between the passband and the stopband.

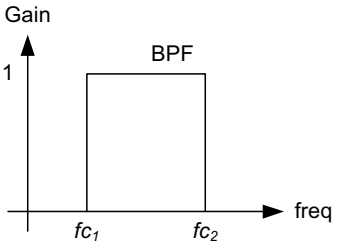
Magnitude frequency responses of ideal filters are shown in Figure 1 where *fc* is the cut-off frequency with *Fs* as the sampling frequency.



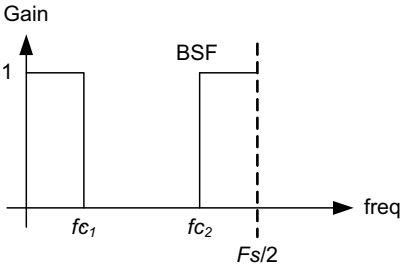
Passband: $0 \le f \le fc$
Stopband: $fc \le f \le Fs/2$
(a)



Passband: $fc \le f \le Fs/2$
Stopband: $0 \le f \le fc$
(b)



Passband: $fc_1 \le f \le fc_2$
Stopband: $0 \le f \le fc_1$ and $fc_2 \le f \le Fs/2$
(c)



Passband: $0 \le f \le fc_1$ and $fc_2 \le f \le Fs/2$
Stopband: $fc_1 \le f \le fc_2$
(d)

Figure 4.1: Ideal magnitude frequency responses (a) LPF (b) HPF (c) BPF (d) BSF.

4.1.1 Low-pass filter

A LPF passes all low-frequency components below the cut-off frequency, *fc* and blocks all higher frequency components above *fc*. Figure 4.2 shows the magnitude frequency response of a LPF in reality, where we can't design 'square' type of filters as shown in Figure 4.1. So, there needs to be transition band between the passband and stopband. The edge frequencies are the end frequencies of passband (*fp*) or stopband (*fs*). So, a practical LPF will allow frequency components below *fp* and remove components higher than *fs*.

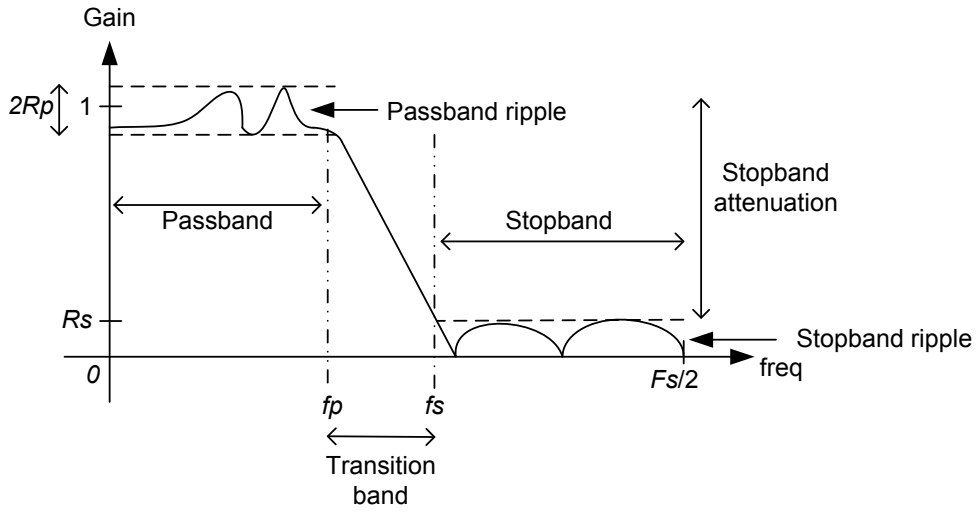


Figure 4.2: Magnitude frequency response of a LPF.

For example, consider a combination of three sinusoidal signals: 2 Hz, 5 Hz and 11 Hz as shown in Figure 4.3.

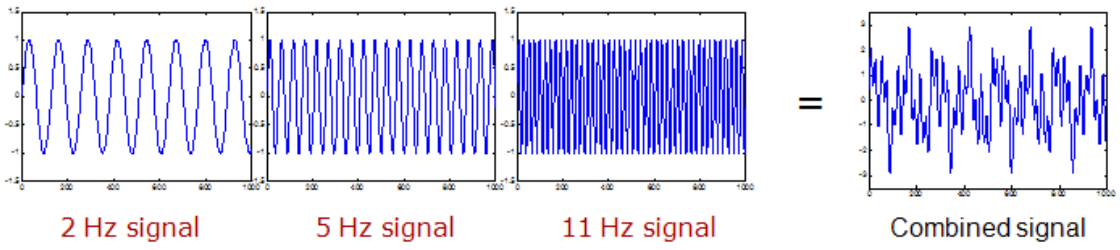


Figure 4.3: A combination of three sinusoidal signals.

The final output signals after LPF at *fp*=3 Hz with *fs*=4 Hz and *fp*=8 Hz with *fs*=9 Hz are shown in Figure 4.4.

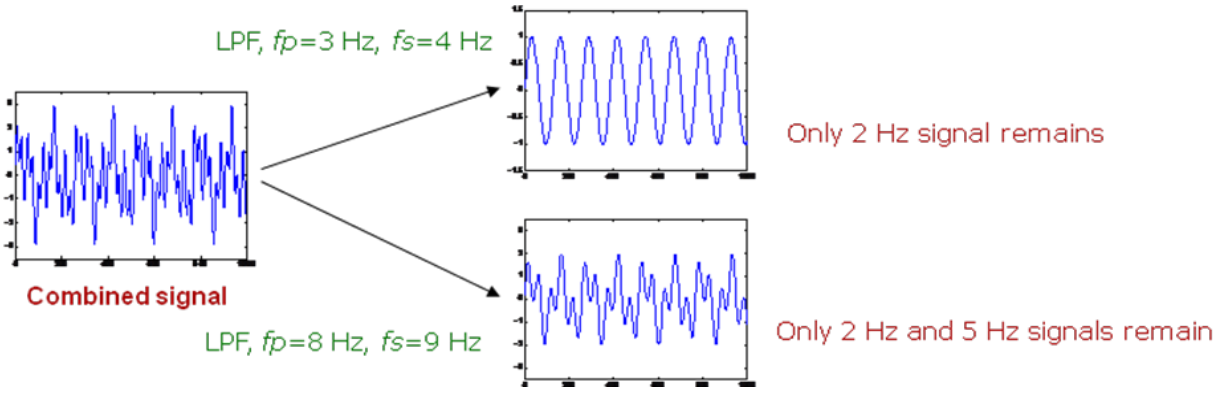


Figure 4.4: LPF of the three sinusoidal signals.

4.1.2 High-pass filter

HPF passes all high-frequency components above the cut-off frequency, *fc* and blocks all lower frequency components below *fc*. The magnitude frequency response of a HPF in reality is shown in Figure 4.5 where it allows frequency components higher than *fp* and remove components below *fs*.

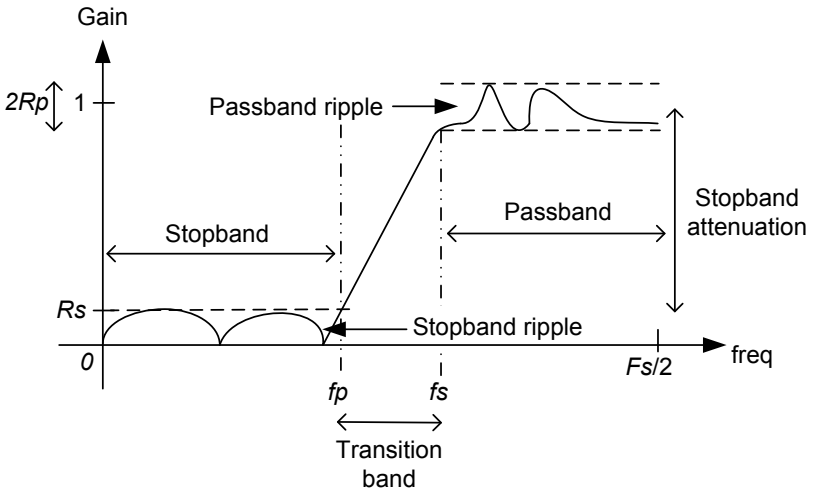


Figure 4.5: Magnitude frequency response of a HPF.

Consider the same combination of three sinusoidal signals: 2 Hz, 5 Hz and 11 Hz as previously. The final output signals after HPF at *fs*=3 Hz with *fp*=4 Hz and *fs*=8 Hz with *fp*=9 Hz are shown in Figure 4.6.

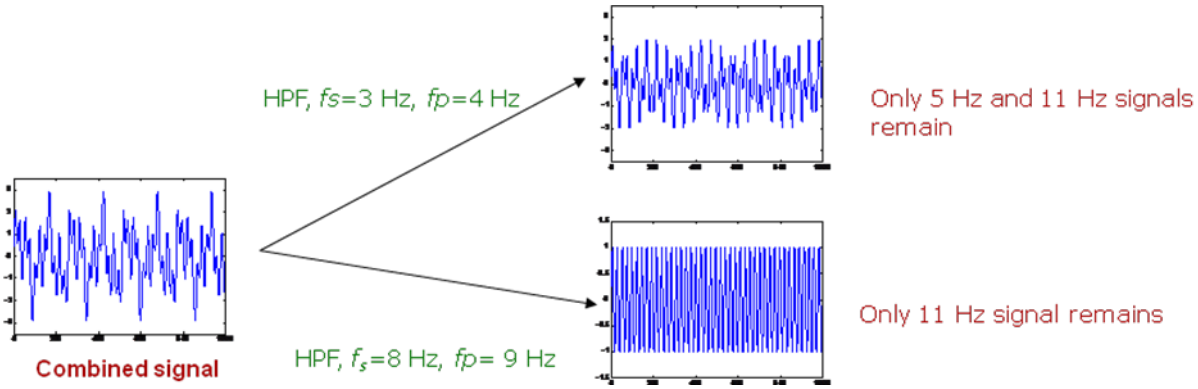


Figure 4.6: HPF of the three sinusoidal signals.

4.1.3 Band-pass and band-stop filters

BPF passes all frequency components between edge passband frequencies, fp_1<freq$_{(allow)}$<fp_2 and blocks all frequencies below and above edge stopband frequencies, freq$_{(block)}$<fs_1; freq$_{(block)}$>fs_2. A BPF can be designed using a LPF and HPF. BSF passes all frequency components lower and higher than edge passband frequencies, freq$_{(allow)}$<fp_1; freq$_{(allow)}$>fp_2 and blocks all frequencies between fs_1<freq$_{(block)}$<fs_2. The magnitude frequency responses of a BPF and a BSF are shown in Figures 4.7 and 4.8.

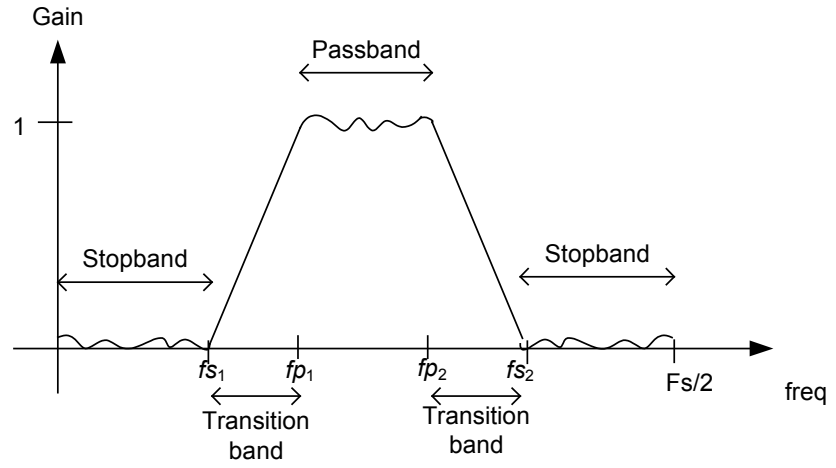


Figure 4.7: Magnitude frequency response of a BPF.

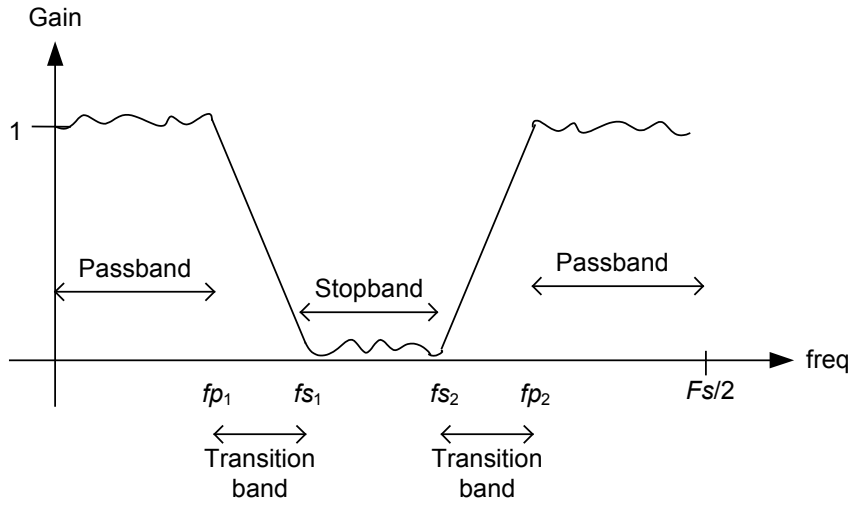
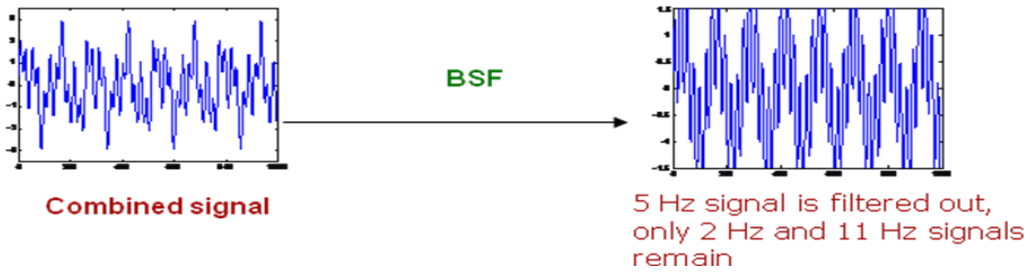


Figure 4.8: Magnitude frequency response of a BSF.

Figure 4.9 shows the output signals after applying BPF at fp_1=4 Hz, fp_2=6 Hz, fs_1=3 Hz, fs_2=7 Hz and BSF at fp_1=4 Hz, fp_2=6 Hz, fs_1=3 Hz, fs_2=7 Hz for the combination of the sinusoidal signals.

(a)



(b)

Figure 4.9: (a) BPF (b) BSF of the three sinusoidal signals.

4.2 Direct filtering in frequency domain

Filtering can be done directly in the frequency domain using the following steps:

- Obtain the Discrete Fourier Transform (DFT) of the signal (from 0 to *Fs*);
- Set to zero the values that are not in the required frequency range i.e. apply a rectangular window;
- Compute the Inverse Discrete Fourier Transform (IDFT).

For example, let use generate a combination of two sinusoidal signal with *f*1=8 Hz and *f*2=25 Hz with *N*=100, *Fs*=200 Hz (shown in Figure 4.10) and say, we wish to design a LPF with *fp*=10 Hz and *fs*=12 Hz. Compute `y=fft(x)` in MATLAB and apply the rectangular window, i.e. set the values `y(7:95)=0`. As MATLAB indexing starts from 1, `y(1:6)` represents DFT values from 0 to 10 Hz[16], which represents the passband range, the stopband range from 12 Hz to 100 Hz is represented by `y(7:51)`. Due to symmetry, we also need to create mirror images of the passband and stopband resulting in the rectangular window as shown in Figure 4.11 (a).

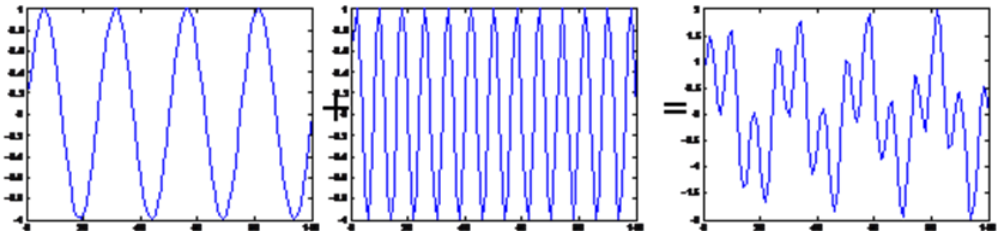


Figure 4.10: Combination of two sinusoidal signals (*f*1=8 Hz and *f*2=25 Hz).

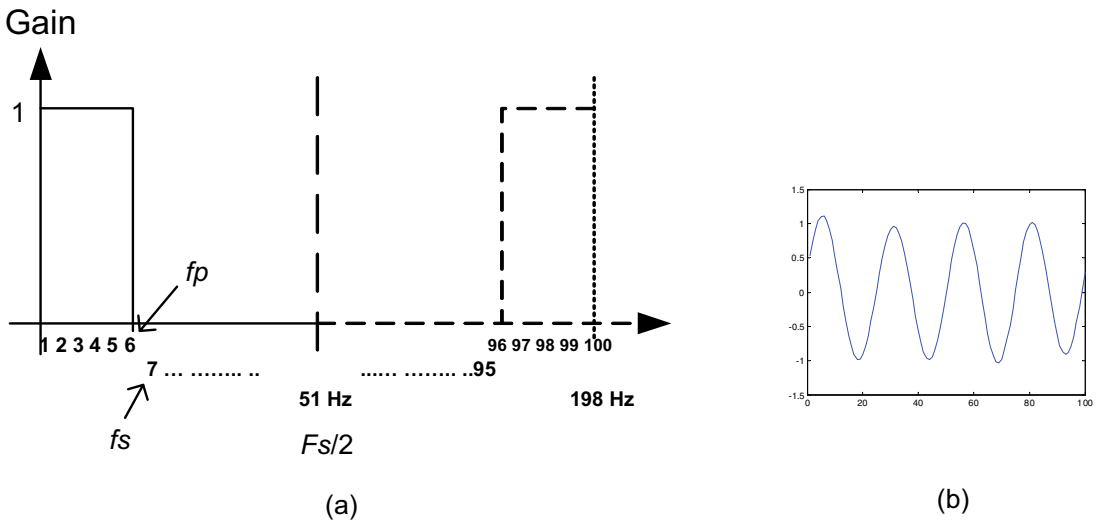


Figure 4.11: Direct LPF (a) rectangular window with *fp*=10 Hz and *fs*=12 Hz (b) filtered output.

Next, compute `yf=ifft(y,'symmetric')` and the low pass filtered signal is obtained as shown in Figure 4.11 (b). In MATLAB, it is useful to force conjugate symmetry, else complex values could be obtained due round-off errors in the `fft` and `ifft` operations. Figure 4.12 shows the whole procedure for the discussed example. This direct filtering method is simplistic to understand but has the disadvantage of high computation cost and requires chunks of data (i.e. real-time filtering is not possible). Thus we normally use finite impulse response (FIR) or infinite impulse response (IIR) filters to perform filtering.

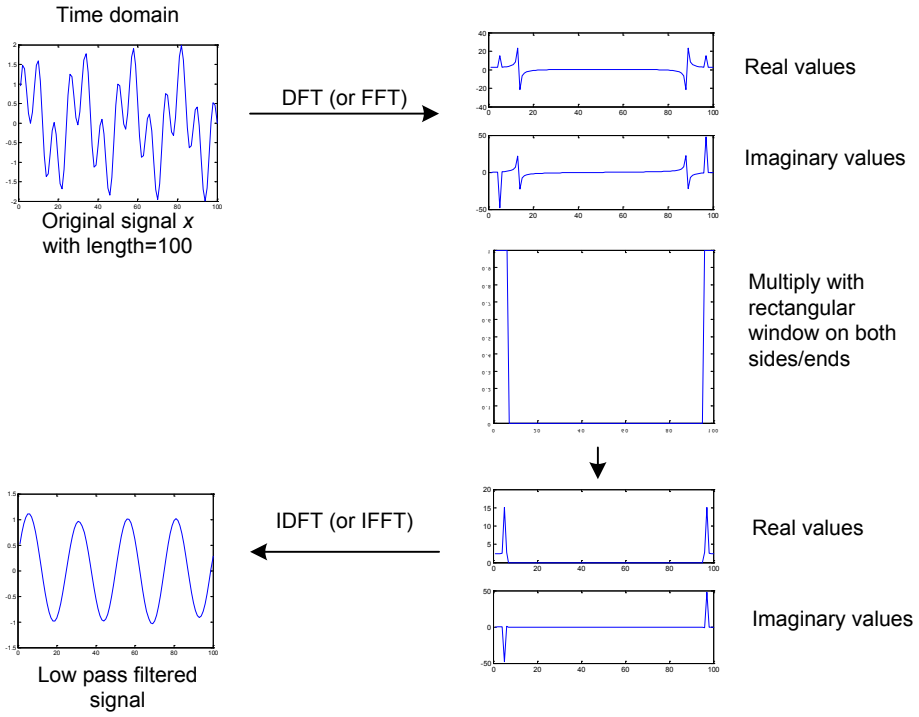


Figure 4.12: LPF example using direct filtering method.

4.3 Time domain filtering

To solve the problems of direct filtering, we could filter in time domain and there are several time domain filtering methods. We will look at design of simple FIR filters and IIR filters using MATLAB. The output from an IIR digital filter is made up of previous inputs and previous outputs:

$$y[n] = \sum_{k=1}^{M} B[k]x[n-k] + \sum_{j=1}^{N} A[j]y[n-j]. \tag{4.1}$$

where B and A are the filter coefficients. The output from a FIR digital filter is made up of previous inputs only, so there is no feedback:

$$y[n] = \sum_{k=1}^{M} B[k]x[n-k]. \tag{4.2}$$

Figure 4.13 shows an example comparing direct filtering in the frequency domain with time domain filtering. It should be obvious from this figure that filtering in time domain is computationally less complicated.

