**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | 29 May 2020 | **Name:** | Anupama J S |
| **Course:** | DSP | **USN:** | 4AL16EC005 |
| **Topic:** | 1. Fourier Transforms 2. FFT 3. FFT Fast Fourier Transform Matlab 4. FIR and IIR Filters 5. Study and analysis FIR and IIR using FDA tool in MatLab 6. Introduction to WT 7. CWT & DWT 8. Implementation of signal Filtering signal using WT in MatLAb 9. Short-time Fourier Transform and the Spectogram 10. Welch's method and windowing 11. ECG Signal Analysis Using MATLAB | **Semester & Section:** | 8th sem “A”section |
| **Github Repository:** | AnupamaJS |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **ABOUT FOURIER TRANSFORM**  In mathematics, a Fourier transform (FT) is a mathematical transform which decomposes a function (often a function of time, or a signal) into its constituent frequencies, such as the expression of a musical chord in terms of the volumes and frequencies of its constituent notes. The term Fourier transform refers to both the frequency domain representation and the mathematical operation that associates the frequency domain representation to a function of time.  The Fourier transform of a function of time is a complex-valued function of frequency, whose magnitude (absolute value) represents the amount of that frequency present in the original function, and whose argument is the phase offset of the basic sinusoid in that frequency. The Fourier transform is not limited to functions of time, but the domain of the original function is commonly referred to as the time domain. There is also an inverse Fourier transform that mathematically synthesizes the original function from its frequency domain representation, as proven by the Fourier inversion theorem.  **FAST FOURIER TRANSFORM**  A fast Fourier transform (FFT) is an algorithm that computes the discrete Fourier transform (DFT) of a sequence, or its inverse (IDFT). Fourier analysis converts a signal from its original domain (often time or space) to a representation in the frequency domain and vice versa. The DFT is obtained by decomposing a sequence of values into components of different frequencies.[1] This operation is useful in many fields, but computing it directly from the definition is often too slow to be practical. An FFT rapidly computes such transformations by factorizing the DFT matrix into a product of sparse (mostly zero) factors.[2] As a result, it manages to reduce the complexity of computing the DFT from {\displaystyle O\left(N^{2}\right)}{\displaystyle O\left(N^{2}\right)}, which arises if one simply applies the definition of DFT, to {\displaystyle O(N\log N)}O(N\log N), where {\displaystyle N}N is the data size. The difference in speed can be enormous, especially for long data sets where N may be in the thousands or millions. In the presence of round-off error, many FFT algorithms are much more accurate than evaluating the DFT definition directly or indirectly. There are many different FFT algorithms based on a wide range of published theories, from simple complex-number arithmetic to group theory and number theory.  Fast Fourier transforms are widely used for applications in engineering, music, science, and mathematics. The basic ideas were popularized in 1965, but some algorithms had been derived as early as 1805. In 1994, Gilbert Strang described the FFT as "the most important numerical algorithm of our lifetime” and it was included in Top 10 Algorithms of 20th Century by the IEEE magazine Computing in Science & Engineering  Y = fft(X)  Y = fft(X,n)  Y = fft(X,n,dim)  **Description**  example  Y = fft(X) computes the discrete Fourier transform (DFT) of X using a fast Fourier transform (FFT) algorithm.  If X is a vector, then fft(X) returns the Fourier transform of the vector.  If X is a matrix, then fft(X) treats the columns of X as vectors and returns the Fourier transform of each column.  If X is a multidimensional array, then fft(X) treats the values along the first array dimension whose size does not equal 1 as vectors and returns the Fourier transform of each vector.  example  Y = fft(X,n) returns the n-point DFT. If no value is specified, Y is the same size as X.  If X is a vector and the length of X is less than n, then X is padded with trailing zeros to length n.  If X is a vector and the length of X is greater than n, then X is truncated to length n.  If X is a matrix, then each column is treated as in the vector case.  If X is a multidimensional array, then the first array dimension whose size does not equal 1 is treated as in the vector case.  example  Y = fft(X,n,dim) returns the Fourier transform along the dimension dim. For example, if X is a matrix, then fft(X,n,2) returns the n-point Fourier transform of each row.  **FIR AND IIR FILTERS**  C:\Users\User\Downloads\WhatsApp Image 2020-05-29 at 7.58.38 PM (2).jpeg  C:\Users\User\Downloads\WhatsApp Image 2020-05-29 at 7.58.38 PM (3).jpeg  Filters have a variety of applications in data acquisition and analysis. They are used to alter the frequency content of a time signal by either reducing or amplifying certain frequencies.  For example, as shown in Figure 1, a low pass filter affects frequency content in a signal in three different ways: Some frequency content remains unchanged, while other frequency content is either reduced in amplitude or removed entirely from the signal.  