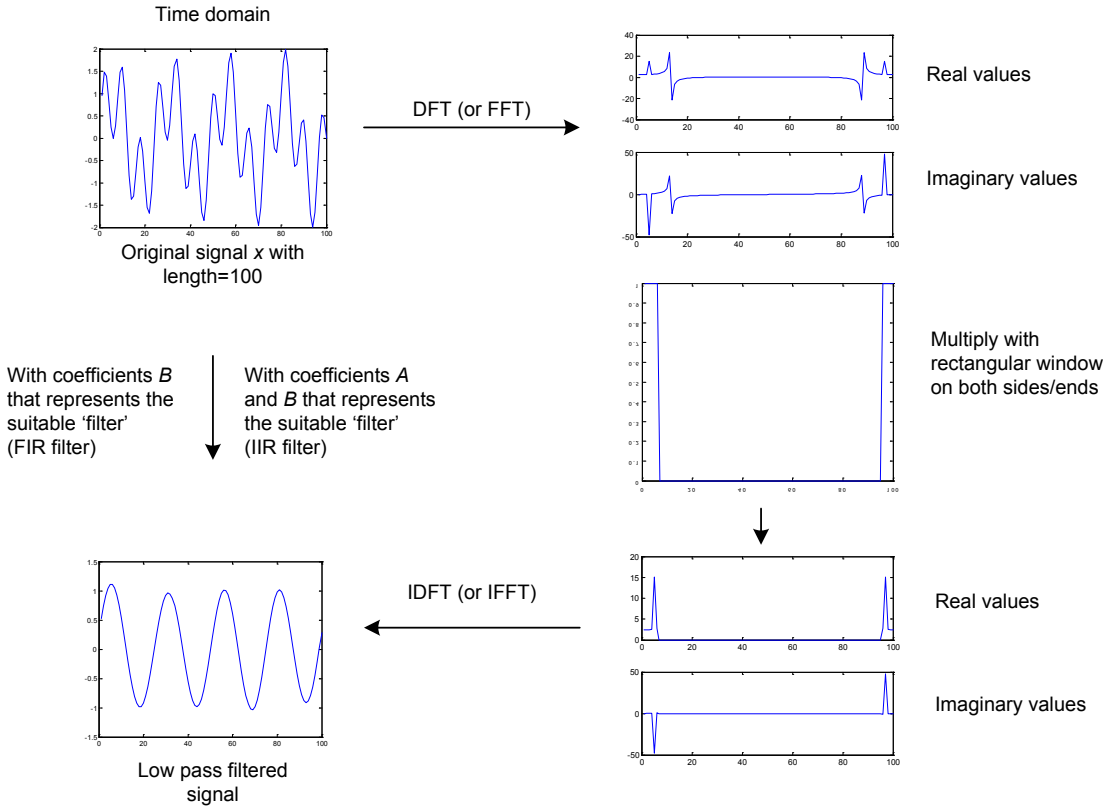


Figure 4.13: LPF comparison using direct frequency filtering and time domain methods.

4.4 Simple FIR filters

Simple FIR filters are also known as sum and/or difference filter. Consider a sum filter

$$y[n] = \frac{1}{2}(x[n] + x[n-1]), \tag{4.3}$$

for every data $x[n]$ in the signal; this simple addition can be shown using Z-transform to act as a LPF! Considering (4.2), the filter coefficients are $B[1]$=0.5 and $B[2]$=0.5.

For hardware design, the block diagram is shown in Figure 4.14.

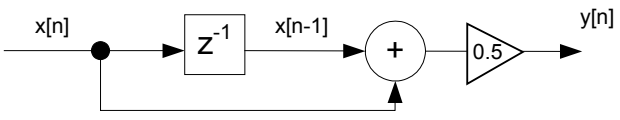


Figure 4.14: Block diagram of a simple sum filter.

The advantages of such a filter are:

- Only one adder and one delay[17] block is needed, so simple design and low cost;
- The filter coefficients are finite values (in this case they are 0.5), so no errors caused by round-off;
- It is an FIR filter, so it is stable[18];
- It's phase response is linear (more on this later).

However, the magnitude (gain) response is not very good as it is far from the ideal 'square' filter (see Figure 4.15).

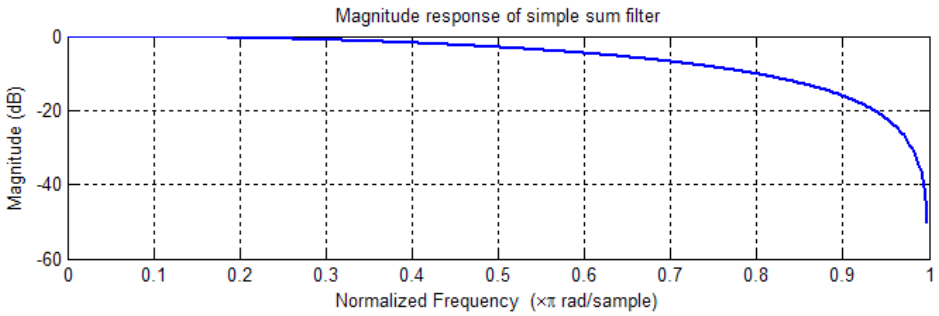


Figure 4.15: Magnitude response of the simple sum filter.

The magnitude (gain) at normalised frequency 0 is 1 (i.e. 0 dB[19]) and the stopband frequency (when gain drops to 0) is at π rad/sample or at *Fs*/2 Hz, where *Fs* is the sampling frequency. So, it appears that there is no stopband and for these cases, we use the 3 dB cut-off as the passband frequency. The 3 dB cut-off frequency is defined as the frequency when the gain drops 3 dB from maximum gain of 1, which is 0 dB.

So when energy is half, i.e. gain=$(1/2)^{0.5}$=0.7071, we have $20\log_{10}(0.7071)$=-3 dB. From Figure 4.15, we can see that the 3 dB cut-off frequency (when gain=0.7071) is approximately ≈ 0.5 π rad/sample or ≈ *Fs*/4 Hz. This is the edge passband frequency and so the passband is from 0 to *Fs*/4 Hz and transition band is from *Fs*/4 to *Fs*/2.

Similarly, a HPF can be designed using a difference filter:

$$y[n] = \frac{1}{2}(x[n] - x[n-1]). \tag{4.4}$$

The block diagram is shown in Figure 4.16.

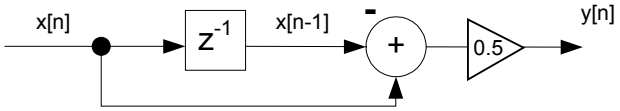


Figure 4.16: Block diagram of a simple difference filter.

And the magnitude response is given in Figure 4.17.

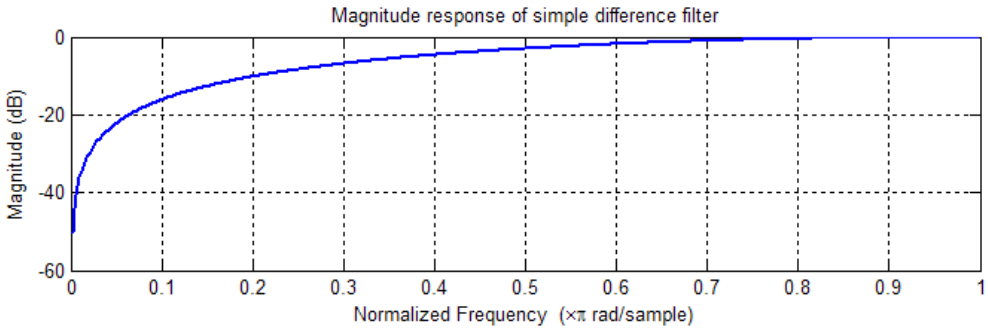


Figure 4.17: Magnitude response of the simple difference filter.

The stopband frequency (when gain=0) is at 0 Hz, i.e. there is no stopband. The gain at *Fs*/2 Hz or π rad/sample is 1 and hence, the edge passband frequency is at *Fs*/4 Hz (using 3 dB cut-off approach). The passband width is from *Fs*/4 to *Fs*/2 Hz.

4.4.1 Increasing the order of the simple filter

The order of the filter can be increased to obtain a smaller passband width and to obtain a frequency response closer to the ideal 'square' filter. The sum filter in (4.3) had an order of 1; if we cascaded another first order sum filter, we will have the block diagram shown in Figure 4.18 (we'll drop the constants for simplicity of discussion):

Figure 4.18: Magnitude response of two first order sum filters (effectively a second order sum filter).

Solving for $z[n]$ in Figure 4.18 will give

$$\begin{aligned} z[n] &= y[n] + y[n-1], \\ &= x[n] + x[n-1] + x[n-1] + x[n-2], \\ &= x[n] + 2x[n-1] + x[n-2]. \end{aligned} \tag{4.5}$$

Eq. (4.5) could be easily verified by replacing values for $x[n]$. For example, using $x[1]$=3, $x[2]$=2 and $x[3]$=5 and computing z[3] for the single cascaded second order filter will give 12. Computation using two first order filters (i.e. $y[2]$ and $y[3]$) will give the same result.

It should be noted that $z[n]$ in the example above will be defined only for n=3 onwards if $x[1]$ is the starting point of the signal, i.e. for every order M, M initial data points will be lost in filtering. Likewise $y[n]$ in (4.3) is defined only from $y[2]$ onwards.

For order M, we have

$$y(n) = \sum_{r=0}^{M} {}^{M}C_r x[n-r] \text{ where } {}^{M}C_r = \frac{M!}{r!(M-r)!}. \tag{4.6}$$

As an example, for order M=3, we will have

$$y(n) = x[n] + x[n-1] + 3x[n-2] + x[n-3]. \tag{4.7}$$

The magnitude response is given in Figure 4.19.

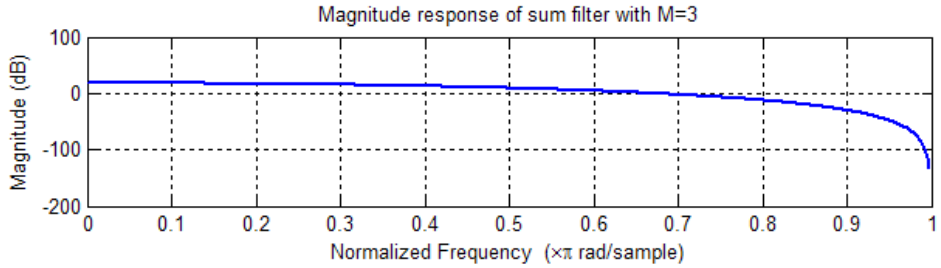


Figure 4.19: Magnitude response of the third order sum filter.

The passband is about 0.302 π rad or ≈*Fs*/6. It could be seen that with increasing order, the passband is becoming smaller without any change in stopband. Also, the response is becoming closer to the ideal 'square'. So, we can increase/decrease *M* depending on the requirements. The 3-dB cut-off frequency is given by

$$fp = \frac{2}{\pi}\cos^{-1}(2^{-1/2M}). \tag{4.8}$$

Similarly for the HPF with order *N*, we will have

$$fp = \frac{2}{\pi}\sin^{-1}(2^{-1/2N}), \tag{4.9}$$

and the magnitude response as

$$y(n) = \sum_{r=0}^{N} (-1)^r \left({}^{N}C_r x[n-r] \right). \tag{4.10}$$

4.4.2 BPF design using sum and difference filter

Similarly, a BPF can be designed using a combination of LPF and HPF. This BPF is known as sum and difference (SD) filter. Different orders, M and N can be chosen to obtain the required frequency response [1]:

$$G_{M,N}(f) = (2\cos \pi fT)^M \left| 2\sin \pi fT \right|^N / Gain_{cf}, \tag{4.11}$$

where $Gain_{cf}$ is the gain at centre frequency given by

$$fcentre = \frac{f_s}{2\pi} \cos^{-1}\left(\frac{M-N}{M+N} \right). \tag{4.12}$$

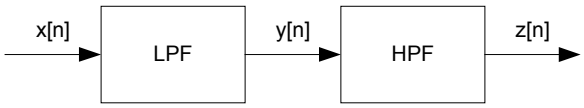


Figure 4.20: BPF design using SD filter.

For example, with filter orders of M=28 and N=8 gives the centre frequency of 40 Hz when Fs=256 Hz. The approximate 3 dB bandwidth is from 32 to 48 Hz (rounded to the nearest integer) and the gain amplification at 40 Hz is approximately 47.21. Figure 4.20 shows this example using different filter orders but with similar centre frequency (which is dependent on ratio of M/N).

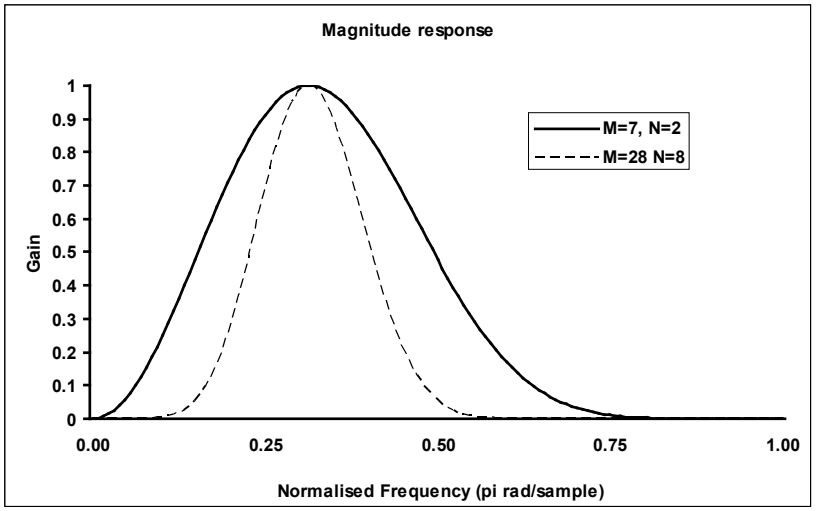


Figure 4.21: BPF magnitude response for M=7, N=2 and M=28 and N=4 (note: ordinate shows gain as actual values and not as dB).

As another example, let us obtain the band pass FIR filter expression for orders, LPF, M=4 and HPF, N=1, i.e. obtain the band pass FIR equation that expresses $z[n]$ in terms of $x[n]$ and delays of $x[n]$. Using (4.6), obtain $y[n]$ in term of $x[n]$ and using (4.10), obtain $z[n]$ in terms $y[n]$. Next, replace $y[n]$ in the latter expression to arrive at

$$z[n] = x[n] + 3x[n-1] + 2x[n-2] - 2x[n-3] - 3x[n-4] - x[n-5]. \tag{4.13}$$

4.5 FIR filter design using window method

The SD filter that we studied in the previous section is simple to design but for practical purposes, we often need filters that can be tailored to suit our required specifications. Consider doing an inverse DFT of the ideal LPF shown in Figure 4.1 (a) to obtain what is known as the impulse response, which are basically the filter coefficients[20].

The impulse response is actually the *sinc* function given by

$$h_{LPF}[n] = \frac{\sin(2\pi f_c n)}{\pi n}, \qquad -\infty < n < \infty \tag{4.14}$$

It would be obvious that we will not be able to use h_{LPF} as the filter coefficients as the length is infinite. So we could use a rectangular window, $w[n]$ to truncate the impulse response. However, by using a finite set of coefficients (i.e. impulse response), the shape of the magnitude response is changed with ripples showing up as in Figure 4.20. This is known as the Gibbs phenomenon – oscillatory behaviour in the magnitude responses caused by truncating the ideal impulse response function (i.e. the rectangular window has an abrupt transition to zero). Gibbs phenomenon can be reduced by

- using a window that tapers smoothly at each end such as Hamming, Hanning, triangular etc (refer to Section 3.5 in the previous chapter);
- providing a smooth transition from passband to stopband in the magnitude specifications.

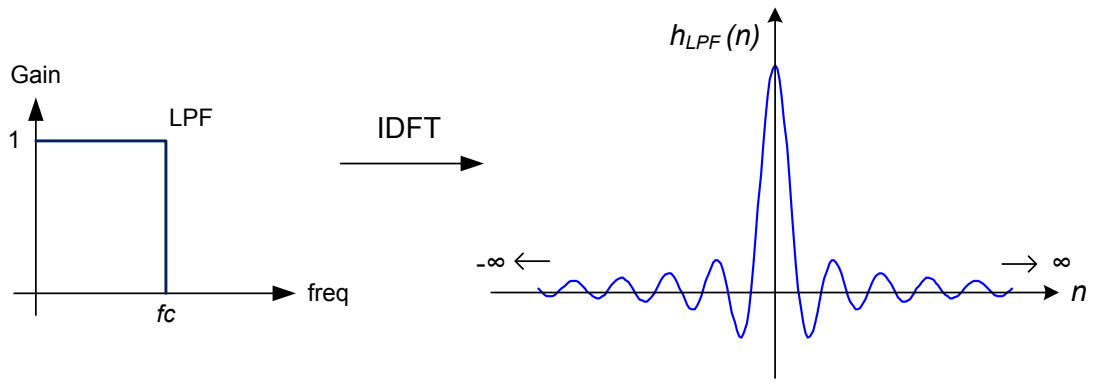


Figure 4.21: Impulse response for LPF.

But a question which one may now raise is on the appropriate length of the filter. The filter length (i.e. order) affects the magnitude response. With a length increase, the number of ripples in both passband and stopband increases[21] but with a corresponding decrease in the ripple widths, i.e. as N increases, the magnitude response approaches closer to the ideal LPF (see Figure 4.21). Similar oscillatory behaviour could be observed in the magnitude responses of the truncated versions of other types of ideal filters.

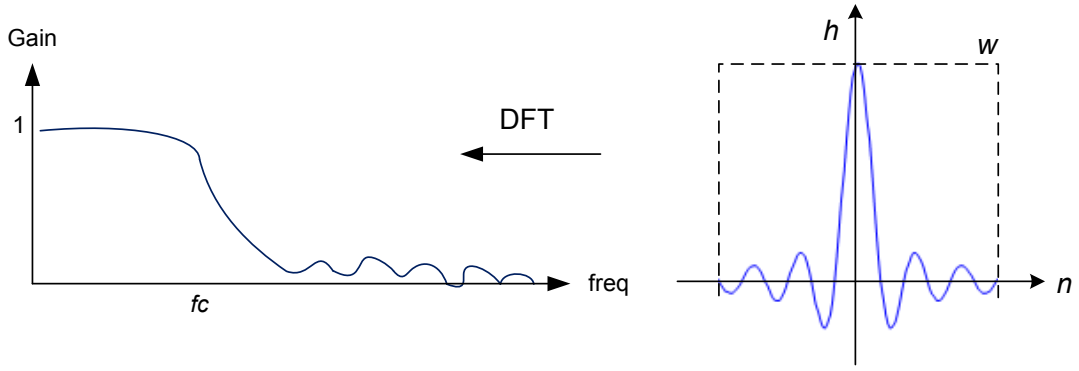
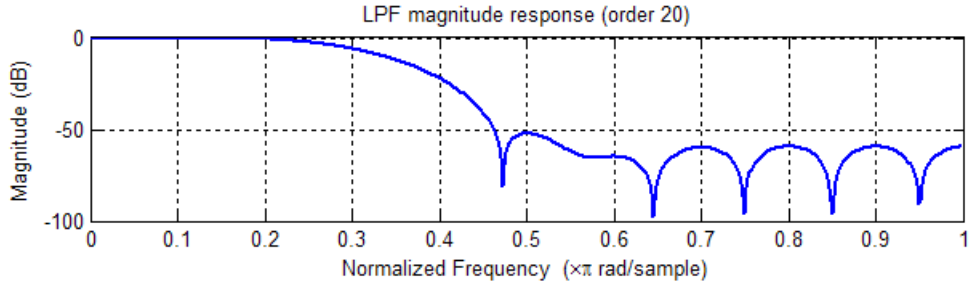
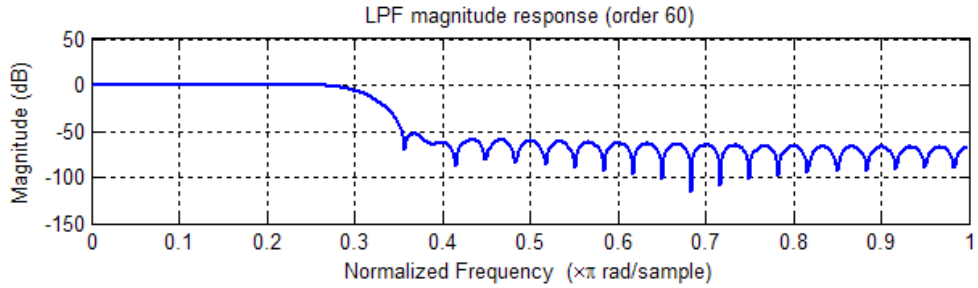


Figure 4.22: Truncated impulse response for LPF and Gibbs phenomenon.



(a)



(b)

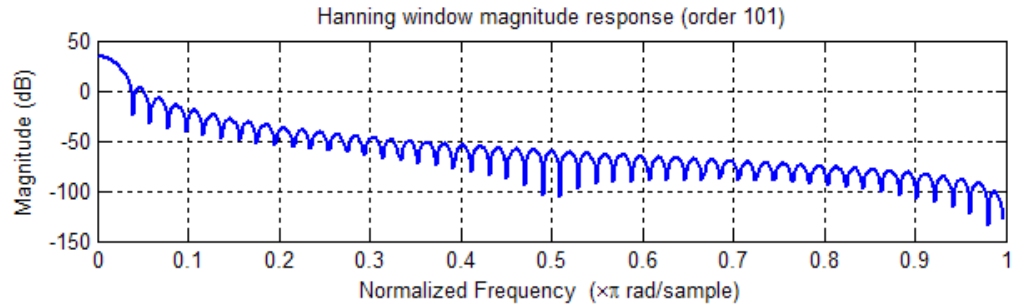
Figure 4.23: LPF magnitude response for different lengths (a) 20 (b) 60.

There are several methods such as Kaiser and Bellanger formulas to select the 'suitable' order i.e. the smallest length that can meet the requirements. Kaiser's formula [2] is given by:

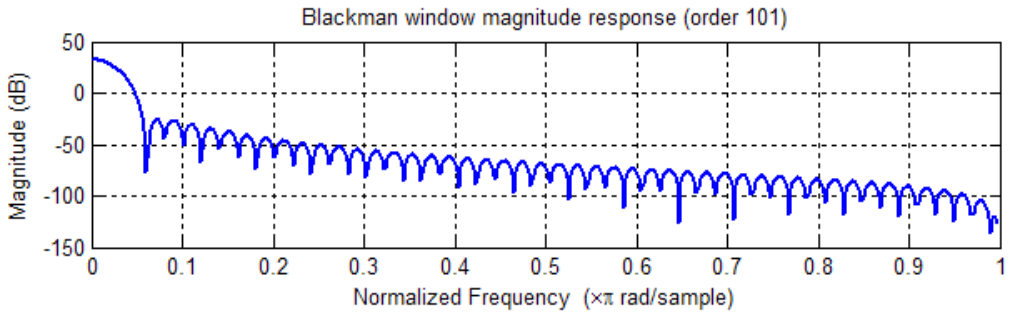
$$N \cong \frac{-20\log_{10}\left(\sqrt{\delta_p \delta_s}\right) - 13}{14.6(f_s - f_p)/F_s} \tag{4.15}$$

where δp and δs are the ripples in the passband and stopband, respectively while fp and fs are passband and stopband edge frequencies. It should be obvious that the actual location of the transition band is immaterial, only the transition width matters.

The next issue is on the choice of window, which could be decided using the areas under main lobe and side lobes. Figure 4.22 shows two windows, Hanning and Blackman designed using order 101; the main lobe is the first ripple while other ripples are known as side lobes.



(a)

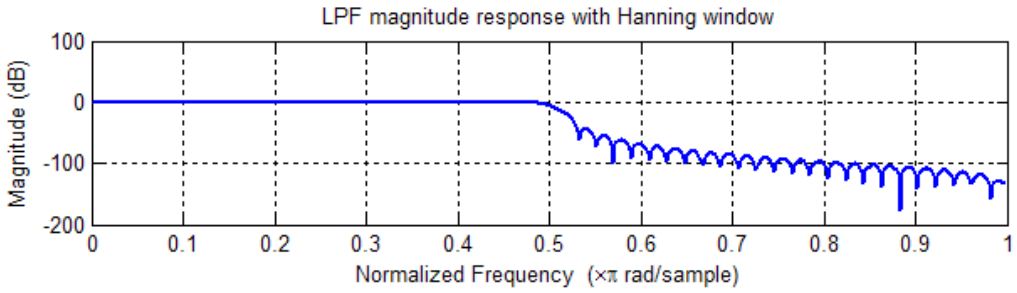


(b)

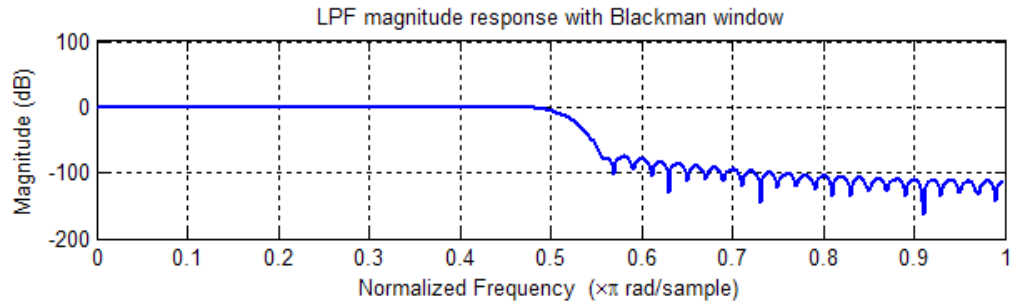
Figure 4.24: Window magnitude response (a) Hanning (b) Blackman.

To ensure a fast transition from passband to stopband, the window should have a very small main lobe width (i.e. area under the main lobe should be small). For a reduction in the ripples, the area under the sidelobes should be small (i.e. to increase the stopband attenuation, we need to decrease the sidelobe amplitudes).

Most of the time, a compromise has to be met with regards to these two requirements (smaller main lobe with smaller side lobes). Consider Figure 4.23, which shows LPF design with cut-off at *Fs*/2, i.e. 0.5 in normalised frequency using the two windows shown in Figure 4.22. From Figure 4.23, we can see that Blackman window, which has smaller area under side lobes but bigger main lobe area, has a higher attenuation in the stopband but with a higher width transition band. The case for Hanning window is the opposite with smaller transition band but with lower stopband attenuation as it has smaller main lobe area but bigger side lobes area.



(a)



(b)

Figure 4.25: LPF design using windows (a) Hanning (b) Blackman.

With the above discussions, we are now ready to design FIR filters using MATLAB. The function `fir1` could be used to obtain the filter coefficients. For example, using Hanning window with length 101 and cut-off at normalised frequency of 0.5:

```
B=fir1(100,0.5,hanning(101));
```

which gives a filter of length 101.

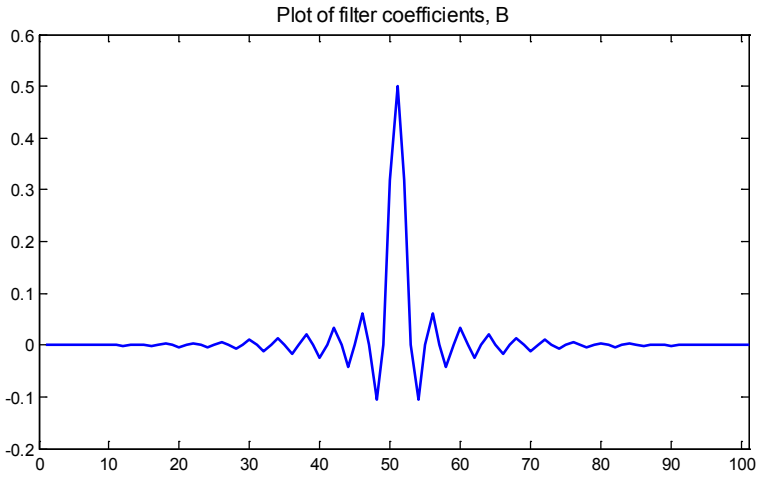


Figure 4.26: Example of FIR filter coefficients.

It can be noticed that the filter coefficients are symmetric; this is important as only then the phase response will be linear[22] in the passband (see Figure 4.25). In chapter 3, we discussed the fact that frequency response can consist of both magnitude and phase. The frequency (both magnitude and phase) response could be obtained using:

```
freqz(B,1); % A is set to 1 for FIR filters
```

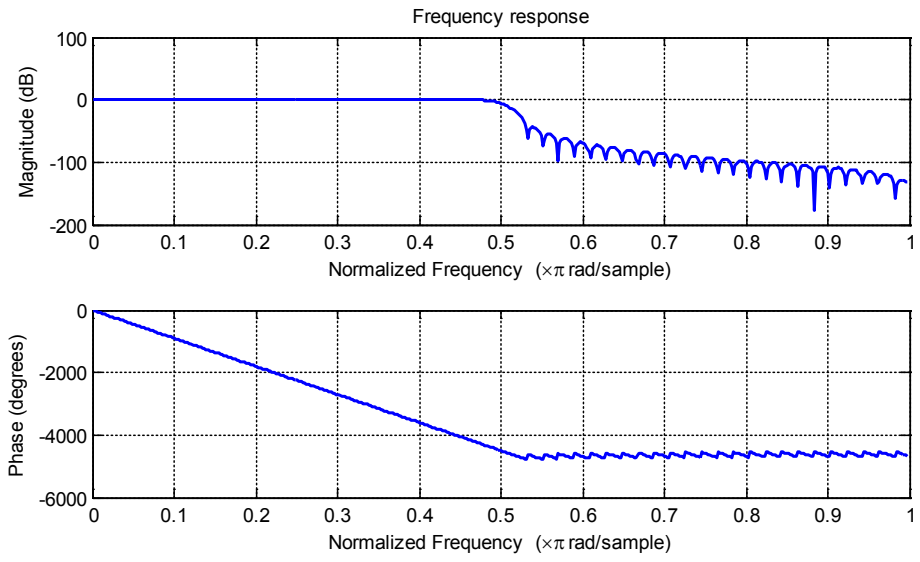


Figure 4.27: Frequency response of the filter coefficients in Figure 4.24.

Let us apply the designed LPF to filter out certain unwanted spectral components. Consider a combination of three sinusoidal waves with normalised frequency components of 0.8, 0.4 and 0.2:

```
i=1:1000; Fs=1000;
x=sin(2*pi*(400/Fs)*i)+sin(2*pi*(200/Fs)*i)+sin(2*pi*(100/Fs)*i);
```

Now, apply the LPF with cut-off at normalised frequency of 0.5:

```
filtered_x=conv(x,B);
```

Basically, the `conv` function implements the convolution of data x with the coefficients B, i.e. it implements (4.2). The spectral plots obtained before and after filtering are shown in Figure 4.26.

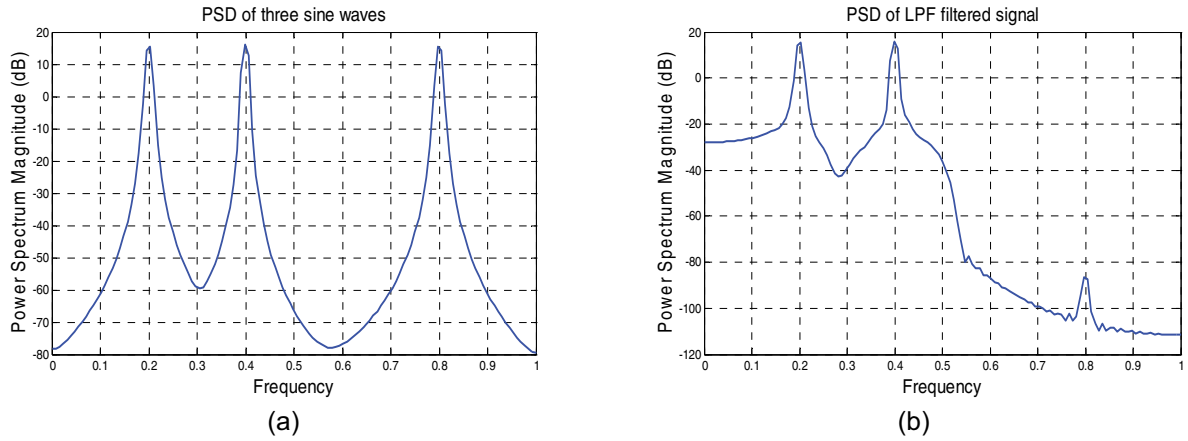


Figure 4.28: PSD (a) before (b) after applying the LPF filter.

From Figures 4.26, the component at normalised frequency 0.8 is still present although attenuated substantially and could be further removed using a higher filter order N or using an IIR filter that has a sharper transition band with higher stopband attenuation ability.

4.6 IIR Filter design

IIR filters have the ability to produce sharper transition band then FIR filters for the same order. However, IIR filters have several disadvantages:

- as there is a feedback element (see Eq. 4.1), there is a chance of instability;
- the filter coefficients are not normally integer, so can have round-off errors;
- hardware design is more complicated;
- most importantly, their phase response is not linear.

It is difficult to design IIR filters digitally. So, the approach is to design analogue IIR filters, then use a bilinear method to transform the analogue filter to digital:

1. convert the digital filter specification into an analogue prototype low-pass filter specification;
2. determine the analogue low-pass filter transfer function, $H_a(s)$;
3. transform $H_a(s)$ in the desired digital transfer function $G[z]$.

The most widely used transformation is the bilinear transformation that maps the imaginary axis in the s-plane ($j\Omega$) onto the unit circle of the z-plane. However, this design requires some knowledge of analogue low-pass filters and also s and z planes and therefore, we will skip this method and straightaway utilise IIR filter design with MATLAB functions.

There are several classical IIR filters such as Butterworth, Chebyshev types I and II and Elliptic. The MATLAB functions to design these filters are

- `butter` – Butterworth filter with flat passband (no ripples);
- `ellip` – Elliptic filter with ripples (but normally requiring lower order than Butterworth for similar transition band);
- `cheby1` – Chebyshev filter controlling peak-to-peak ripple in the passband;
- `cheby2` – Chebyshev filter controlling the amount of stopband ripple.

First step is to estimate the order of the filter given the specifications using functions: `buttord`, `ellipord`, `cheb1ord`, or `cheb2ord`. Next, obtain the coefficients, *B* and *A* using functions: `butter`, `ellip`, `cheby1`, or `cheby2`. Finally, implement the filter on the data using `filtfilt` function – the reason for choosing this function is described next.

The main problem with the IIR filter is that the phase response is not linear. As an illustration, let us design an elliptic LPF with specifications: *Fs*=200 Hz; passband = 0 to 40 Hz; passband ripple to be less than 3 dB; stopband from 50 Hz to *Fs*/2 and stopband attenuation to be at least 30 dB.

```
[N,Wn]=ellipord(40/100,50/100,3,30); %note use normalised frequency
[B,A]=ellip(N,3,30,40/100);
```

Note that *N* is the order of the filter and is it common to use the same order for both numerator and denominator. The frequency response of the filter can be obtained using function `freqz (B,A)`, which is shown in Figure 4.27. The phase response in the passband[23] is not a straight line (i.e. not linear) and hence, different frequency components will be delayed differently when passed through the filter and cause distortion. To avoid this problem, filtering is performed twice – once forward and once reverse to cancel the phase effects and obtain zero-phase. This can be implemented using *filtfilt* function in MATLAB. In this function, after filtering in the forward direction, the filtered sequence is then reversed and run back through the filter; the final output is the reverse. The result has precisely zero phase distortion and magnitude modified by the square of the filter's magnitude response.

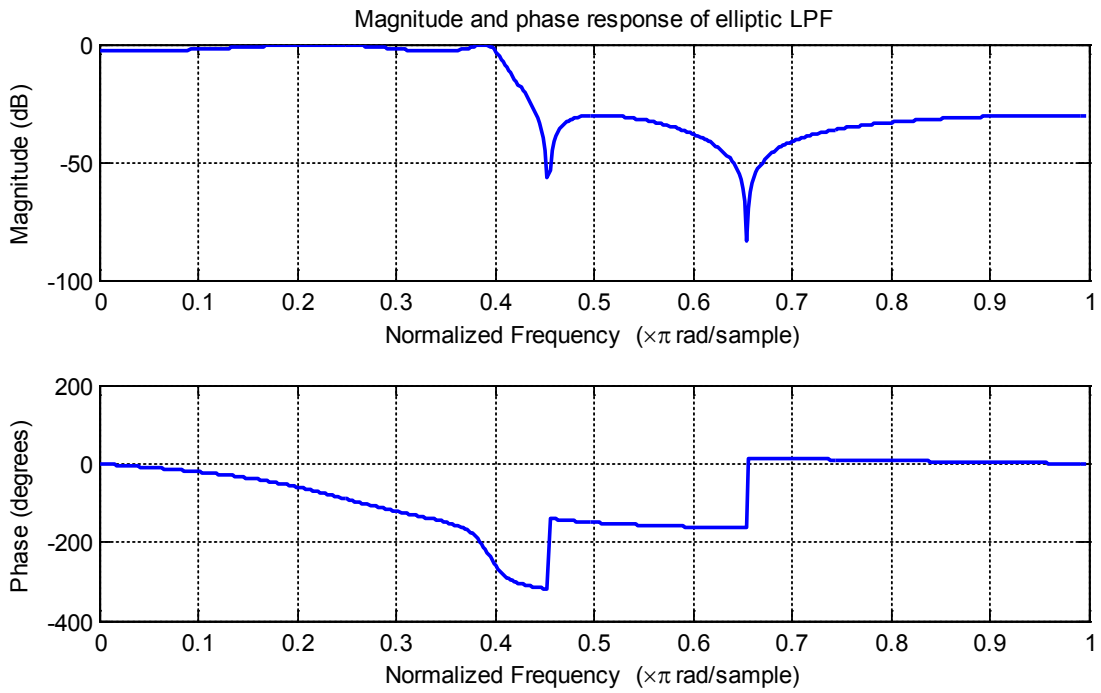


Figure 4.29: Magnitude and phase response of elliptic filter.

Let us use this filter to reduce 50 Hz powerline noise from an ECG signal:

```
ecgf=filtfilt(B,A,ecg);
```

The results are shown in Figure 4.28, where it can clearly be seen the effectiveness of the filter in reducing the noise.

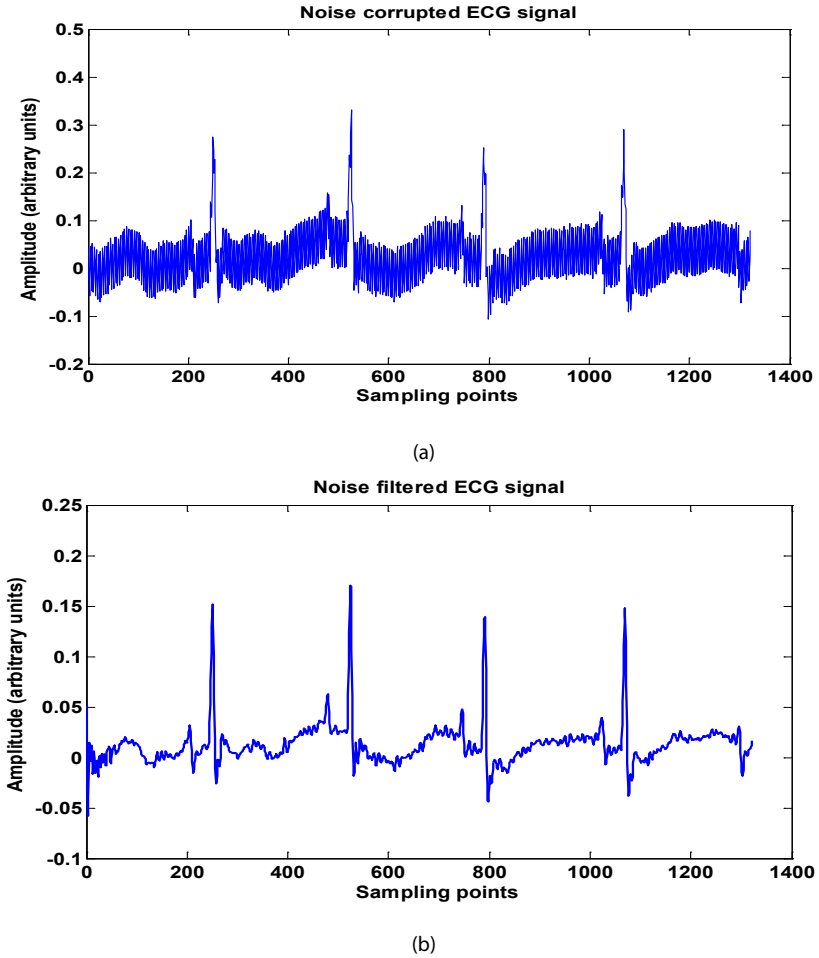


Figure 4.30: ECG signal (a) with noise (b) noise filtered.

4.7 References

[1] G.E.P. Box and G.M. Jenkins, *Time Series Analysis: Forecasting and Control*, Holden Day, San Francisco, 1976.

[2] S.K. Mitra, *Digital Signal Processing: A Practical Approach (3rd edition)*, McGraw-Hill, 2006.

5 Feature extraction

Features are computed values that should be representative of the signal and be reproducible at different times. Other criteria for the features will depend on the application, for example:

- Smaller dimension than the signal;
- High inter-class variance with low intra-class variance;
- Robust/enhanced representation of the signal (i.e. invariant to changes caused by noise, scale factors etc).

5.1 Simple features

Examples of simple features are:

- Mean (but not useful if we set it to zero as a pre-processing step for noise removal);
- Standard deviation;
- Energy (which can be computed by using variance after setting mean to zero) – very often, we measure the energy in different spectral bands and use them as features. For example, in the case of electroencephalogram (EEG), bands like delta, theta, alpha, beta and gamma are normally used. To do this, we can filter the signal in the specific band and then compute the energy. For example, using MATLAB code (with IIR Elliptic filter):

```
%example of values for the EEG bands
%Wp=passband range, Ws=stopband range, Fs=sampling frequency
% for delta band
Wp=[3]/(Fs/2); Ws=[3.5]/(Fs/2); %low pass filter
% for theta band
Wp=[4 7]/(Fs/2); Ws=[3.5 7.5]/(Fs/2); %band pass filter
% for alpha band
Wp=[8 13]/(Fs/2); Ws=[7.5 13.5]/(Fs/2); %band pass filter
% for beta band
Wp=[14 29]/(Fs/2); Ws=[13.5 29.5]/(Fs/2); %band pass filter
% for gamma band
Wp=[30 50]/(Fs/2); Ws=[29.5 50.5]/(Fs/2); %band pass filter

%select appropriate N for each band using ellipord
N=ellipord (Wp, Ws, Rp, Rs); %eg: Rp=0.1 and Rs=30
%then use ellip to obtain filter coefficients
[B,A] = ellip (N, Rp, Rs, Wp);
%then filter the signal X using filtfilt
Xf=filfilt(B,A,X);
%compute energy using var
EXf=var(Xf);
```

5.2 Correlation

Correlation of a test signal with a template signal can be used as a feature. It is a measure of similarity between two signals. In MATLAB, we can use `R=corrcoef(X,Y)` where X is the test signal and Y is the template signal. This is useful when we have a template of signals and need to test the class/category of the test signal. For example, if we have templates for electrocardiogram (ECG) signals from five different heart ailments, then we can use the correlation value from a test ECG signal with each of the templates:

```
%ECGX is the test signal
R1=corrcoef(ECGX,ECGY1); ECGY1 is one of the templates
R2=corrcoef(ECGX,ECGY2);
R3=corrcoef(ECGX,ECGY3);
R4=corrcoef(ECGX,ECGY4);
R5=corrcoef(ECGX,ECGY5);
```

The highest correlation will tell us which activity the test ECG signal is likely to belong. This method is more suitable for ECG signals rather than EEG as EEG signals are more 'random' as compared to ECG signals which generally have more known patterns (such as sinus rhythm, atrial fibrillation etc).