filter\_overview.png  C:\Users\User\Downloads\WhatsApp Image 2020-05-29 at 8.22.28 PM.jpeg  Figure 1: A low pass filter passes low frequencies unaltered (left) and removes high frequencies (right).  Filters can also amplify frequency content, not just reduce or remove it. The amount that a filter adjusts the amplitude of a signal can be expressed in either linear terms (i.e., amplification factor) or decibels of gain/attenuation as shown in Figure 2.  filter\_gain\_attenutation.png  C:\Users\User\Downloads\WhatsApp Image 2020-05-29 at 8.24.12 PM.jpeg  Figure 2: Left graph – Linear filter amplitude in multiplication factor versus frequency, Right graph – Same filter in decibel scale of attenuation versus frequency.  The linear scale has the following equivalents in decibel scale: Halving of linear amplitude is 6 dB of attenuation No adjustment of amplitude corresponds to linear gain of 1, or 0 dB, of attenuation While useful to view filter characteristics in the frequency domain, filters perform their work in the time domain  filter\_what\_it\_does.png  C:\Users\User\Downloads\WhatsApp Image 2020-05-29 at 8.23.52 PM.jpeg  Figure 3: An input signal with high frequency noise is passed through a low pass filter. The resulting output has the high frequency noise removed, resulting in a clean signal.  A filter takes a time domain signal as input, modifies the frequency content, and outputs a new time domain signal. This can be useful in a variety of applications.  **INTRODUCTION TO WT**  **C:\Users\User\Downloads\WhatsApp Image 2020-05-29 at 7.58.39 PM.jpeg**  Wavelet Analysis and its Applications, Volume 1: An Introduction to Wavelets provides an introductory treatise on wavelet analysis with an emphasis on spline-wavelets and time-frequency analysis. This book is divided into seven chapters. Chapter 1 presents a brief overview of the subject, including classification of wavelets, integral wavelet transform for time-frequency analysis, multi-resolution analysis highlighting the important properties of splines, and wavelet algorithms for decomposition and reconstruction of functions. The preliminary material on Fourier analysis and signal theory is covered in Chapters 2 and 3. Chapter 4 covers the introductory study of cardinal splines, while Chapter 5 describes a general approach to the analysis and construction of scaling functions and wavelets. Spline-wavelets are deliberated in Chapter 6. The last chapter is devoted to an investigation of orthogonal wavelets and wavelet packets. This volume serves as a textbook for an introductory one-semester course on “wavelet analysis for upper-division undergraduate or beginning graduate mathematics and engineering students.  **CWT & DWT**  **C:\Users\User\Downloads\WhatsApp Image 2020-05-29 at 7.58.39 PM (1).jpeg**  This topic describes the major differences between the continuous wavelet transform (CWT) and the discrete wavelet transform (DWT) – both decimated and nondecimated versions. cwt is a discretized version of the CWT so that it can be implemented in a computational environment. This discussion focuses on the 1-D case, but is applicable to higher dimensions. The cwt wavelet transform compares a signal with shifted and scaled (stretched or shrunk) copies of a basic wavelet. If ψ(t) is a wavelet centered at t=0 with time support on [-T/2, T/2], then is centered at t = u with time support [-sT/2+u, sT/2+u]. The cwt function uses L1 normalization so that all frequency amplitudes are normalized to the same value. If 0<s<1, the wavelet is contracted (shrunk) and if s>1, the wavelet is stretched. The mathematical term for this is dilation. See Continuous Wavelet Transform and Scale-Based Analysis for examples of how this operation extracts features in the signal by matching it against dilated and translated wavelets. The major difference between the CWT and discrete wavelet transforms, such as the dwt and modwt, is how the scale parameter is discretized. The CWT discretizes scale more finely than the discrete wavelet transform. In the CWT, you typically fix some base which is a fractional power of two, for example, 2 where v is an integer greater than 1. The v parameter is often referred to as the number of “voices per octave”. Different scales are obtained by raising this base scale to positive integer powers, for example 2  j=1,2,3,…. The translation parameter in the CWT is discretized to integer values, denoted here by m. The resulting discretized wavelets for the CWT are The reason v is referred to as the number of voices per octave is because increasing the scale by an octave (a doubling) requires v intermediate scales. Take for example 2 There are v intermediate steps. Common values for v are 10,12,14,16, and 32. The larger the value of v, the finer the discretization of the scale parameter, s. However, this also increases the amount of computation required because the CWT must be computed for every scale. The difference between scales on a log2 scale is 1/v. See CWT-Based Time-Frequency Analysis and Continuous Wavelet Analysis of Modulated Signals for examples of scale vectors with the CWT. In the discrete wavelet transform, the