5.3 Autoregressive model

Autoregressive[24] (AR) model is another popular linear feature extraction method for biological signals. A real valued, zero mean, stationary, non-deterministic, autoregressive process of order p is given by

$$x[n] = -\sum_{k=1}^{p} a_k x[n-k] + e[n] \tag{5.1}$$

where p is the model order, $x[n]$ is the data of the signal at sampled point n, a_k are the real valued AR coefficients, and $e[n]$ represents the white noise error term independent of past samples. AR modelling could be seen as a process to obtain an equation that fits the signal (like in curve fitting).

AR modelling tries to model the signal assuming that a data point is closely related to the previous few data points. This is suitable for modelling biosignals. Many different techniques have been proposed to estimate a_k such as Yule-Walker (YW) method. However, YW method is computationally complex and is erroneous for small data segments due to difficulties in properly estimating the autocorrelation function. Hence, recursive algorithms have been proposed to estimate a_k with order p using a_k of previous order p-*1*. Examples of such methods are Burg's and Levinson-Durbin algorithms but the former is more accurate than the latter since it uses more data points simultaneously by minimising not only a forward error but also a backward error [1]. In MATLAB, we can compute the a_k coefficients using `arburg(x,p)` function.

5.2.1 Choosing the autoregressive model order

A model order which is too high will overfit the data and represent too much noise but a model order which is too small will not sufficiently represent the signal. So a compromise has to be made for the model order. There are many methods to compute the model order like Akaike Information Criterion (AIC), Final Prediction Error, Criterion Autoregressive Transfer, Minimum Description Length etc. [1, 2]. AIC is most common and its use will be explained here:

$$AIC(p) = \ln(\sigma_p^2) + 2p/N \tag{5.2}$$

where p is the model order, N is the length of the signal and σ_p^2 is the variance of the error sequence at order p. The first component of the AIC function represents the fitting ability (higher order, better fit) while the second component represents a penalty function with increasing order. In MATLAB, the code `[A,E] = arburg(x,p)` returns the a_k coefficients of signal x in A and error variance in E (using order p).

AIC is computed for order 1 to a certain maximum order (depending on the application, rule of thumb is $p < N/3$, though the selected order is typically lower than $N/3$) and then the order p that minimises the AIC function is chosen to be the optimal order. Also, in general, every additional peak in the spectral plot will require increase of two in the model order [2]. So, model order of six will be required when using AR method for a combination of three sinusoidal components, each with distinct frequency. However, in most cases, it is difficult to know the exact number of spectral peaks and methods like AIC should be used to obtain the AR model order.

For example, using the ECG signal shown in Figure 1.2 and computing AIC (from order 1 to order 20) gives us the following plot:

```
for p=1:20;
[A,E(p)]=arburg(x,p);
AIC(p)=log(E(p))+2*p/(length(x));
end
plot(AIC)
```

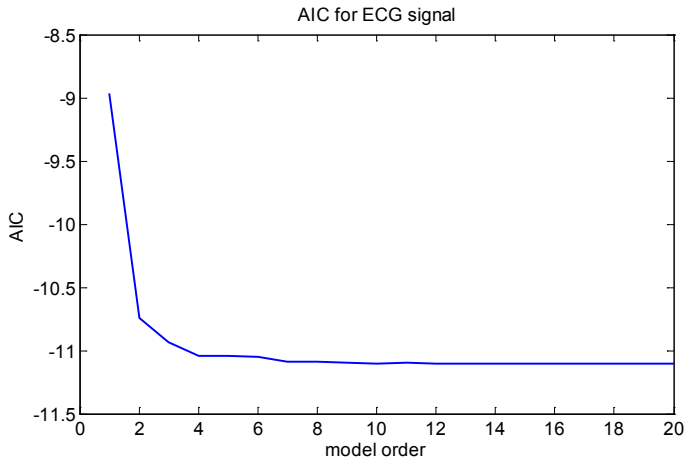


Figure 5.1: AIC for ECG signal shown in Figure 1.2.

It can be seen that AIC values do not change significantly after model order 4, so order 4 can be chosen as the appropriate order.

5.2.2 Autoregressive model to predict signal values

AR model can also be used to predict values of a signal. For example, assume that we have a signal *x:*

```
x=[0.58 0.42 0.52 0.33 0.43 0.26 0.58 0.76 0.53 0.64];
```

the AR coefficients of order 3 are `A = [-0.46 -0.41 -0.10]`. In MATLAB, we will get AR coefficients in the form `[1.0000 -0.4618 -0.4048 -0.1058]`; which are the AR coefficients obtained if we were to rewrite (5.1):

$$\sum_{k=0}^{p} a_k x[n-k] = e[n]. \tag{5.3}$$

Computing $x[10]$ using this 3rd order AR model by ignoring the error term for simplicity, we obtain[25]

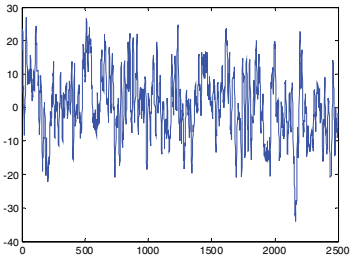
$$x(n) = -\sum_{k=1}^{3} a_k x(n-k), \tag{5.4}$$

x[10] = -(-0.46*x[9]-0.41*x[8]-0.10*x[7])
x[10] = -(-0.46*0.53-0.41*0.76-0.10*0.58)
x[10] = 0.61.

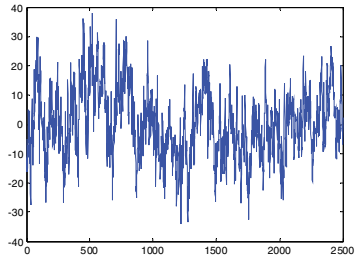
The error in this prediction is 0.64 – 0.61=0.03.

5.2.3 Autoregressive coefficients as features to discriminate mental tasks

As an example to illustrate the usage of AR coefficients as features, consider EEG signals obtained during two different mental activities[26]. In Figure 5.2 (a), two EEG plots for one subject are shown from two mental activities (mathematical activity and imagining an object being rotated). The abscissa is sampling points while the ordinate is amplitude (arbitrary units). Figure 5.2 (b) shows another two EEG plots (one from each mental activity taken at another time from the same subject). From these EEG plots, it is difficult to differentiate the maths and object rotation activities. But using the 6^{th} order AR coefficients, the math and object rotation activities can be differentiated. This is though exact values are not produced by the AR model for the same mental activity, the values are sufficiently close within a mental activity and sufficiently different across activities (especially the first few AR coefficients).

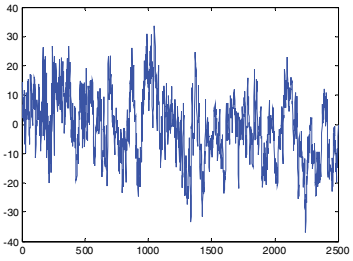


Maths activity (session 1)
A = [-1.5661 0.7114 -0.1843 -0.0583
0.2179 -0.0769]

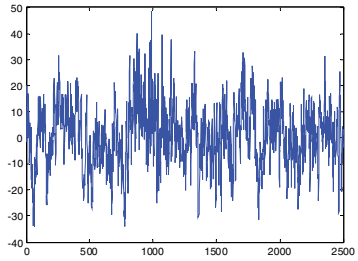


Object rotation activity (session 1)
A = [-0.6128 -0.1677 -0.1159 -0.0733
0.0179 0.0299]

(a)



Maths activity (session 2)
A = [-1.6091 0.603 -0.1931 -0.0432
0.2112 -0.0553]



Object rotation activity (session 2)
A = [-0.5647 -0.2189 -0.0826 -0.0756
0.0083 0.0215]

(b)

Figure 5.2: Example illustrating the use of AR coefficients as features.

5.3 Spectral features – Power spectral density

Power spectral density (PSD) values are also used as features for biological signals. Different methods can be used to obtain PSD, for example periodogram, Welch and AR (more on this later).

For example, consider PSD plots obtained from EEG during a math activity and while at rest (shown in Figure 5.3). It can be seen that there is more spectral power in the lower frequency during maths activity as compared to resting (specifically shown by the encircled red peak around 12 Hz, note that the frequency scale is normalised). The other peak at 60 Hz that is observable in both plots is due to powerline interference[27].

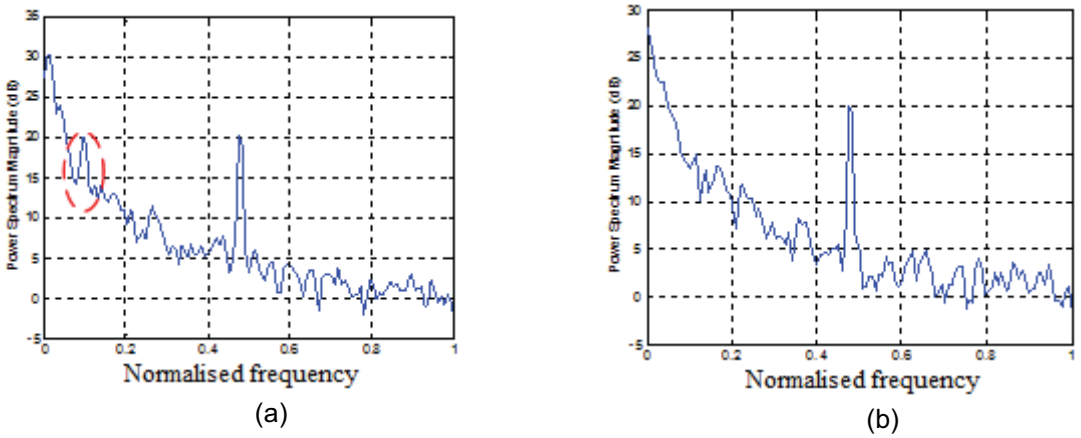


Figure 5.3: Example illustrating PSD as features (a) math activity (b) rest.

5.4 Power spectral density derived features

While we can use PSD as features, we can also use values derived from the PSD such as asymmetry ratio, peak values and spectral coherence as features.

5.4.1 Asymmetry ratio PSD

Asymmetry ratio PSD can be computed using

$$ASPSD = \left[\frac{P_1 - P_2}{P_1 + P_2}\right], \tag{5.5}$$

where P_1 is the PSD in one channel and P_2 is the PSD in another channel but in the opposite hemisphere. This feature is useful for EEG analysis when the mental activities that are studied exhibit inter-hemispheric difference. For example, it is known that math activity exhibit a higher power spectrum in the left hemisphere whereas visual tasks do so in the right hemisphere. So, AS_{PSD} using EEG from channels O1 and O2 (as shown in Figure 5.4) will be higher for maths activity as compared to visual activity if P_1 and P_2 are channels from left and right hemispheres, respectively.

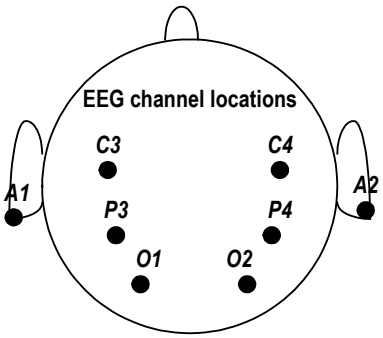


Figure 5.4: EEG channel location used to record the mental task EEG data [3].

Consider the PSD plots for math and object rotation (visual) activities shown in Figure 5.5. AS_{PSD} using channels O1 and O2 are computed. Figure 5.6 (a) shows AS_{PSD} for maths activity and Figure 5.6 (b) shows AS_{PSD} for visual activity. If the AS_{PSD} features are not sufficiently distinct, we can further compute the total sum of AS_{PSD} as a feature. The result is as expected: total AS_{PSD} of 0.3262 is higher for math activity as compared to total AS_{PSD} of -0.1433 for object rotation activity.

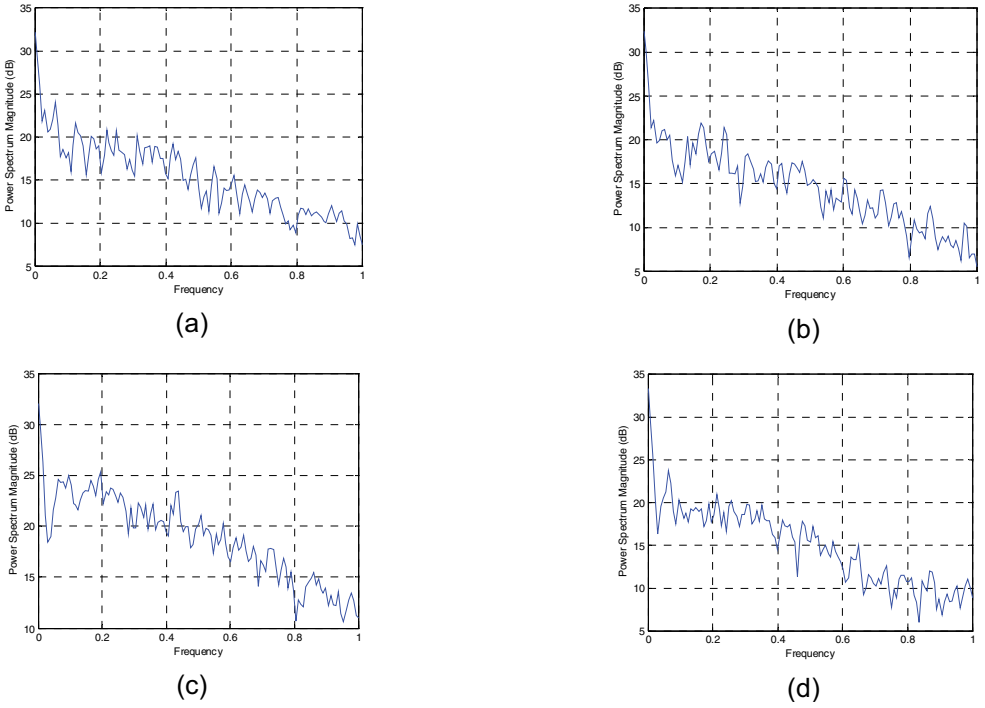


Figure 5.5: PSD for mental activities (a) math, channel O1 (b) math, channel O2 (c) object rotation, channel O1 (d) object rotation, channel O2.

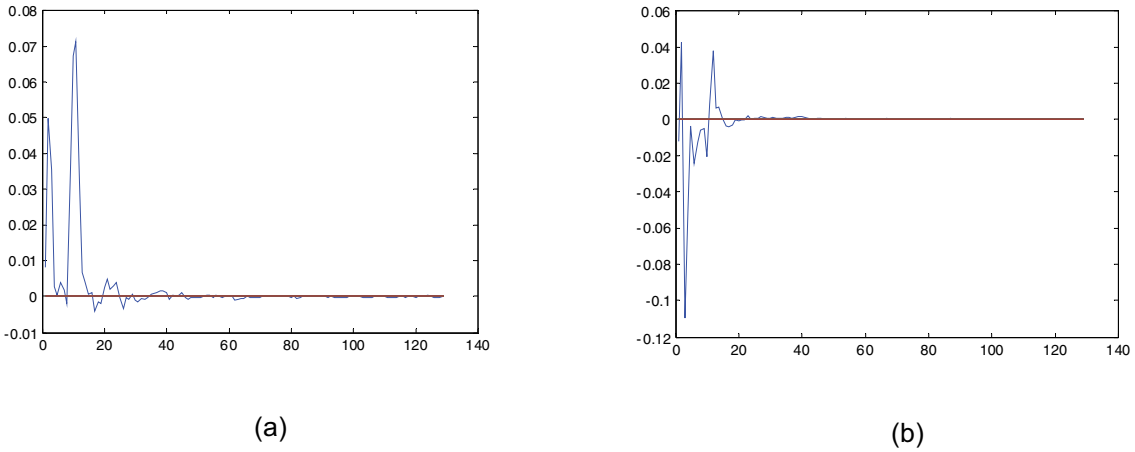


Figure 5.6: AS_{PSD} for mental activities (a) math (b) rotation.

5.4.2 Spectral correlation/coherence

This is similar to the correlation that we discussed earlier in Section 5.2 except with the difference that the spectral correlation is computed in frequency domain instead of time. So the spectral correlation is computed using PSD values. With MATLAB, we can still use `R=corrcoef(X,Y)` where X and Y are the PSD values from two signals.

For example, assume that EEG spectral correlation of object rotation and math activities from two sessions S1 and S2 are computed:

```
%PSDrS1 - PSD from rotation activity EEG from session S1
%PSDmS1 - PSD from math activity EEG from session S1
SR = corr(PSDrS1,PSDrS2); answer - 0.9151
SM = corr(PSDmS1,PSDmS2); answer - 0.8921
SMRS1=corr(PSDmS1,PSDrS1); answer - 0.7582
SMRS2=corr(PSDmS2,PSDrS2); answer - 0.6410
```

It should be obvious that spectral correlation between similar mental tasks should be higher than across mental tasks and the obtained answers indicate this situation.

5.4.3 Spectral peaks

Peak values in the PSD plots are also useful features. For example, consider the hypothetical PSD plot shown in Figure 5.7.

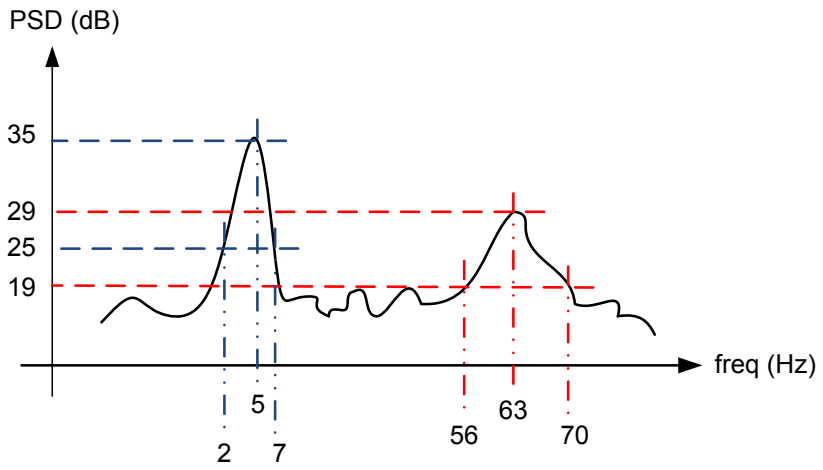


Figure 5.7: Hypothetical PSD plot.

From the PSD plot, we can identify[28] two peaks in the PSD plot and then compute the following parameters as features

- Frequency – the actual frequency value of the peak (i.e. 5 Hz and 63 Hz in the plot)
- Amplitude – the amplitude (i.e. PSD) value of the peak (i.e. 35 dB and 29 dB in the plot)
- Width – the width of the peak computed using some threshold, eg. 10 dB from maximum (draw a line at 25 dB and 19 dB, intersection gives 7-2=5 Hz and 70-56=14 Hz as widths in the plot)
- Area – the area under the peak can also be used as features, this is normally computed as the sum of PSD values from say, ± 3 Hz from the peak amplitude (compute the sum of PSD values from 2-8 Hz and 60-66 Hz in the plot)

5.5 Power spectral density computation using AR features

We have looked at PSD estimation using periodogram and Welch methods in Chapter 3, it is also possible to use AR method to obtain PSD. After obtaining the a_k coefficients and σ_p^2 using `arburg` function and with some suitable order p chosen by AIC, we can proceed to compute PSD using:

$$S(f) = \frac{\sigma_p^2 T}{|\sum_{k=0}^{p} a_k e^{-i2\pi fkT}|^2} \tag{5.6}$$

This is known as parametric estimation of PSD as we compute PSD using AR parameters of the signal rather than the data points in the signal. Using MATLAB, we can use the `pburg (x,p)` function.

Spectral estimation with AR method does not suffer from frequency resolution problem (like those based on Fourier transform) and is better when the signals have a very narrowband (i.e. limited to a small frequency range). AR method also requires fewer cycles – sometimes even one cycle is enough to give good estimation of the spectral content. But AR method is more sensitive to noise, so classical methods like periodogram should be used if there is a lot of noise in the signal. Figure 5.8 shows an example of spectral analysis using AR method for the ECG signal shown in Figure 1.2. Comparing this figure with Figure 3.15, it is obvious that AR method gives a finer and smoother spectral plot.

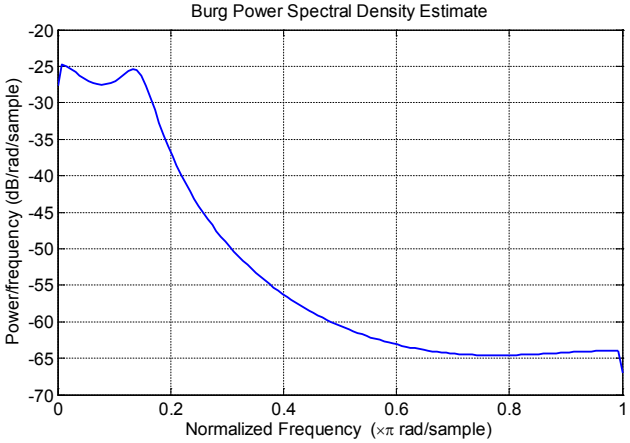


Figure 5.8: PSD of ECG signal using AR method.

5.6 Hjorth descriptors

Hjorth descriptors are useful features to represent biological signals and there are three descriptors, namely activity, mobility and complexity.

Activity is simply the variance of the signal, representing the energy. Mobility is given by $M_x = \frac{\sigma_{x'}}{\sigma_x}$ where $\sigma_{x'}$ is the standard deviation of first derivative of x, which can be obtained using $x[n]'=x[n]-x[n-1]$. Complexity (form factor) is the most useful, $FF = \frac{M_{x'}}{M_x} = \frac{\sigma_{x''}/\sigma_{x'}}{\sigma_{x'}/\sigma_x}$ which gives a computational value for the shape of the signal. The second derivative can be computed as $x[n]'=x[n]-2x[n-1]+x[n-2]$.

As an example, for the math EEG signal shown in Figure 5.2 (a), the Hjorth descriptors can be computed using:

```
Activity=var(EEG)
for i=2:length(EEG),
EEG1(i)=EEG(i)-EEG(i-1);
end
Mobility=std(EEG1)/std(EEG)
for i=3:length(EEG),
EEG2(i)=EEG(i)-2*EEG(i-1)+EEG(i-2);
end
FF=(std(EEG2)/std(EEG1))/(std(EEG1)/std(EEG))
```

These descriptors have been applied successfully for the detection of ectopic beats and will be discussed in the final chapter. However, it should be noted that these descriptors are very sensitive to the presence of noise.

5.7 Time domain features

Features can also be extracted from the time series of the signal. For example, there are frequently distinguishable peaks in evoked potential signals such as P1, P2, P3, N1, N2, and N3 (shown in Figure 5.9). The peaks are normally identified using the latency, which is the time delay (in ms) from the stimulus onset. For example, it is known that the third positive component, P3 obtained during the oddball paradigm has a latency of about 300 ms. Each peak will also have an amplitude and is normally measured relative to pre-stimulus baseline (say about 100 ms or 200 ms prior to stimulus onset). It is also possible to compute peak-peak amplitudes, such as N1-P2 amplitude.

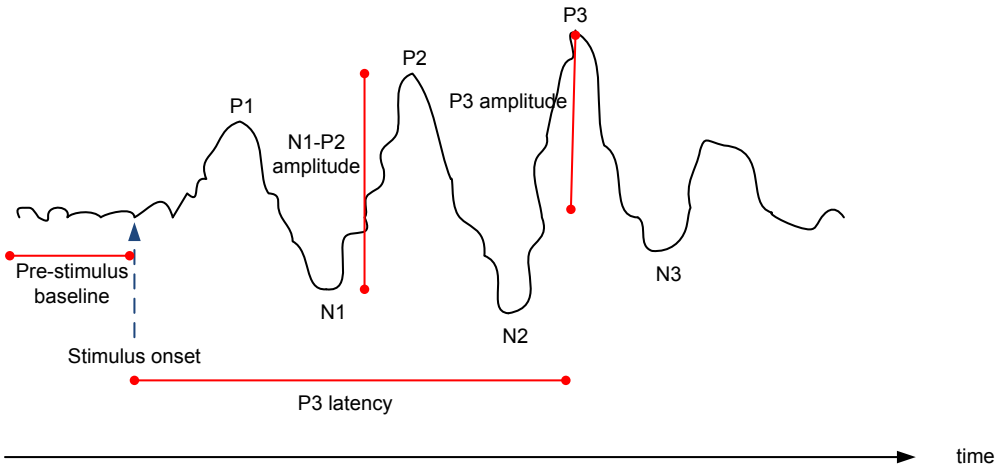


Figure 5.9: Simple features from time-series of the signal.

Another feature that can be extracted directly from time-series is zero crossing. It measures the number of times that the signal crosses the zero amplitude line. But simply finding the occurrences of zero, say using `find(x==0)` may not work as the values may not be zero exactly. The other way is to compute the number of occurrences when the data points in the signal changes from +ve to –ve and –ve to +ve. The mean should be removed from the signal before computing zero crossing.

5.8 Joint time-frequency features

It is possible to compute features that are both in time and frequency domains, such methods are like

- Linear, non-parametric methods: Short-time Fourier transform and Wavelet transform;
- Quadratic, non-parametric methods (with improved time-frequency resolution): Wigner-Ville distribution and related techniques.

Joint time-frequency [4] analysis is out of scope for discussion in this book.

Summarising, several feature extraction methods have been discussed in this chapter as examples but these are not at all exhaustive and we'll look at a few more in the final chapter.

5.9 References

[1] R. Shiavi, *Introduction to Applied Statistical Signal Analysis (2nd edition)*, Academic Press, 1999.

[2] L. Sornmo and P. Laguna, *Bioelectrical Signal Processing in Cardiac and Neurological Applications,* Elsevier, 2005.

[3] Mental Task EEG data: Available: http://www.cs.colostate.edu/eeg/eegsoftware.html#keirndata.

[4] S. Qian and D. Chen, *Joint Time-Frequency Analysis*, Prentice Hall, 1996.

6 Classification methodologies

In this chapter, we'll discuss classification methodologies. Specifically, we'll look into the following:

- *k*-Nearest Neighbour (*k*-NN) and multilayer-perceptron (MLP) neural network classifiers;
- Classifier performance measures;
- Cross validation approaches;
- Statistical measure to compare the performances of two methods.

6.1 What is classification?

Classification is a process where the unknown labels of test data/patterns are predicted and can be divided into supervised and unsupervised. We'll concentrate on supervised classification methodologies here.

Normally, for classification studies, we have three types of dataset: training, validation and testing. The class/category labels of the training and validation datasets are known, while for testing data, the class/category labels are unknown. The training process is where a classification model/decision rule is formed using the training dataset while validation dataset is used to decide the best parameters for the model. The validation dataset is also used to obtain a performance measure on the goodness of the model by classifying i.e. predicting the validation dataset class labels and comparing with the actual labels. If the performance measure is satisfactory, the obtained classification model could then be used to predict the unknown classes of the test datasets and this is known as testing process.

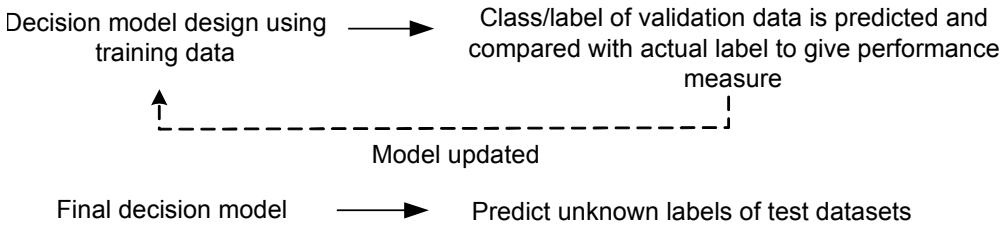


Figure 6.1: Classification procedure (training and testing).

6.2 Nearest Neighbour classifier

Nearest Neighbour is one of the simplest classifiers. A pattern in the test data is classified by calculating the distance to all the patterns in the training data; the class of the training pattern that gives the shortest distance determines the class of the test pattern. Consider a plot of several training patterns (each with two features) from two classes (A and B) and a test pattern as shown in Figure 6.2. The nearest neighbour to x is training pattern A, hence x is predicted as belonging to class A.

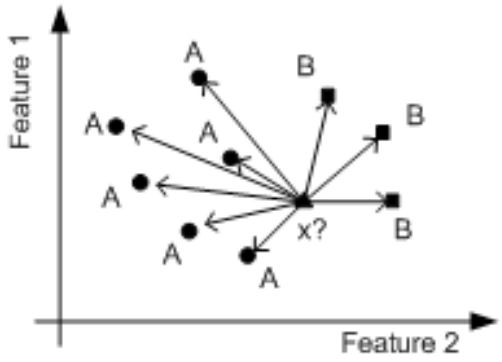


Figure 6.2: Plot of several training patterns from two classes, A and B and one test pattern, x.

Euclidean and Manhattan (city block) are commonly used distance measures. Euclidean distance between two points can be computed as

$$d_E(p,q) = \sqrt{\sum_{i=1}^{R}(p_i - q_i)^2}\ , \tag{6.1}$$

where R is the number of features. For example, in the Euclidean plane with two features, distance from points (2, 5) to (6, 8) is sqrt $((2\text{-}6)^2 + (5\text{-}8)^2) = 5$.

Manhattan distance can be computed using

$$d_M(p,q) = \sqrt{\sum_{i=1}^{R}|\ p_i - q_i\ |}\ . \tag{6.2}$$

For example, the distance between the two points now would be $|2\text{-}6| + |5\text{-}8| = 7$.

The k-NN classifier extends this idea by taking the k number of nearest patterns and using a majority rule to decide the label of the test class. It is common to select a small value for k[29] and odd to break ties. In the example above, N=9 and using a k value of 3, the nearest neighbours are patterns from class A, A, and B, so x is classified (i.e. predicted) as belonging to class A. Larger k (i.e. larger training dataset) help reduce the effects of noisy patterns within the training dataset but at a higher computational cost.

Another illustrative k-NN example is shown in Figure 6.3. With k=1, pattern X will be classified as from class B, while with k=3 and k=5, the predicted class will be B and A, respectively. It should be noted that k=3 should be chosen as N=7 here: sqrt (7) = 2.65 ≈ 3.

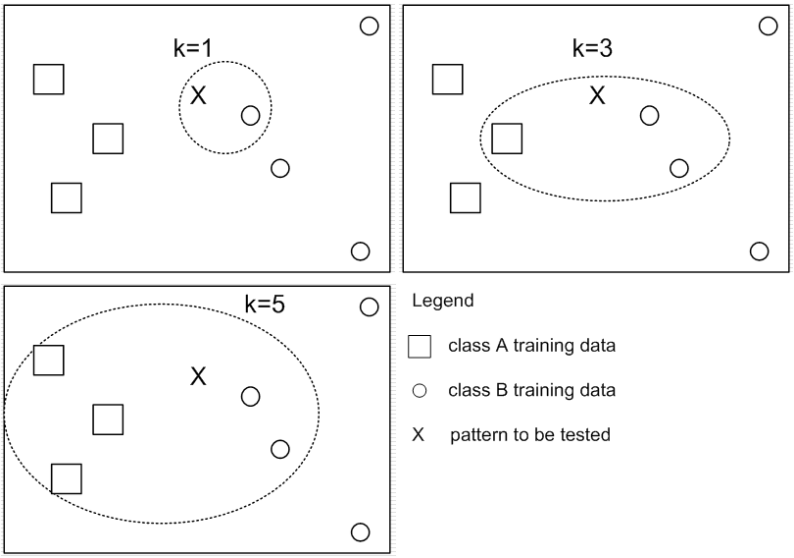


Figure 6.3: *k*-NN example with varying *k* values.

6.2.1 k-NN algorithm

The steps in the *k*-NN algorithm are

1. Determine parameter *k*, i.e. the number of nearest neighbours to use;
2. Calculate the distance between the test pattern and all the training patterns;
3. Sort the distance and determine the *k* nearest neighbours;
4. Gather the category/class labels of the nearest neighbours;
5. Use simple majority of nearest neighbours' classes to determine the class of the test pattern.

As a numerical example, consider the following problem. Assume that there are two autoregressive (AR) features obtained from electroencephalogram (EEG) signals recorded from two alcoholic and two non-alcoholic subjects[30] as shown in Table 6.1. Now, using *k*-NN, let us classify whether a subject Y with test data, Y(3,7), i.e. X1=3 and X2=7 is alcoholic or non-alcoholic.

	AR features		
Pattern no.	X1	X2	Subject
T1	7	7	Alcoholic
T2	7	4	Alcoholic
T3	3	4	Non-alcoholic
T4	1	4	Non-alcoholic

Table 6.1: Training data for *k*-NN numerical example

As a first step, determine *k*. Here, *k* could be 1 or 3 as sqrt(4)=2 and we often select *k* to be odd. Next, compute, say Euclidean distance of Y(3,7) to each training pattern:

	Training patterns	Euclidean distance
Test pattern Y(3,7)	T1(7,7)	$\sqrt{(7-3)^2+(7-7)^2}=4$
	T2(7,4)	$\sqrt{(7-3)^2+(4-7)^2}=5$
	T3(3,4)	$\sqrt{(3-3)^2+(4-7)^2}=3$
	T4(1,4)	$\sqrt{(1-3)^2+(4-7)^2}=3.6$

With *k*=1, the closest training pattern to Y is T3. As T3 is from a non-alcoholic subject, subject Y is classified as non-alcoholic with *k*=1. Similarly with *k*=3, the closest training patterns to Y are T3, T4 and T1. The majority is non-alcoholic class, so subject Y is classified as non-alcoholic also with *k*=3.

6.2.2 Advantages and disadvantages of *k*-NN classifier

The advantages of *k*-NN classifier are

- Robust to outliers (noisy training patterns) if *k* is sufficiently high. For example, test pattern X is correctly classified (with *k*=3) as A in Figure 6.4 even though there is a noisy pattern from class B, which X is closest.

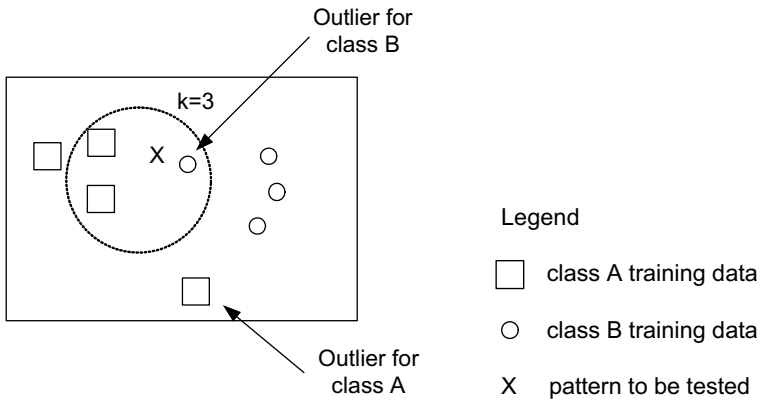


Figure 6.4: *k*-NN is robust to outliers.

- Generally, it gives good classification performance as the decision boundaries can be non-linear.
- The algorithm is easy to understand and implement.

The disadvantages of *k*-NN are

- Parameter estimation: There is the need to determine value of parameter *k* (number of nearest neighbours) through trial and error. Though the rule of thumb sqrt(*N*) could be used; when *N* is relatively large, this rule is difficult to be applied. Also, different distant measures can give different performances and again a trial and error method is normally used to decide the best distance criterion.
- Not robust to irrelevant inputs as irrelevant features have the same influence on the classification as do good features. A solution to this is to multiply the distances from the features such that irrelevant and redundant features have a lower weight (this is known as weighted *k*-NN). For example, if there are two features and feature 1 is more discriminatory, then weight for feature 1 could be say 0.7 and weight for feature 2 will be 0.3. Using this method, Manhattan distance of two points, A(1,3) and X(6,5) would be $0.7*|6-1| + 0.3*|5-3| = 4.1$ rather than standard Manhattan distance of $|6-1| + |5-3| = 7$.
- Computation cost is quite high as there is the need to compute distances of each test pattern to all training patterns.
- The training model is not easy to interpret.

6.2.3 MATLAB program for *k*-NN

In this section, we'll look at a MATLAB program for implementing *k*-NN. We'll use the alcoholic and non-alcoholic example discussed earlier. The training dataset array will be denoted as 'train' and test dataset as 'test'. Also, let us assume alcoholic class is represented by '0' and non-alcoholic class by '1'.

```
%initial data
train=[7 7; 7 4; 3 4; 1 4];
test=[3 7];
target=[0 0 1 1]; %alcoholic represented by '0'; non-alcoholic by
'1'

%determine distances, say Euclidean
for i=1:length(train)
     Edist(i)=sqrt(sum((train(i,:)-test).*(train(i,:)-test)));
end

%join target information with the distance
Et=[Edist; target]

%transpose
Et=Et';

%sort the distances
sortEt=sortrows(Et)

%determine majority of k neighbours, say k=3
k=3;
predicted_class=mode(sortEt(1:k,2))

%display output
if (predicted_class==0)
     disp('Subject is alcoholic')
else
     disp('Subject is non-alcoholic')
end
```

The output display from the program is shown in Figure 6.5.

```
Et =

    4.0000    5.0000    3.0000    3.6056
         0         0    1.0000    1.0000


sortEt =

    3.0000    1.0000
    3.6056    1.0000
    4.0000         0
    5.0000         0


predicted_class =

     1

Subject is non-alcoholic
```

Figure 6.5: *k*-NN program output display.

6.2.4 Reducing *k*-NN training dataset size

To use *k*-NN, we need to compute distances of the test pattern from every training pattern and this is computationally very expensive. So, there are techniques to reduce the training dataset size where the general idea is to remove as many training patterns as possible with minimal impact on the output of the classifier (i.e. performance).

The simplest method is to use the decision surface for the classifier, where the patterns closest to the boundary are the ones which determine the shape of the boundary and these patterns cannot be removed. Patterns that are far from the boundary, and surrounded by same class patterns are redundant, and can be removed without affecting the output of the classifier. Figure 6.6 illustrates this method with an example. But this method can only be applied for smaller number of patterns with simple decision boundaries. So, we'll look at two better methods[31]: Condensed Nearest Neighbour and Edited Nearest Neighbour.

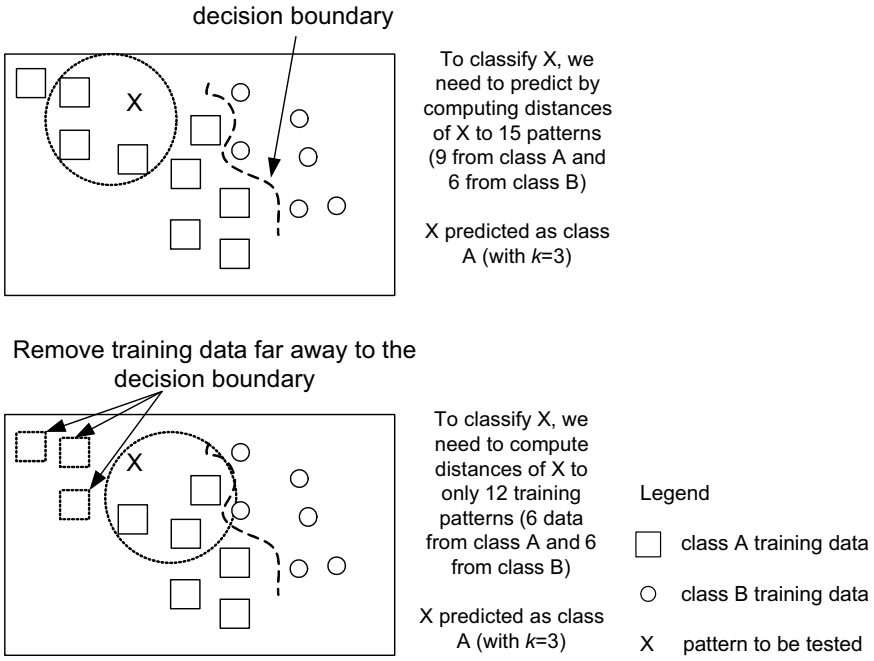


Figure 6.6: Simple method to reduce *k*-NN training pattern size.

6.2.5 Condensed Nearest Neighbour

The algorithm is as follows [1]:

Using training patterns, create two bags: store and grab. Put all the training patterns in grab bag:

1. Take one training pattern from each class from the grab bag and put into the store bag
2. Take all the available training patterns in grab bag and classify using the patterns in the store bag as training dataset

3a. If any data is incorrectly classified, then it is added to the store
3b. If it is correctly classified, leave it in grab bag
4. Repeat steps 2 and 3a/3b until the grab bag is empty or there are no more transfers to the store (i.e. no change in the grab bag)
5. The store then forms the condensed training set

As a result of this condensation, the classification performance can drop. Figure 6.7 shows an example where we have five training patterns from two classes (A and B): T1, T2 are from class A and T3, T4, T5 are from class B.

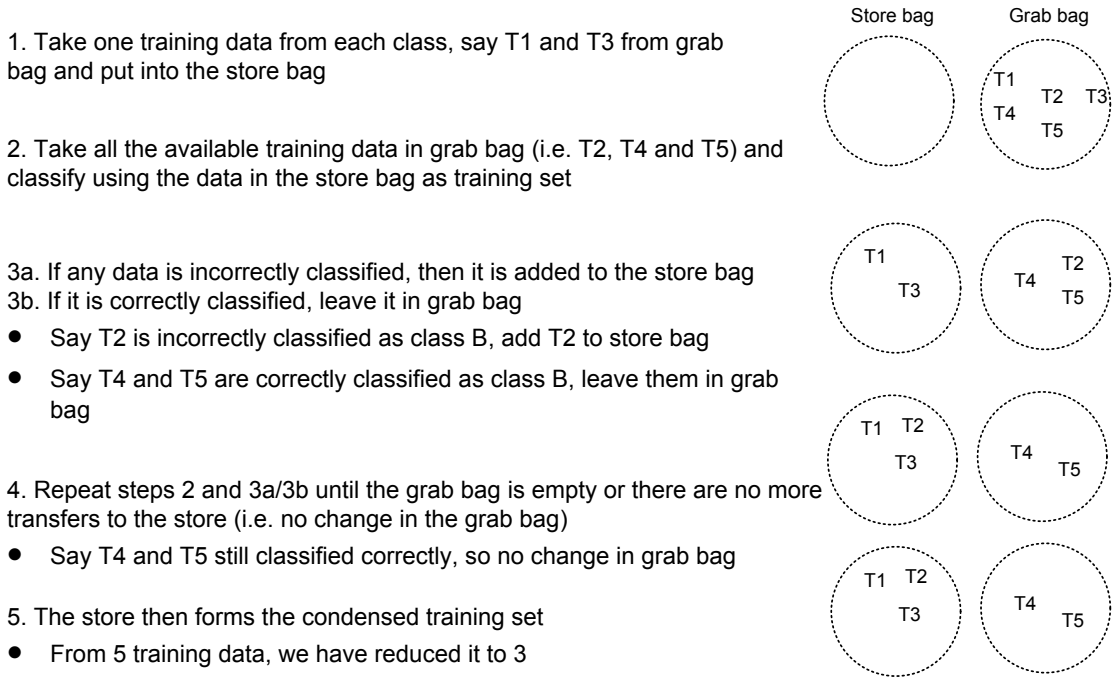


Figure 6.7: Condensed Nearest Neighbour example.