scale parameter is always discretized to integer powers of 2, 2j, j=1,2,3,..., so that the number of voices per octave is always 1. The difference between scales on a log2 scale is always 1 for discrete wavelet transforms. Note that this is a much coarser sampling of the scale parameter, s, than is the case with the CWT. Further, in the decimated (downsampled) discrete wavelet transform (DWT), the translation parameter is always proportional to the scale. This means that at scale, 2j, you always translate by 2jm where m is a nonnegative integer. In nondecimated discrete wavelet transforms like modwt and swt, the scale parameter is restricted to powers of two, but the translation parameter is an integer like in the CWT. The discretized wavelet for the DWT  **WELCH'S METHOD AND WINDOWING**  C:\Users\User\Downloads\WhatsApp Image 2020-05-29 at 7.58.40 PM.jpeg  Welch's method, named after [Peter D. Welch](https://en.wikipedia.org/w/index.php?title=Peter_D._Welch&action=edit&redlink=1), is an approach for [spectral density estimation](https://en.wikipedia.org/wiki/Spectral_density_estimation). It is used in [physics](https://en.wikipedia.org/wiki/Physics), [engineering](https://en.wikipedia.org/wiki/Engineering), and applied [mathematics](https://en.wikipedia.org/wiki/Mathematics) for estimating the [power](https://en.wikipedia.org/wiki/Electric_power) of a [signal](https://en.wikipedia.org/wiki/Signal_(electrical_engineering)) at different [frequencies](https://en.wikipedia.org/wiki/Frequency). The method is based on the concept of using [periodogram](https://en.wikipedia.org/wiki/Periodogram" \o "Periodogram) spectrum estimates, which are the result of converting a signal from the time domain to the [frequency domain](https://en.wikipedia.org/wiki/Frequency_domain). Welch's method is an improvement on the standard [periodogram](https://en.wikipedia.org/wiki/Periodogram" \o "Periodogram) spectrum estimating method and on [Bartlett's method](https://en.wikipedia.org/wiki/Bartlett%27s_method), in that it reduces noise in the estimated [power spectra](https://en.wikipedia.org/wiki/Power_spectrum) in exchange for reducing the frequency resolution. Due to the noise caused by imperfect and finite data, the noise reduction from Welch's method is often desired.  The Welch method is based on [Bartlett's method](https://en.wikipedia.org/wiki/Bartlett%27s_method) and differs in two ways:   1. The signal is split up into overlapping segments: the original data segment is split up into L data segments of length M, overlapping by D points.    1. If D = M / 2, the overlap is said to be 50%    2. If D = 0, the overlap is said to be 0%. This is the same situation as in the [Bartlett's method](https://en.wikipedia.org/wiki/Bartlett%27s_method). 2. The overlapping segments are then windowed: After the data is split up into overlapping segments, the individual L data segments have a window applied to them (in the time domain).    1. Most [window functions](https://en.wikipedia.org/wiki/Window_function) afford more influence to the data at the center of the set than to data at the edges, which represents a loss of information. To mitigate that loss, the individual data sets are commonly overlapped in time (as in the above step).    2. The windowing of the segments is what makes the Welch method a "modified" [periodogram](https://en.wikipedia.org/wiki/Periodogram" \o "Periodogram).   After doing the above, the [periodogram](https://en.wikipedia.org/wiki/Periodogram" \o "Periodogram) is calculated by computing the [discrete Fourier transform](https://en.wikipedia.org/wiki/Discrete_Fourier_transform), and then computing the squared magnitude of the result. The individual [periodograms](https://en.wikipedia.org/wiki/Periodogram" \o "Periodogram) are then averaged, which reduces the variance of the individual power measurements. The end result is an array of power measurements vs. frequency "bin". |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | 29 May 2020 | **Name:** | Anupama J S |
| **Course:** | Python | **USN:** | 4AL16EC005 |
| **Topic:** | Object oriented programming | **Semester & Section:** | 8th sem “A”section |
| **Github Repository:** | AnupamaJS |  |  |
| **AFTERNOON SESSION DETAILS** | | | |
| **OBJECT-ORIENTED PROGRAMMING**  **C:\Users\User\Downloads\WhatsApp Image 2020-05-29 at 7.58.38 PM.jpeg**  Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which can contain data, in the form of fields (often known as attributes or properties), and code, in the form of procedures (often known as methods). A feature of objects is an object's procedures that can access and often modify the data fields of the object with which they are associated (objects have a notion of "this" or "self"). In OOP, computer programs are designed by making them out of objects that interact with one another. OOP languages are diverse, but the most popular ones are class-based, meaning that objects are instances of classes, which also determine their types.  Many of the most widely used programming languages (such as C++, Java, Python, etc.) are multi-paradigm and they support object-oriented programming to a greater or lesser degree, typically in combination with imperative, procedural programming. Significant object-oriented languages include Java, C++, C#, Python, R, PHP, JavaScript, Ruby, Perl, Object Pascal, Objective-C, Dart, Swift, Scala, Kotlin, Common Lisp, MATLAB, and Smalltalk.  