6.2.6 Edited Nearest Neighbour

This is a simpler process where a pattern is taken in the training set and removed if it doesn't agree with its *k* nearest neighbours. This process is repeated for the entire training data. Several passes can be made to remove a further percentage of data. This produces smooth boundaries and also removes outlier/noisy data. Figure 6.8 shows an example of this method.

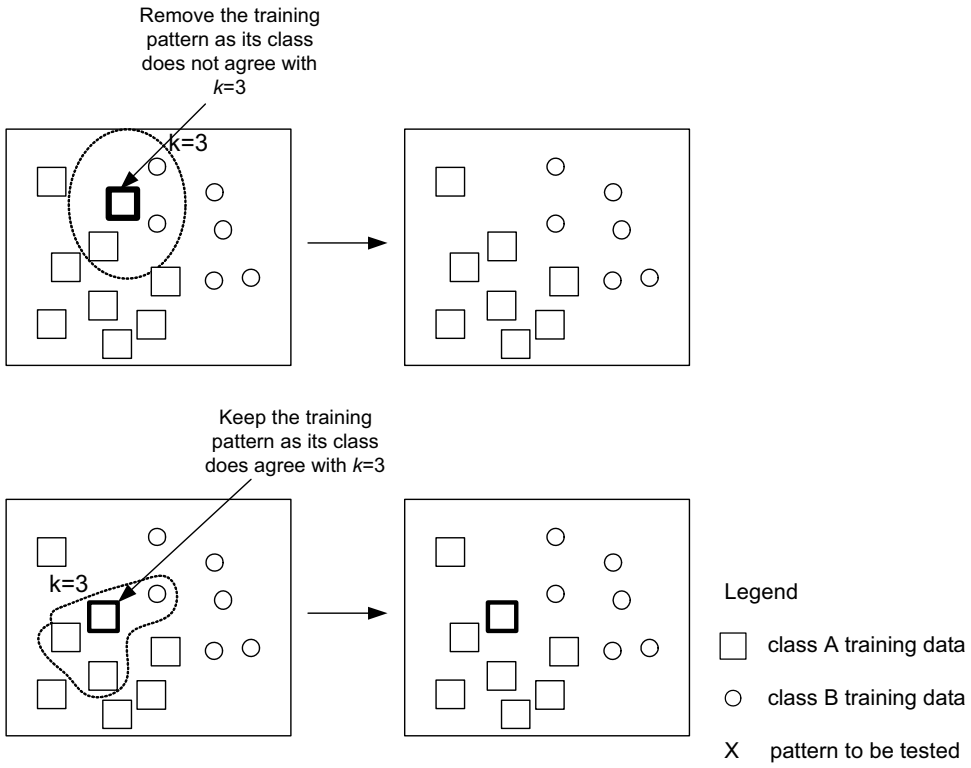


Figure 6.8: Edited Nearest Neighbour example.

6.3 Artificial neuron

In the first chapter, we looked at the biological neuron and here, a description of artificial neuron will be given before discussing the neural network classifier. An artificial neuron shown in Figure 6.9 is a very simple abstract of a biological neuron shown in Figure 1.8. It has a basic computational element, called a node or unit. Each input x has an associated weight w which can be modified and the inputs x corresponds to signals from other neuron's axons, while x_0 is a *special* bias input with weight w_0.

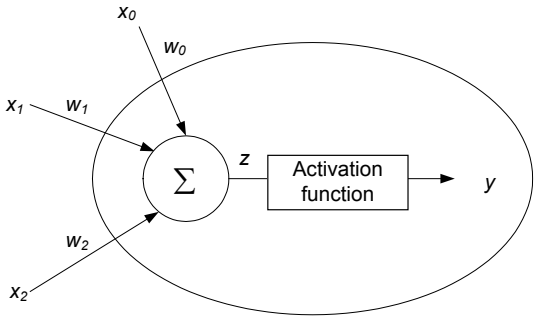


Figure 6.9: A simple artificial neuron model.

Weights, w correspond to synaptic modulation (i.e. something like strength/amount of neurotransmitters) and the summation corresponds to cell body:

$$z = x_0 w_0 + x_1 w_1 + x_2 w_2 + \ldots + x_m w_m = \sum_j x_j w_j \ . \tag{6.3}$$

The activation function corresponds to axon hillock, it computes function f of the weighted sum of its inputs. Hence, neuron output $y=f(z)$ corresponds to an axon signal. The simplest f is the linear function, i.e. output $y=z$. There are other functions, such as the binary, sigmoid and tanh. An activation function should be continuous, differentiable and bounded. Sigmoid is the most common activation function and is defined by

$$y = \frac{1}{1+e^{-z}} \ . \tag{6.4}$$

Consider an example to compute the neuron output. Obtain the output, y given the following: the activation function is sigmoid; the inputs are x_1=0.5 and x_2=0.15; the weights are w_1=0.02, w_2=0.7; the bias (x_0) is 1.0 and the bias weight (w_0) is 0.9.

The answer: net output (before activation function) = 0.5*0.02+0.15*0.7+0.9*1.0 =1.015 while neuron output (after activation function) = $1/(1+e^{-1.015})$ = 0.7340.

6.4 Multilayer-Perceptron neural network

The Multilayer-Perceptron (MLP) is one of the most widely used neural network architecture in classification problems. Input quantities are processed through successive layers of neurons, where there are three[32] layers (though it is possible to have two or four layers). An input layer (which receives inputs) normally has neurons (i.e. units) equal to the number of input features while the output layer (which generate outputs) will have neurons equal to the number of classes in the problem[33]. A hidden layer (the layer in between) can have any number of neurons and this value is normally decided through trial and error.

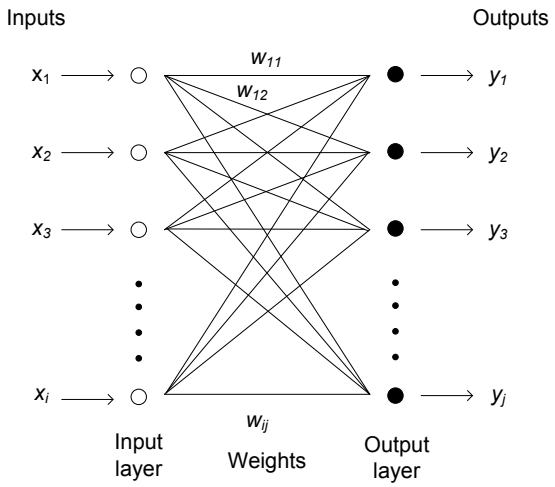


Figure 6.10: MLP architecture with three layers.

Training is conducted to obtain the desired outputs given the inputs – this is supervised learning. Training changes the weights to minimise the output error using gradient descent methods and one such popular training method is the backpropagation (BP) algorithm [2]. There are many variants of BP algorithm to speed up training and improve accuracy but we will utilise the functions available in MATLAB to implement the MLP-BP classifier.

6.5 MLP-BP classifier architecture

MLP is able to carry out complex classification provided that there are sufficient number of layers and sufficient number of units in each layer. The use of MLP-BP as classifier will be described using an example to detect ectopic beats (described in Section 1.1). In this example, assume that we have ten features representing an ECG signal and we wish to classify the features into three classes: normal (Nml), premature ventricular contraction (PVC) or supraventricular contraction (SVC).

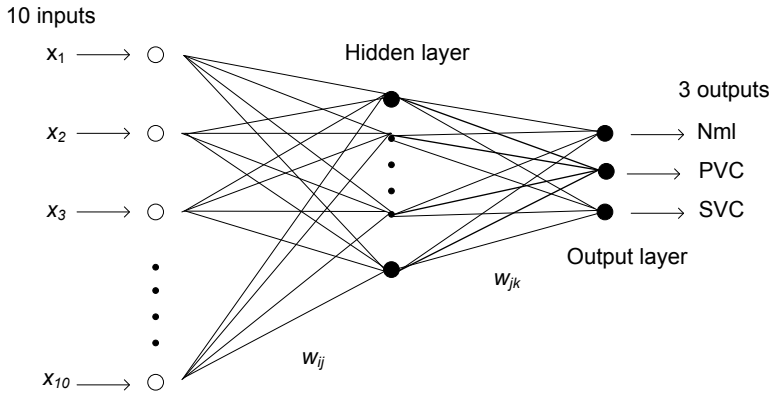


Figure 6.11: MLP architecture for classification example.

The input units will be ten and the output units will be three, while the hidden units can vary from ten[34] up to any number, though we normally limit to a few multiplicative factor of the minimum number. For example, in this case, the hidden units could vary from 10 to 50 and to reduce computational time, we could use only a few values in this range such as 10, 20, 30, 40 and 50.

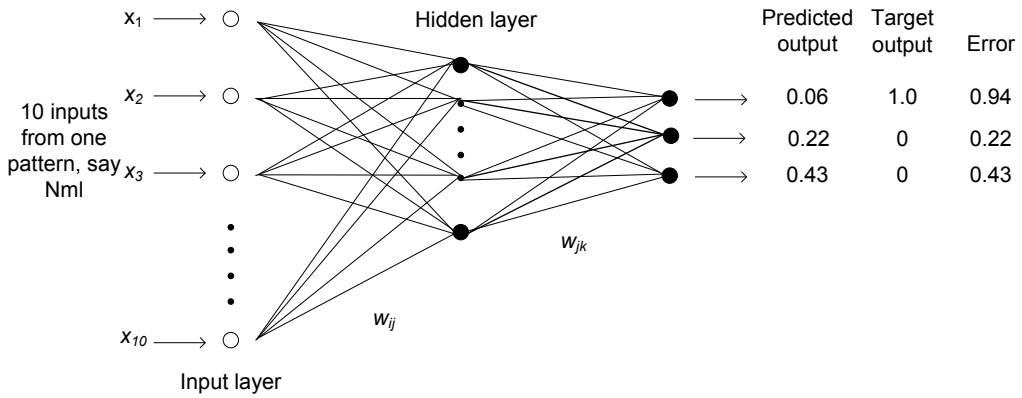
6.5.1 Training MLP-BP classifier

Assume we have 150 patterns, i.e. 50 patterns from each class. Dividing this data into two sets equally (later, we will see other ways of dividing the dataset), we will have 75 patterns for training and 75 patterns for validation.

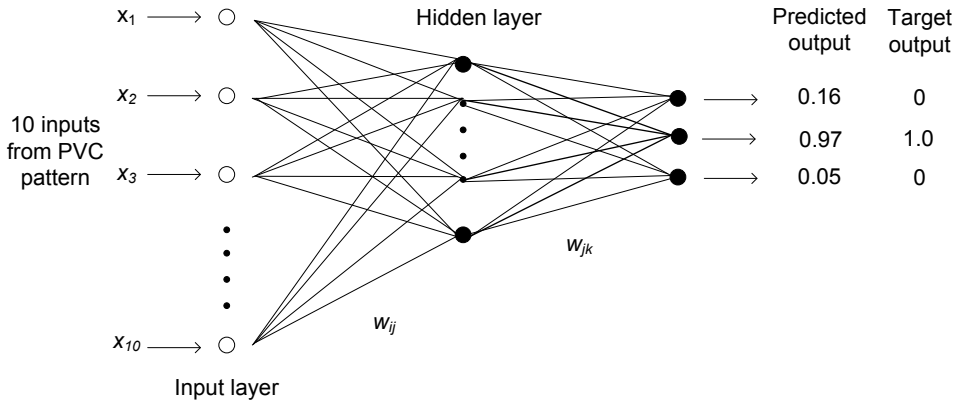
The training patterns will be used to train the neural network. The steps are:

1. All the weights are randomly initialised.
2. Iteration=1: Training is done by feeding the input features from one pattern at a time and the outputs are either set to [1 0 0] or [0 1 0] or [0 0 1] depending on whether the input pattern is from Nml, PVC or SVC.
3. A forward propagation is done for each unit in the hidden layer and similarly for the output layer using the inputs $x_1 \ldots x_{10}$ and equations such as (6.3) and (6.4). Since the weights are randomly initialised, the three predicted outputs, y_1, y_2, y_3 obtained from this forward pass will be different from the target (desired) outputs. Mean square error (MSE) of target outputs against predicted outputs is computed. This forward pass is done for all the 75 training patterns.
4. The MSE of all the 75 patterns are used by the BP algorithm to update/change the weights[35]. This is known as the backward pass and the weight change will be in the direction of reducing the MSE, i.e. to obtain predicted outputs closer to the target outputs.
5. End of iteration 1, repeat the iterations (forward and backward passes) until the desired error is reached or maximum iteration is reached, then training is completed.

This procedure is illustrated graphically in Figure 6.12 (a) where after the first iteration, there is a large error between the target and predicted output but after completing the training, say after 1000 iterations (Figure 6.12 (b)), the neural network is able to give predicted outputs close to the target.



(a)



(b)

Figure 6.12: MLP-BP neural network (a) 1st iteration (b) after completing training.

6.5.2 Testing the performance of MLP-BP classifier

To test the performance of the classifier, patterns from validation dataset are normally used. Here, the weights do not change and there is only one single forward pass. Each pattern input is fed into the classifier and the resulting outputs are computed. The maximal output is assumed to be the predicted class. An example is shown in Figure 6.13. Assuming that we trained the neural network with the first output to represent Nml, the second output to represent PVC and the third output to represent SVC (see step 2 of the previous section), then this pattern is predicted as belonging to SVC. If it is the actual class, then it is correct classification; else it is incorrect classification. This is repeated for the rest of the patterns. The classification percentage is normally computed as (no. of correct patterns/total validation patterns)*100% but we'll also look at a few other performance measures later. If the performance measure is satisfactory then we could utilise the trained MLP-BP neural network to predict classes of unknown patterns in the testing dataset. Otherwise, the MLP-BP is trained again until satisfactory performance is achieved before attempting prediction of classes from test patterns.

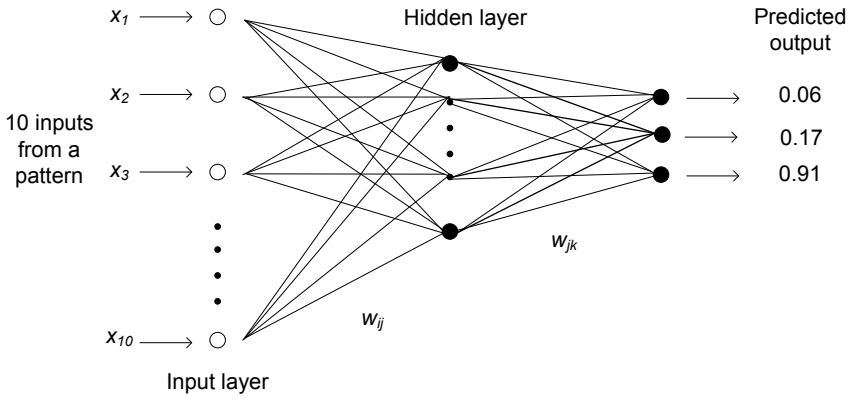


Figure 6.13: Testing the trained MLP-BP neural network.

6.5.3 MLP-BP classifier implementation using MATLAB

The following MATLAB program shows the implementation of the MLP-BP classifier. Initially the targets are specified and the architecture of the neural network is created. Next, the training parameters are specified and the training started. Finally prediction using either testing or validation data is done.

```
%define target values, i.e. for 75 patterns with 3 classes
%alternate class feeding
t=[1 0 0; 0 1 0; 0 0 1];
t=repmat(t,25,1);

%design network architecture using newff
%X are the training patterns, minmax to define range of X
%logsig (sigmoid) activation functions
%traingd is the standard BP training method
%HU is the hidden layer unit size, HU=10; no. of outputs=3
%number of units in input layer is automatically obtained
net=newff(minmax(X), [HU,3], {'logsig', 'logsig'},'traingd');

%training parameters
net.trainParam.epochs=1000; %maximum iteration
net.trainParam.show=100; %show error values every 100 iterations
net.trainParam.goal=1e-4; %error limit
[net, tr]=train(net,X,t); %start training

%validation/testing using sim function
%The array A will contain output values, find maximum for each
%pattern and determine the predicted/classified output, then the
%classification percentage (for validation data)
A=sim(net,XX); %XX are the validation or testing patterns
```

6.5.4 MLP-BP problems

MLP-BP has been successfully used in many applications in different disciplines such as control systems, data fusion and financial forecasting in addition to biological signal classification. It should be noted that while we only discussed the usage of MLP-BP for classification, it can also be used as function approximator.

However, it is known to have several problems:

- Local minimum: The error surface can have more than one local minimum. The system may converge to a local minimum and get stuck there; BP algorithm is guaranteed to converge to some local minimum but not necessarily to the global minimum (shown in Figure 6.14)

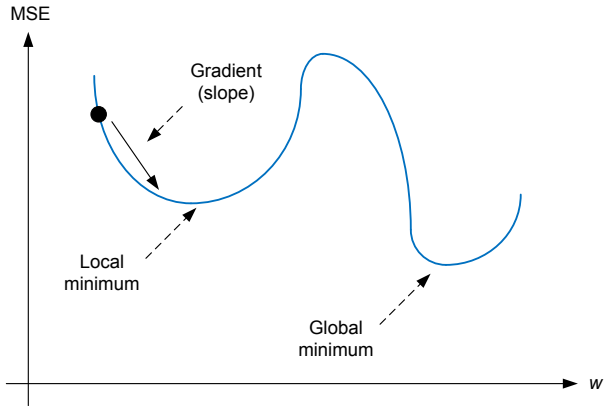


Figure 6.14: MSE as a function of weight, *w* showing local and global minima.

- Starting search space: Different initialisations of the weights can produce different results.
- Overfitting: The neural network is trained '*too much*', so that the error is low during training but gives high error during validation.

To solve these problems, the following approaches could be used:

- To avoid getting stuck in local minimum: Use a momentum term added to weight change (Δw) to avoid getting stuck in a local minimum. In MATLAB, this can be done using `traingdx` instead of `traingd`.
- To solve the problem of initial search space: Repeat a number of classification experiments with different initialisations and choose the one that gives the best validation performance.
- To prevent overfitting: use a validation dataset to test the behaviour of the error after say every 100 iterations and use *early stopping* if necessary, i.e. stop training when the error of the validation dataset starts to increase, provided that the training error is acceptable. Figure 6.15 illustrates this early stopping procedure where the threshold (green line) is the required training error. The training should be stopped early since the required training error has been met and the validation error is beginning to increase.

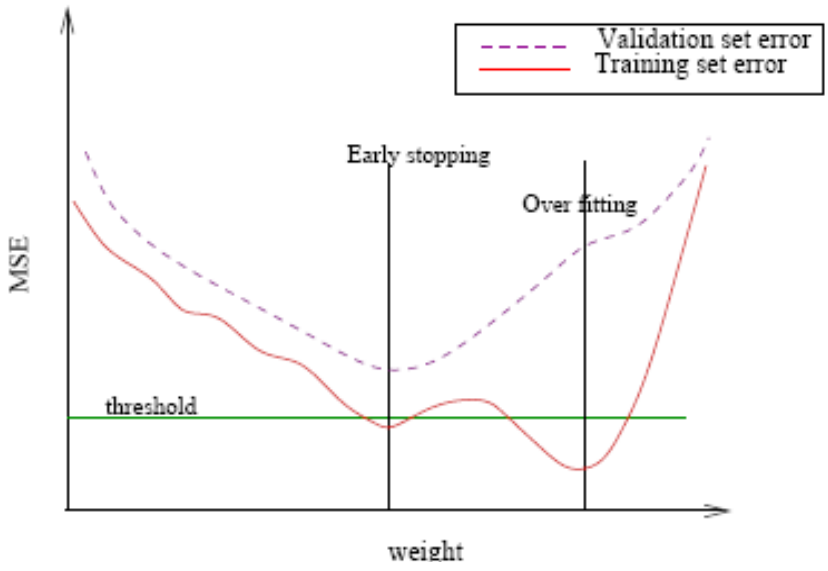


Figure 6.15: Early stopping.

6.6 Performance measures

There are many approaches to compute the performance measure of the classifier. Some of the commonly used ones are given here:

- Root Mean Square (RMS) of the error: this can be calculated for either the training or validations datasets

$$RMS = \sqrt{\frac{1}{N}\sum_{t}^{N}(t-y)^2}\,, \tag{6.5}$$

where t is the target/desired output and y is the actual predicted output. N is the number of patterns (instances) in the data set.

- Recognition rate: is the ratio of the number of patterns that were correctly classified to the total number of patterns being validated.
- Scoring method: patterns that are correctly classified are given high/positive scores, patterns that are not classified are given no score and patterns that are incorrectly classified are given a low/negative score. A total score is estimated by adding all scores of all patterns and finding the ratio to the total number of patterns.
- Confusion matrix: Table 6. 2 shows the confusion matrix of a two class (A and B) classifier where
  - p is the number of correct predictions of patterns from class A
  - q is the number of incorrect predictions of patterns from class A (i.e. predicted as from class B)
  - r is the number of incorrect predictions of patterns from class B (i.e. predicted as from class A)
  - s is the number of correct predictions of patterns from class B

So, we have

$$recognition \quad rate = \frac{p+s}{p+q+r+s}. \tag{6.6}$$

Such information will be useful since not only we'll be able to see how well the classifier performed overall but also how well were the predictions for each class.

		Classifier prediction	
		A	B
Actual Class	A	p	q
	B	r	s

Table 6.2: Confusion matrix for a two-class classifier

6.7 Cross validation

The holdout is the standard approach in classification where the dataset is separated into training and validation sets with no overlap. The common training/validation dataset ratio is 50:50. After using the training patterns to construct the model, classification/prediction is done using the validation patterns. The advantage of this approach is that it takes least time to compute but is disadvantageous as classification results can have a high variance, depending on which patterns end up in the training set and which end up in the validation set, and thus the classification may be significantly different depending on how the division is made. Of course, if training and validation dataset size is large, then the classification variance will be lower. But very often, when we deal with smaller dataset sizes, we can use N fold cross validation method.

In this method, the combined training and validation dataset is divided into N non-overlapping subsets, and the holdout method is repeated N times where each time, one of the N subsets will be used as the validation set and the other N-1 subsets are put together to form a training set. Then the average classification error across all N trials is computed.

The advantage of this is that the variance of the resulting classification is reduced as N is increased, i.e. the classification results becomes more reliable but the disadvantage is obvious; the training and classification algorithm has to be rerun N times, which means it takes up to N times as much computation as compared to the case without cross fold validation.

6.7.1 Equal class weight

If the dataset has equal number of patterns from each class, each subset can be made to have equal number of patterns from each class. For example, see Figure 6.16 which shows 10 fold cross validation using equal class weight for the dataset that we used in Section 6.4.2. The total number of patterns here is 150, i.e. 75 from training and 75 from validation datasets.

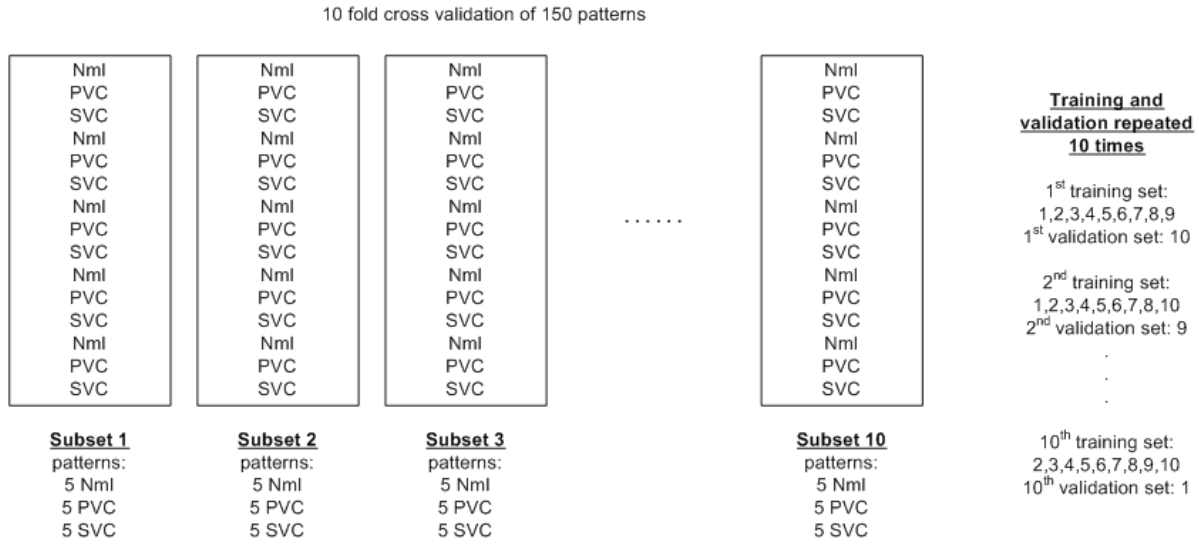


Figure 6.16: Ten fold cross validation.

A variant of this method exists to reduce computational time: the dataset is divided into N subsets and the holdout method is repeated several times but using different $N/2$ training subsets and $N/2$ validation subsets without overlap. Then the average classification error is computed. This gives us the freedom to choose the number of times to repeat the classification based on our computational resource. The 50:50 ratio of training/testing data as above could be varied depending on the application; 90:10 and 80:20 are the other common ratios.

6.7.2 Leave one out

Leave one out is N-fold cross validation at its extreme with N equal to L, the total number of patterns. Here, the holdout method is repeated L times, with L-1 subsets for training and the one single remaining pattern is used for validation. Then the average classification error across all L validations is computed. This is a good method but computationally very expensive. For the ectopic beat problem discussed earlier, this will be mean that we train with 149 patterns and classify the remaining one pattern; repeat for 150 times with different train and validation patterns.

If equal class weight (with class size C) is used, then the holdout method is repeated L/C times using L-C training data and C validation data where each subset has equal number of patterns from each class. In the discussed example, this will be to train with 147 patterns and classify the remaining three patterns from each class, repeat for 50 times with different train and validation patterns each time.

6.8 Statistical measure to compare two methods

Very often, when we have utilised more than one method of pre-processing, feature extraction or classification, we arrive at a problem on deciding which method to choose for the final test stage. The classification performance is generally used to decide (assuming that other measures such as time and complexity are secondary) but sometimes, this will not be clear if one set of methods give better results occasionally but not all the time.

Consider Figure 6.17 where it is obvious that the performance of method A surpasses that of B since none of the classification performances using method B are superior to method A. However, considering Figure 6.18, the situation is not straightforward on which is the better method. In such cases, we can employ statistical methods such as t-test or rank based methods to help us in deciding the better method.

Ten fold cross validation performance	Classification (%)	
	Method A	Method B
Set 1	96.5	92.3
Set 2	95.4	89.5
Set 3	98.5	91.6
Set 4	96.2	90.5
Set 5	94.3	88.8
Set 6	93.5	87.9
Set 7	92.8	92.3
Set 8	94.3	89.8
Set 9	96.8	90.1
Set 10	93.4	91.2

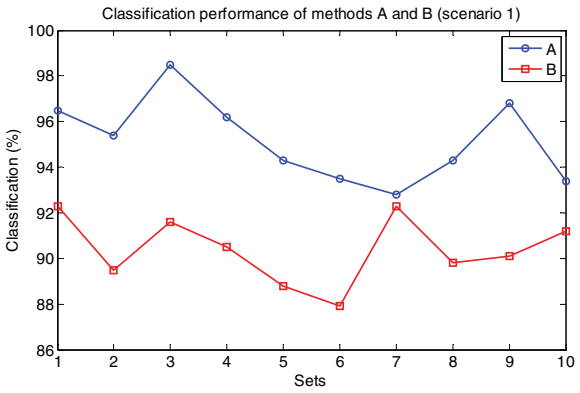


Figure 6.17: Classification performance of methods A and B (scenario 1).

Ten fold cross validation performance	Classification (%)	
	Method A	Method B
Set 1	89.5	97.3
Set 2	90.2	94.5
Set 3	93.2	96.6
Set 4	97.8	95.5
Set 5	92.6	93.8
Set 6	90.1	92.9
Set 7	89.8	97.3
Set 8	97.5	94.8
Set 9	98.8	95.1
Set 10	96.7	96.2

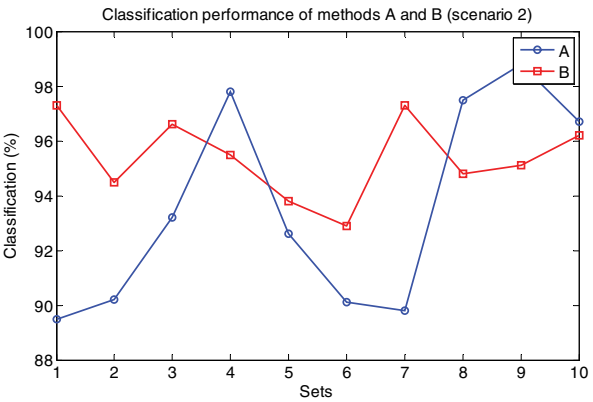


Figure 6.18: Classification performance of methods A and B (scenario 2).

6.8.1 Hypothesis testing

Hypothesis testing refers to statistical methods that can be used to arrive at a yes or no decision regarding a particular hypothesis. There are two competing hypotheses: null (H_0) and alternative (H_1). The null hypothesis is the original assertion. For example, when considering the results such as in Figures 6.17 and 6.18, the null hypothesis will be that the means are equal while the alternative hypothesis could be one of the following:

- Means are not equal (H_1: A≠B);
- Mean of A is greater than B (H_1: A>B);
- Mean of B is greater than A (H_1: A<B).

A significance level is related to the degree of certainty that is required in order to reject the null hypothesis in favour of the alternative hypothesis. By taking a small sample, we cannot be certain about the population. So we can decide to reject the null hypothesis if the probability of observing the sampled result is less than the significance level. For a typical significance level of 10%, the notation is $\alpha = 0.1$. For this significance level, the probability of incorrectly rejecting the null hypothesis when it is actually true is 10%. A lower value can be chosen to be more certain (i.e. to reduce the error of rejecting H_0).

A test statistic will be computed from the data and its value can be used to decide to reject H_0 or not to reject H_0. But as there are many different possible test statistics that can be used, frequently we convert test statistic value to *p*-value, which is independent of test statistic approach. The *p*-value is the probability of observing the given data under the assumption that the null hypothesis is true. Small *p*-values suggest that the null hypothesis is unlikely to be true. The smaller it is, the more convincing is the rejection of the H_0.

For the results shown in Figure 6.18, let us employ t-test[36] for comparing means from the two classification methods (as shown in Figure 6.18). Our hypotheses will be H_0: A=B and H_1: A<B.

Using MATLAB,

```
[H, p] = ttest2(A,B,0.1,'left');
```

gives H=1 and p=0.089. Since the *p*-value<α, then H_0 is rejected. This means that at significance level 0.1, we can conclude that method A is less superior to method B. The MATLAB generated H=1 also denotes the acceptance of H_1. So using the t-test approach, we can conclude that method B is superior to method A.

6.9 References

[1] E. Alpaydin, *Introduction to Machine Learning,* MIT Press, 2004.

[2] S. Haykin, *Neural Networks: A Comprehensive Foundation (Second Edition)*, Prentice Hall, 1999.

7 Applications

In this chapter, we will discuss a few examples of biological signal analysis applications:

- Ectopic beat detection using electrocardiogram (ECG) and blood pressure (BP) signals
- Electroencephalogram (EEG) based brain-computer interface designs
- Short-term visual memory impairment in alcohol abusers using visual evoked potential (VEP) signals
- Identification of heart sounds using phonocardiogram (PCG)

7.1 Ectopic beat detection using ECG and BP signals

Premature heart beats (or ectopic beats) are those that originate from other than sino-atrial locations. Premature ventricular contraction (PVC) originates from the ventricular while the premature supraventricular contraction (PSC) originates either in the atrial or junctional (nodal) but grouped together because of the similar electrocardiogram (ECG) waveform. The occurrences of ectopic heart beats are not life threatening but signify problems with the heart and some forms of ectopics can indicate a predisposition to life-threatening arrhythmias [1]. For example, frequent occurrences of PVC (> 6 beats per minute) may lead to ventricular fibrillation (VF).

As such, early detection of such beats is important but detection of these beats is time-consuming because of the occasional nature of occurrence. Hence, intelligent and automated computer based detection would be an advantage especially in long-term patient monitoring.

In the study conducted in [2], 13 features were obtained from ECG and BP signals and used to classify the beat as from either PVC, PSC or normal category. The block diagram shown in Figure 7.1 shows the steps utilised in this study.

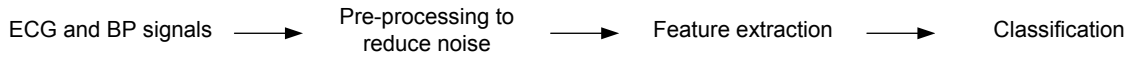


Figure 7.1: Block diagram showing the steps in classification of the ectopic beats.

ECG from lead I and BP signals were obtained from Massachusetts General Hospital/Marquette Foundation (MGH/MF) database [3]. A total of 3000 beats were used in the study with 1000 beats from each class (N, PVC and PSC). The dataset was divided randomly in two (with equal beats from each class), one as training set to train the classifier and another as validation set to give a performance measure.

An infinite impulse response (IIR) Butterworth bandpass filter was used to filter both the ECG and BP signals to reduce baseline (low frequency noise) and 60 Hz powerline interference (high frequency noise). Rather than implementing a bandpass filter directly, a combination of low-pass filter (LPF) and high-pass filter (HPF) was used here since the high-pass filter would have a cut-off close to zero and MATLAB can cause errors while performing the computation. An example of such a pre-processing step is shown in Figure 1.1 and Figure 1.2. For the ECG signal, the passband was from 1–35 Hz, while for the BP, the passband was from 0.5 to 15 Hz. These ranges were deemed suitable to keep most of the required signal components.

Before the features could be extracted from the ECG signals, the R peak has to be detected. The authors in [2] have utilised a modified Pan-Thompkins algorithm where a bandpass filter (7–14 Hz) was used to capture most of the QRS component. This was followed by squaring (to convert the values to all positive), window integration (similar to moving average, to smooth the signals) and thresholding (to decide the location of the R peak). Normally, the threshold value is difficult to be decided prior and requires adaptive methods to obtain a compromise between proper R peak detection and avoiding false positives caused by noise. Figure 7.2 shows an example of this procedure to detect the location of the R peak. It should be noted that in each of the steps, any delay should be taken into account. For example, in [2], a moving window integration of 54 samples (150 ms with 360 Hz sampling frequency) was applied and hence the resulting signal has to be shifted by 53 samples to obtain the R peak accurately. After thresholding, the maximum peak in each block was assumed to be the R peak.

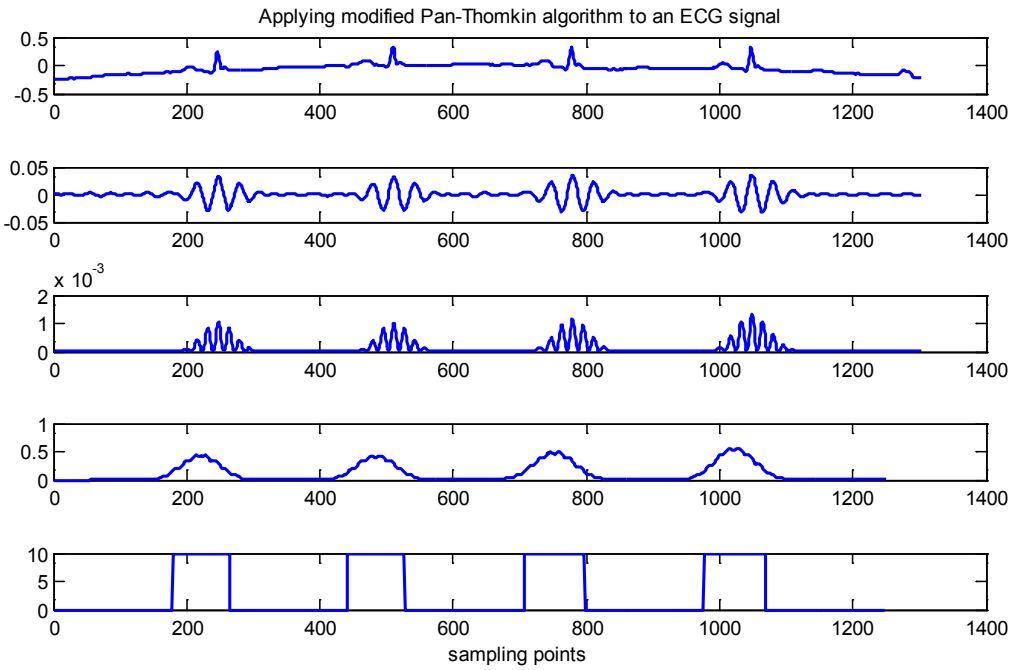


Figure 7.2: Modified Pan-Thompkins algorithm [2] for an ECG signal, from top to bottom: ECG signal, bandpass filtered, squared, moving window integrated and thresholded.

Once the R peak has been detected, five features were computed from the ECG. These were the R peak amplitude, R-R interval before the beat (pre R-R interval), R-R interval after the beat (post R-R interval), Hjorth's mobility and Hjorth's complexity. All these five features were normalised with values from a normal beat from the same subject. The computation of mobility and complexity were done for the QRS complex, where the Q peak was assumed to be within 50 ms before the R peak and S peak to be within 100 ms after R peak, i.e. the features were computed using 150 ms of ECG data.

For the BP signals, 8 features were derived. First, the systolic pressure (SP, peak) and diastolic pressure (DP, trough) values were detected by simple peak and trough detection methods. Then, mean arterial pressure (MAP) and pulse pressure (PP) were computed using

$$MAP = DP + \frac{1}{3}SP - DP \text{ and } PP = SP - DP\,. \tag{7.1}$$

The SP and DP for the beats prior and after the beat being studied were computed (pre SP, post SP, pre DP and post DP). DP changes (delta DP) and SP changes (delta SP) were computed to give a total of 8 features. Again these features were normalised with values from a normal beat.

A multilayer perceptron (MLP) neural network (using the backpropagation (BP) algorithm) was trained with the 1500 training patterns. The inputs were 13 and the outputs fixed to 3 (as there were 3 classes). The hidden units were varied from 10 to 50 in steps of 10 and an average classification accuracy of 96.11% was obtained.

Overall, the study showed that with proper signal analysis procedures, automated detection of ectopic beats is possible and could lead to saving of lives.

7.2 EEG based brain-computer interface design

BCI technologies are useful for the severely disabled to communicate with their external surroundings as they circumvent the usage of peripheral nerves and limbs, thereby creating a direct link between thoughts and computers (machines). There are many approaches for BCI technologies but the most common is the non-invasive EEG due to its low cost, portability and ease of use [4]. In methods using non-invasive EEG, several divisions could be formed, namely those based on motor imagery, steady state evoked potential, transient evoked potential, slow cortical potential and mental activity (task). In the following sub-section, we will discuss two of these methods.

Figure 7.3 shows the components that are found in a general BCI system. The EEG data is collected, normally from multiple channels (some even up to 128), during an external stimulus, which could be visual, auditory, somatosensory or internally induced thoughts such motor imagery[37] and mental activities (such as mental arithmetic). The data is normally pre-processed to reduce noise and then features are extracted to be classified by a classifier in the decision process, which basically recognises the specific category of the mental activity. A device command then translates this (sometimes using a translation algorithm) into meaningful control, for example moving the cursor on a computer screen, spell out letters/words or control a wheelchair. In some systems, a feedback is given to the subject to allow adjustments for improving the performance.

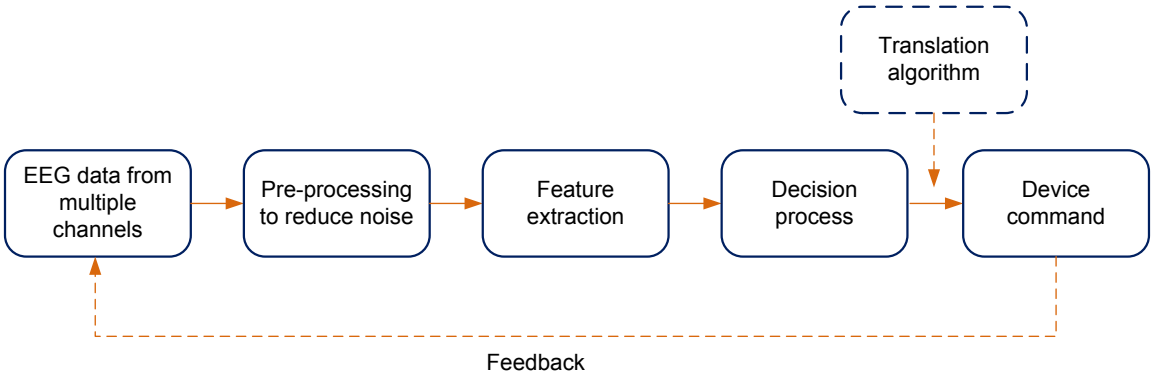


Figure 7.3: General components of a BCI system.

7.2.1 BCI based on transient visual evoked potential

The EEG that is in response to an external stimulus is known as evoked potential (sometimes known as event related potential). Transient evoked potential is the response to single stimulus while if the stimulus occurs at a repetitive rate that is at least 6 Hz, the responses become compounded and result in steady state evoked potential that has the same frequency as of the stimulus rate. If the stimulus modality is visual (for example a picture), the resulting response is known as visual evoked potential (VEP).

Figure 7.4 shows an example of the BCI speller paradigm based on VEP. In the original design [5], a subject will concentrate on a specific character (known as the target) and each row and each column will flash randomly. The number of flashes will be 12 (as there are 6 rows and 6 columns) and when the specific row or column that contains the target letter flashes, the VEP response will have P300 component (see Section 1.2.3) that occurs around 300–600 ms after stimulus onset. It should be noted that when the rows or columns that do not contain the target letter (i.e. containing the non-target characters), the P300 component will not appear in the VEP response. The common inter-stimulus interval (ISI), i.e. the time between the flashes is around 300–1000 ms though much smaller ISIs have been proposed [6]. The actual flash itself will be 'on' for about 100 ms. The probability of the target row or column is 0.167 as the target row/column will be one out of the six available rows/columns. This stimulus paradigm is known as oddball where the target probability is less than the non-target probability.

The amplitude of P300 component in the VEP is much smaller than the ongoing background EEG and hence the flashing of rows and columns are normally repeated for several trials where each trial encompasses 12 flashes of the rows and columns. With averaging or using other means such as independent component analysis [7], the P300 component in these signals are enhanced. As the P300 component is generally below 8 Hz, a low-pass filter is used in the pre-processing step. Next, the signal is normally downsampled (to reduce the length and computation time in the following steps) and features such as the P300 amplitude is extracted from the signal[38].