Object-oriented programming uses objects, but not all of the associated techniques and structures are supported directly in languages that claim to support OOP. The features listed below are common among languages considered to be strongly class- and object-oriented (or multi-paradigm with OOP support), with notable exceptions mentioned  **Shared with non-OOP predecessor languages**  Variables that can store information formatted in a small number of built-in data types like integers and alphanumeric characters. This may include data structures like strings, lists, and hash tables that are either built-in or result from combining variables using memory pointers.  Procedures – also known as functions, methods, routines, or subroutines – that take input, generate output, and manipulate data. Modern languages include structured programming constructs like loops and conditionals. Modular programming support provides the ability to group procedures into files and modules for organizational purposes. Modules are namespaced so identifiers in one module will not conflict with a procedure or variable sharing the same name in another file or module.  **INHERITANCE**  C:\Users\User\Downloads\WhatsApp Image 2020-05-29 at 8.44.11 PM.jpeg  In [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming), inheritance is the mechanism of basing an [object](https://en.wikipedia.org/wiki/Object_(computer_science)) or [class](https://en.wikipedia.org/wiki/Class_(computer_programming)) upon another object ([prototype-based inheritance](https://en.wikipedia.org/wiki/Prototype-based_programming)) or class ([class-based inheritance](https://en.wikipedia.org/wiki/Class-based_programming)), retaining similar implementation. Also defined as deriving new classes ([sub classes](https://en.wikipedia.org/wiki/Inheritance_(object-oriented_programming)#Subclasses_and_superclasses)) from existing ones such as super class or [base class](https://en.wikipedia.org/wiki/Fragile_base_class) and then forming them into a hierarchy of classes. In most class-based object-oriented languages, an object created through inheritance, a "child object", acquires all the properties and behaviors of the "parent object" , with the exception of: [constructors](https://en.wikipedia.org/wiki/Constructor_(object-oriented_programming)), destructor, [overloaded operators](https://en.wikipedia.org/wiki/Operator_overloading) and [friend functions](https://en.wikipedia.org/wiki/Friend_function) of the base class. Inheritance allows programmers to create classes that are built upon existing classes, to specify a new implementation while maintaining the same behaviors ([realizing an interface](https://en.wikipedia.org/wiki/Class_diagram#Realization/Implementation)), to reuse code and to independently extend original software via public classes and interfaces. The relationships of objects or classes through inheritance give rise to a [directed graph](https://en.wikipedia.org/wiki/Directed_graph).  Inheritance was invented in 1969 for [Simula](https://en.wikipedia.org/wiki/Simula" \o "Simula) and is now used throughout many object-oriented programming languages such as [Java](https://en.wikipedia.org/wiki/Java_(programming_language)), C++ or Python.  An inherited class is called a subclass of its parent class or super class. The term "inheritance" is loosely used for both class-based and prototype-based programming, but in narrow use the term is reserved for class-based programming (one class *inherits from* another), with the corresponding technique in prototype-based programming being instead called [*delegation*](https://en.wikipedia.org/wiki/Delegation_(object-oriented_programming)) (one object *delegates to* another).  Inheritance should not be confused with [subtyping](https://en.wikipedia.org/wiki/Subtyping). In some languages inheritance and subtyping agree, whereas in others they differ; in general, subtyping establishes an [is-a](https://en.wikipedia.org/wiki/Is-a" \o "Is-a) relationship, whereas inheritance only reuses implementation and establishes a syntactic relationship, not necessarily a semantic relationship (inheritance does not ensure [behavioral subtyping](https://en.wikipedia.org/wiki/Behavioral_subtyping" \o "Behavioral subtyping)). To distinguish these concepts, subtyping is also known as *interface inheritance*, whereas inheritance as defined here is known as *implementation inheritance* or *code inheritance*.[]](https://en.wikipedia.org/wiki/Inheritance_(object-oriented_programming)#cite_note-Mikhajlov-6) Still, inheritance is a commonly used mechanism for establishing subtype relationships.  Inheritance is contrasted with [object composition](https://en.wikipedia.org/wiki/Object_composition), where one object *contains* another object (or objects of one class contain objects of another class); see [composition over inheritance](https://en.wikipedia.org/wiki/Composition_over_inheritance). Composition implements a [has-a](https://en.wikipedia.org/wiki/Has-a" \o "Has-a) relationship, in contrast to the is-a relationship of subtyping. | | | |