A classifier such as neural network is used to detect the occurrence of the P300 component (i.e. classify the VEP as from target or non-target row/column) and synchronising this information with actual row/column flashing, the target character focused by the subject can be predicted.

A	B	C	D	E	F
G	H	I	J	K	L
M	N	O	P	Q	R
S	T	U	V	W	X
Y	Z	1	2	3	4
5	6	7	8	9	space

Figure 7.4: Donchin speller paradigm.

There are many variations of the P300 based VEP BCI such as flashing each letter rather than each row/column and the use of pictures [8] instead of alphanumeric characters.

Consider another stimulus paradigm example using four different colours (green, yellow, blue, magenta) as shown in Figure 7.5 for use in authentication systems. This method could be used to design a passcode that follows a specific sequence, for example 'green, green, blue, yellow, blue, magenta'. Passcode templates for subjects can be matched to the detected colour sequence through VEP responses and hence the system can function as passcode system[39]. The raw VEP responses that have been averaged from 40 trials (from responses for each colour) are shown in Figure 7.6 (a) while the filtered responses (with mean removed) are shown in Figure 7.6 (b). The sampling frequency used here was 256 Hz. It can be seen that the highest peak around sampling points of 77–154 (corresponding to 300–600 ms after stimulus onset) is magenta and this was the colour that the subject focused, so the target colour was correctly predicted using P300 amplitude.

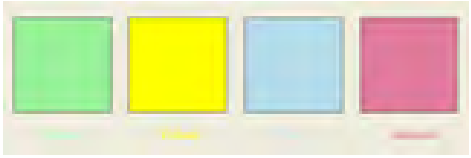


Figure 7.5: Example of colour stimulus presentation paradigm.

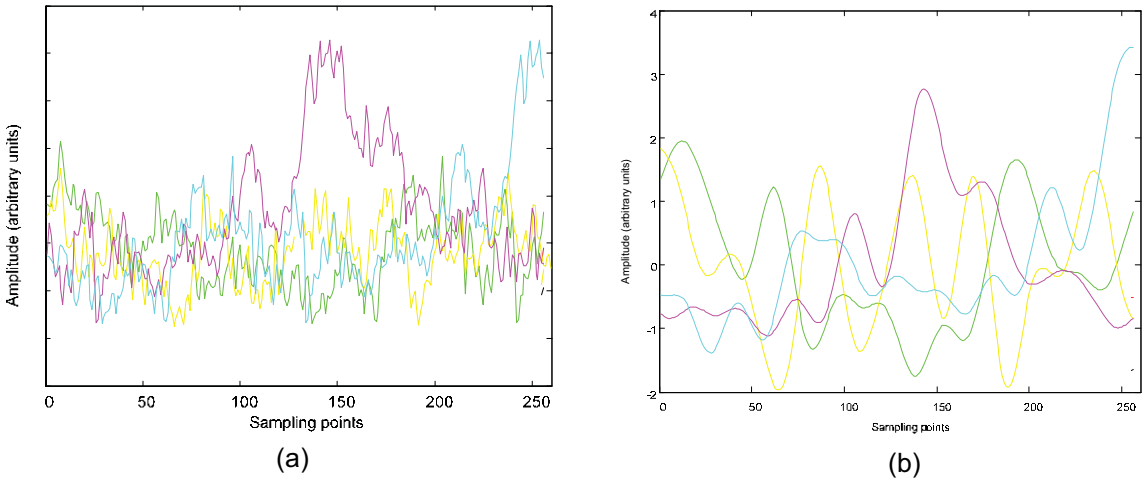


Figure 7.6: VEP responses (a) raw (b) filtered.

7.2.2 BCI based on mental tasks

BCI can also be designed using mental activities/tasks. For example, the study in [9] proposed using the following five different mental activities:

- Relaxed mode as baseline;
- Mental letter composing to a friend
- Non-trivial mental arithmetic
- Visual imagination of an object being rotated
- Mentally visualising numbers being written on a board

These five activities were proposed as they involve one hemisphere of the brain more than the other (except baseline) and hence, the asymmetry ratio discussed in Section 5.5.1 can be used as features.

The study in [10] utilised the EEG data obtained during these five mental activities from channels located at C3, C4, P3, P4, O1, and O2. An IIR elliptic filter was used to filter the EEG in the bands 0–3 Hz (delta), 4–7 Hz (theta), 8–13 Hz (alpha), 14–20 Hz (beta), and 24–37 Hz (gamma). Order five was used as it was sufficient to obtain a minimum of 30-dB attenuation at frequencies ±0.5 Hz beyond the passbands. The features were powers of the specific spectral bands that were computed using the variance of the filtered output. Additionally, asymmetry ratio was computed for each spectral band:

$$Asym(i, j) = (Power(i) - Power(j))/(Power(i) + Power(j)), \tag{7.2}$$

where the indices *i* and *j* are the electrodes from left and right hemispheres, respectively; *power(i)* and *power (j)* are spectral band powers in these electrodes. Asymmetry ratio as used in this study was useful as it gave an indication on the amount of inter-hemispheric difference of EEG activity.

The total number of features was 75 (i.e. five bands x six band powers x nine asymmetry ratios). An Elman neural network [11] trained by the resilient backpropagation [12] algorithm (ENN-RBP) was used to recognise the mental task activity. The ENN-RBP training and validation were repeated for hidden unit (HU) sizes of 20, 40, 60, 80, and 100 using a ten-fold cross validation where the 50:50 ratio was used. The results indicated that the use of gamma band as an additional band to the conventionally used delta, theta, alpha and beta gave improved classification accuracy.

Once the mental activities are successfully recognised by the ENN-RBP classifier, they could be used in conjunction with a translation algorithm such as Morse code (a few examples are shown in Figure 7.7) to construct English letters, Arabic numerals and punctuation marks to form words and complete sentences as proposed in another study [13]. Three mental activities[40] were used, one each to represent *dot*, *dash* and *space*. The *space* is required to denote the end of either a *dot* or *dash* and starting of a new *dot* or *dash.* This allows the subjects to focus on the sequence of mental tasks, without having to worry about the time duration of each mental task and is particularly useful for constructing letters like 'H' or 'S', which consist of consecutive *dots* or *dashes*. So, to construct the letter A, the subject will have to focus on mental activities in the sequence *space*, *dot*, *space*, *dash*, *space*. If relaxed mode, maths, object rotation activities are used to represent *space*, *dot* and *dash*, respectively then the sequence of mental activities would be in the form: relaxed, maths, relaxed, rotation, relaxed.

A	● —	B	— ● ● ●	0	— — — — —
E	●	H	● ● ● ●	X	— ●● —

Figure 7.7: Morse code examples showing usage of *dot* and *dash* to represent characters.

7.3 Short-term visual memory impairment in alcohol abusers using visual evoked potential signals

It is known that long term alcohol consumption can cause various impairments to the functional ability of the brain. The study in [14] using VEP has shown that long term alcohol abuse causes impairments in short-term visual memory. The utilised paradigm was modified sample to matching paradigm where a black and white line picture (S1) from the Snodgrass and Vanderwart set [15] was shown to the subject for 300 ms and the subject was asked to remember the picture. After an ISI of 1.6 s, a second picture (S2) was shown either in a matching (S2M) or non-matching (S2N) condition. In the S2M condition, the second picture was identical while for the S2N condition, the picture was completely different, even in terms of semantic category (for example, if S1 was of a snake, then S2N was something from a non-animal category). Figure 7.8 shows an example of the stimulus presentation. VEP responses were recorded from 61 channels following an extension of the 10-20 system with nose electrode used as ground and channel Cz as reference.

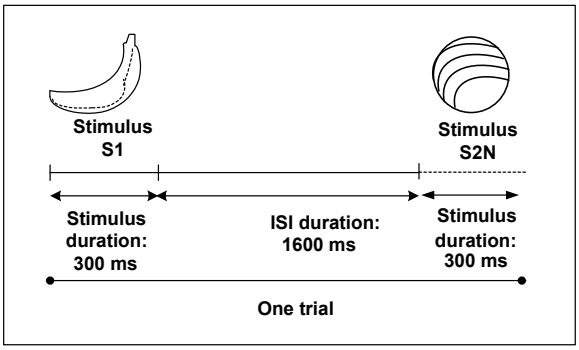


Figure 7.7: Stimulus presentation as used in the study [14].

VEP responses contaminated with eye blinks (above 73.3 μV) were removed from the analysis. The VEP signals were filtered from 0.02 to 50 Hz and sampled at 256 Hz. Each VEP response was recorded from 190 ms pre-stimulus and 1440 ms post-stimulus. The pre-stimulus data was used as baseline measure to compute the amplitude of the peaks. VEP responses were obtained from each 48 control and 77 alcoholics and the multi-trial responses were averaged to reduce background EEG. Multivariate analysis of variance (MANOVA) test were conducted for frontal, parietal, temporal and occipital regions and it was found that amplitudes of component c240 (positive peak evoked around 240 ms after stimulus onset) were significantly higher for controls as compared to alcoholics. However, this was found only for the S2N condition and not S2M, which could possibly be due to the simpler matching condition in S2M that is not sufficient enough to generate significant difference in the VEP responses.

The study also found that alcoholics responded with longer latency (the subjects were also required to press a mouse button for S2M and another button for S2N) and made more errors in deciding the matching/non-matching condition as compared to controls. Overall, the study has shown that c240 could be used as a short-term visual memory marker and that long-term alcohol abuse causes memory impairments.

7.4 Identification of heart sounds using phonocardiogram

PCG is a measure of the sonic vibrations of the heart and is often an useful additional measure to ECG in detecting problems with the heart. It is actually a digital representation of the sounds measured using stethoscope. A major difficulty in analysing PCG signal lies with the segmentation of a single cycle i.e. in the detection of the first (S1) and second heart (S2) sounds and most methods require ECG or carotid pulse and very few studies have attempted segmentation based on PCG alone.

The sounds S1 and S2 correspond to the normal heart sounds of *lup* and *dup*, respectively. S1 and S2 are the major heart sounds but there could be several other signal activities between first and second sounds such as S3, S4, murmurs, clicks and snaps for an abnormal heart. The accurate detection of such beats would be useful to diagnose heart problems and the first step is to segment a single PCG cycle.

In the study [16], the authors utilised PCG to detect three different heart sounds: normal, systolic murmur and diastolic murmur. The obtained PCG signals were normalised from 0 to 1 and low-pass filtered using Chebyshev type I filter with 3-dB cutoff frequency at 750 Hz. The MATLAB function, *filtfilt* was used to avoid phase distortion. Next, homomorphic filtering [17] was used to detect the peaks S1 and S2. Homomorphic filtering provides smooth envelope of the signal where the envelope resolution (smoothness) can be controlled.

Energy of the heart sound can be assumed to be a multiplication of slow varying S1 and S2 sounds with fast varying murmurs. Using logarithmic transformation, homomorphic filtering technique converts a combination of signals multiplied in time domain into linear combination[41]. The usage of a low-pass filter (another Chebyshev filter with cut-off at 10 Hz) removed the higher frequency components (i.e. murmurs, etc) and kept only the S1 and S2 components (but still in the logarithmic domain). This was followed by exponentiation to obtain the S1 and S2 components in time domain.

Figure 7.8 and 7.9 shows the method where it can be seen that the murmurs did not affect the envelope obtained during homomorphic filtering and hence accurate detection of S1 and S2 peaks was obtained.

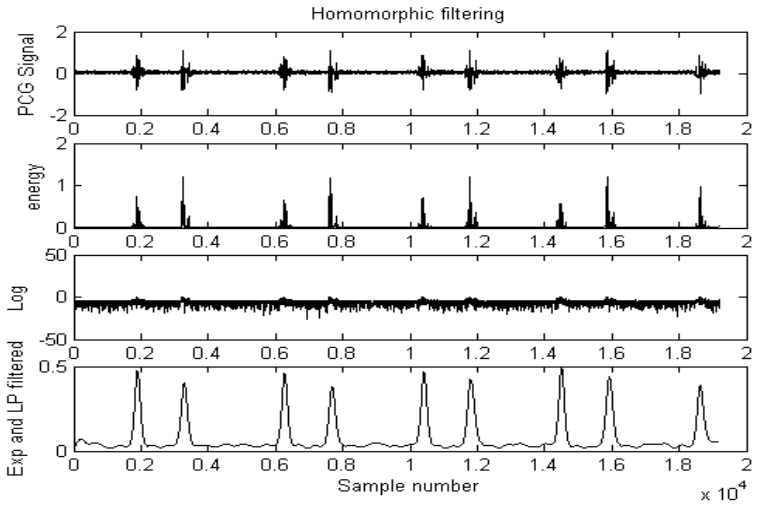


Figure 7.8: Peak detection for PCG signal (Normal). Figure from [16], used with permission from Elsevier.

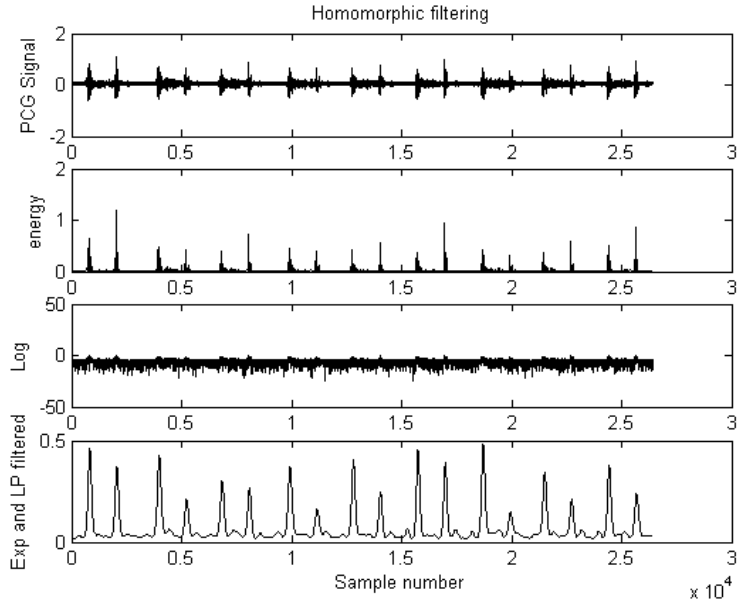


Figure 7.9: Peak detection for PCG signal (systolic murmur). Figure from [16], used with permission from Elsevier.

K-means clustering was then used to improve the S1-S2 cycle detection. Daubechies-2 wavelet was computed as features where the component formed by the wavelet detail coefficients at the second decomposition level was split into 32 sub windows and the energies of these windows were computed as features. Grow and Learn (GAL) and MLP-BP classifiers were used to classify these features into the three categories (normal, systolic murmur, diastolic murmur) and it was found that the performance of both the classifiers were about the same and gave a maximal recognition accuracy of 97.01%, indicating the success of the proposed method.

7.5 References

[1] B.M. Beasley, *Understanding EKGs: A Practical Approach*, Prentice Hall, 2003.

[2] R. Palaniappan, C.N. Gupta, C.K. Luk, and S.M. Krishnan, "Multi-parameter detection of ectopic heart beats," *Proceedings of IEEE International Workshop on Biomedical Circuits and Systems*, S2.4.1:4.4, 2004.

[3] The Massachusetts General Hospital/Marquette Foundation (MGH/MF) Waveform Database: Available: http://www.physionet.org/pn3/mghdb/.

[4] R. Palaniappan, C.S. Syan and P. Raveendran, "Current practices in electroencephalogram based brain-computer interfaces," published in M. Khosrow-Pour (ed.): *Encyclopedia of Information Science and Technology, 2nd ed.*," II:888-901, IGI Global, 2009.

[5] E. Donchin, K.M. Spencer, and R. Wijesinghe, "The mental prosthesis: assessing the speed of a P300-based brain-computer interface," *IEEE Transactions on Rehabilitation Engineering*, 8(2):174-179, June 2000.

[6] M. Thulasidas, G. Cuntai, and W. Jiankang, "Robust classification of EEG signal for brain-computer interface," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 14:24-29, 2006.

[7] C.N. Gupta and R. Palaniappan, "Enhanced detection of visual evoked potentials in brain-computer interface using genetic algorithm and cyclostationary analysis," *Computational Intelligence and Neuroscience (special issue on Brain-Computer Interfaces: Towards Practical Implementations and Potential Applications)*, DOI:10.1155/2007/28692, 12 pages, 2007.

[8] U. Hoffmann, J.M. Vesin, T. Ebrahimi, and K. Diserens, "An efficient p300-based brain-computer interface for disabled subjects," *Journal of Neuroscience Methods*, 167(1):115-125, 2008.

[9] Z.A. Keirn and J. I. Aunon, "A new mode of communication between man and his surroundings," *IEEE Transactions on Biomedical Engineering*, 37(12):1209-1214, 1990.

[10] R. Palaniappan, "Utilizing gamma band spectral power to improve mental task based brain computer interface design," *IEEE Transactions of Neural Systems and Rehabilitation Engineering*, 14(3): 299-303, 2006.

[11] D. Mandic and J. Chambers, *Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures and Stability*. Wiley, 2001.

[12] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," *Proceedings of IEEE International Conference on Neural Networks,* 586–591, 1993.

[13] R. Palaniappan, P. Raveendran, S. Nishida, and N. Saiwaki, "A new brain-computer interface design using fuzzy ARTMAP," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 10(3): 140–148, 2002.

[14] X.L. Zhang, H. Begleiter, B. Porjesz, and A. Litke, "Electrophysiological evidence of memory impairment in alcoholic patients," *Biological Psychiatry*, 42:1157-1171, 1997.

[15] J.G. Snodgrass and M. Vanderwart, "A standardized set of 260 Pictures: Norms for name agreement, image agreement, familiarity, and visual complexity," *Journal of Experimental Psychology: Human Learning and Memory*, 6(2):174-215, 1980.

[16] C.N. Gupta, R. Palaniappan, S. Swaminathan, and S. M. Krishnan, "Neural network classification of homomorphic segmented heart sounds," *Applied Soft Computing*, 7(1):286-297, 2007.

[17] J.R. Deller, J.G. Proakis, and J.L. Hansen, *Discrete Time Processing of Speech Signals*, Prentice Hall, 1993.

Endnotes

1. Sometimes referred to as biosignal analysis, in short.
2. Pathological conditions are abnormal conditions that are often indications of the presence of disease (s).
3. With the scope of the text in mind, this explanation of how a neuron functions is greatly simplified.
4. Hence, FP would denote frontal-parietal.
5. Most EP components are named using their typical latency after stimulus onset. Hence N100 refers to a component that occurs around 100 ms. Sometimes, the peaks are named in sequence, so P1 refers to the first positive peak, N2 refers to the second negative peak etc.
6. The MathWorks Inc.
7. Known as a transfer function.
8. Filtering methods will be studied in Chapter Four.
9. The actual process of analogue to digital conversion involves sampling, quantisation and coding but for the purposes of the discussions in this book, sampling process is sufficient as we'll only utilise discrete-time signals.
10. Though they can be recorded using multiple sensors (channels), the data from each sensor are still 1D time series.
11. The spectral (i.e. frequency) magnitude at *Fs* is same as at 0 Hz.
12. Here *j* represents $\sqrt{-1}$ but some books use *i* instead.
13. If MATLAB is used, the *n* index starts from 1.
14. Hence, 5 Hz will not be represented in the spectrum.
15. As the signal is periodic, both starting and ending points are the same.
16. Since frequency spacing is 2 Hz.
17. Square block in Figure 4.14 – in this case, it delays $x[n]$ by one point to give $x[n\text{-}1]$.
18. FIR filters are stable as there is no feedback (i.e. all poles are at origin in z-plane).
19. This is since when gain equals 1, $20\log_{10}(1)=0$ dB.
20. For example, the filter coefficients for (4.13) is $h[n]=[1\ 3\ 2\ -2\ -3\ -1]$, i.e. $B[1]=1$, $B[2]=3$, $B[3]=2$, $B[4]$-2, $B[5]=-3$ and $B[6]=-1$ considering (4.2).
21. But height of largest ripple remains the same independent of length.
22. If it is desired to pass input signal components in a certain frequency range undistorted in both magnitude and phase, then the filter should exhibit a unity magnitude response and a zero-phase response in the band of interest. Alternatively, a unity magnitude response and a linear-phase response in the band of interest are also acceptable. Linear-phase response denotes that all frequency components are delayed by a same amount.
23. Most filters do not have linear phase response in stopband but this does not concern us as the components in this spectral range will be attenuated significantly.
24. Part of the family of parametric methods.

25. The * here denotes multiplication.
26. These EEG signals were obtained from [3].
27. EEG data recorded in USA and obtained from [3].
28. Positive peak can be identified for point n if $x[n]$-$x[n$-1]>0 and $x[n]$-$x[n$+1]>0 and vice versa for negative peak.
29. Typically k=sqrt(N), where N is the number of training patterns.
30. The alcoholic subjects here are those that have a long history of alcohol abuse; the non-alcoholics are control subjects.
31. There are even advanced methods such as Voronoi Diagram Approach, Gabriel Graph Approach, and Relative Neighbour Graph Approach but we will not look at these here.
32. Note that the input layer is sometimes not considered as a layer. Hence, this three layer MLP is also known as two-layer network.
33. This is the usual practice, though there are variations.
34. Rule of thumb: minimum number of hidden units should be the same as input units.
35. It is also possible to update the weights after every pattern presentation.
36. The distribution of the classification performance data is assumed to be normal, if this is not known, or if the distribution is not normal, then ranked based test should be utilised.
37. Subject imagines moving his/her limb, for example imagination of hand clasping.
38. In a recent study [8], the entire signal is used as feature after downsampling.
39. Similar to conventional password systems used to login into computers but using colours and instead of keyboard inputs, responses from the brain are used. This is less prone to fraud as 'shoulder surfing' can be avoided.
40. It was shown in this study that different subjects had different combination of suitable three mental activities.
41. I.e. multiplicative to addition operation.