**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **25/5/2020** | **Name:** | **GAURAV NR** |
| **Course:** | **Digital signal processing** | **USN:** | **4AL15EC025** |
| **Topic:** | **Introduction to fourier series and fourier transform, Hilbert transform, Fourier series using matlab and python** | **Semester & Section:** | **8th sem A sec** |
| **Github Repository:** | **gaurav** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session** |
| **Report –**  A Fourier [series](https://en.wikipedia.org/wiki/Series_(mathematics)) is a [periodic function](https://en.wikipedia.org/wiki/Periodic_function) composed of harmonically related [sinusoids](https://en.wikipedia.org/wiki/Sine_wave), combined by a weighted summation. With appropriate weights, one cycle (or *period*) of the summation can be made to approximate an arbitrary function in that interval (or the entire function if it too is periodic). As such, the summation is a synthesis of another function. The [discrete-time Fourier transform](https://en.wikipedia.org/wiki/Discrete-time_Fourier_transform) is an example of Fourier series. The process of deriving the weights that describe a given function is a form of [Fourier analysis](https://en.wikipedia.org/wiki/Fourier_analysis). For functions on unbounded intervals, the analysis and synthesis analogies are [Fourier transform](https://en.wikipedia.org/wiki/Fourier_transform) and inverse transform.  A Fourier series is an expansion of a [periodic function](https://mathworld.wolfram.com/PeriodicFunction.html) f(x) in terms of an infinite sum of [sines](https://mathworld.wolfram.com/Sine.html) and [cosines](https://mathworld.wolfram.com/Cosine.html). Fourier series make use of the [orthogonality](https://mathworld.wolfram.com/OrthogonalFunctions.html) relationships of the [sine](https://mathworld.wolfram.com/Sine.html) and [cosine](https://mathworld.wolfram.com/Cosine.html) functions. The computation and study of Fourier series is known as [harmonic analysis](https://mathworld.wolfram.com/HarmonicAnalysis.html) and is extremely useful as a way to break up an *arbitrary* periodic function into a set of simple terms that can be plugged in, solved individually, and then recombined to obtain the solution to the original problem or an approximation to it to whatever accuracy is desired or practical. Examples of successive approximations to common functions using Fourier series are illustrated above.  In particular, since the [superposition principle](https://mathworld.wolfram.com/SuperpositionPrinciple.html) holds for solutions of a linear homogeneous [ordinary differential equation](https://mathworld.wolfram.com/OrdinaryDifferentialEquation.html), if such an equation can be solved in the case of a single sinusoid, the solution for an arbitrary function is immediately available by expressing the original function as a Fourier series and then plugging in the solution for each sinusoidal component. In some special cases where the Fourier series can be summed in closed form, this technique can even yield analytic solutions.  Any set of functions that form a [complete orthogonal system](https://mathworld.wolfram.com/CompleteOrthogonalSystem.html) have a corresponding [generalized Fourier series](https://mathworld.wolfram.com/GeneralizedFourierSeries.html) analogous to the Fourier series. For example, using orthogonality of the roots of a [Bessel function of the first kind](https://mathworld.wolfram.com/BesselFunctionoftheFirstKind.html) gives a so-called [Fourier-Bessel series](https://mathworld.wolfram.com/Fourier-BesselSeries.html). The computation of the (usual) Fourier series is based on the integral identities   |  |  |  |  | | --- | --- | --- | --- | | int_(-pi)^pisin(mx)sin(nx)dx | = | pidelta_(mn) | (1) | | int_(-pi)^picos(mx)cos(nx)dx | = | pidelta_(mn) | (2) | | int_(-pi)^pisin(mx)cos(nx)dx | = | 0 | (3) | | int_(-pi)^pisin(mx)dx | = | 0 | (4) | | int_(-pi)^picos(mx)dx | = | 0 | (5) |   for m,n!=0, where delta_(mn) is the [Kronecker delta](https://mathworld.wolfram.com/KroneckerDelta.html).  Using the method for a [generalized Fourier series](https://mathworld.wolfram.com/GeneralizedFourierSeries.html), the usual Fourier series involving sines and cosines is obtained by taking f_1(x)=cosx and f_2(x)=sinx. Since these functions form a [complete orthogonal system](https://mathworld.wolfram.com/CompleteOrthogonalSystem.html) over [-pi,pi], the Fourier series of a function f(x) is given by   |  |  | | --- | --- | | f(x)=1/2a_0+sum_(n=1)^inftya_ncos(nx)+sum_(n=1)^inftyb_nsin(nx), | (6) |   Where   |  |  |  |  | | --- | --- | --- | --- | | a_0 | = | 1/piint_(-pi)^pif(x)dx | (7) | | a_n | = | 1/piint_(-pi)^pif(x)cos(nx)dx | (8) | | b_n | = | 1/piint_(-pi)^pif(x)sin(nx)dx | (9) |   and n=1, 2, 3, .... Note that the coefficient of the constant term a_0 has been written in a special form compared to the general form for a [generalized Fourier series](https://mathworld.wolfram.com/GeneralizedFourierSeries.html) in order to preserve symmetry with the definitions of a_n and b_n.  The Fourier cosine coefficient a_n and sine coefficient b_n are implemented in the [Wolfram Language](https://www.wolfram.com/language/) as [FourierCosCoefficient](https://reference.wolfram.com/language/ref/FourierCosCoefficient.html)[expr, t, n] and [FourierSinCoefficient](https://reference.wolfram.com/language/ref/FourierSinCoefficient.html)[expr, t, n], respectively. A Fourier series converges to the function f^_ (equal to the original function at points of continuity or to the average of the two limits at points of discontinuity)   |  |  | | --- | --- | | f^_={1/2[lim_(x->x_0^-)f(x)+lim_(x->x_0^+)f(x)]   for -pi<x_0<pi; 1/2[lim_(x->-pi^+)f(x)+lim_(x->pi_-)f(x)]   for x_0=-pi,pi | (10) |   if the function satisfies so-called [Dirichlet boundary conditions](https://mathworld.wolfram.com/DirichletBoundaryConditions.html). [Dini's test](https://mathworld.wolfram.com/DinisTest.html) gives a condition for the convergence of Fourier series.  **FOURIER TRANSFORM**    A Fourier transform (FT) is a [mathematical transform](https://en.wikipedia.org/wiki/Integral_transform) which decomposes a [function](https://en.wikipedia.org/wiki/Function_(mathematics)) (often a [function of time,](https://en.wikipedia.org/wiki/Time-variant_system) or a [signal](https://en.wikipedia.org/wiki/Signal)) into its constituent [frequencies](https://en.wikipedia.org/wiki/Frequency), such as the expression of a musical [chord](https://en.wikipedia.org/wiki/Chord_(music)) in terms of the volumes and frequencies of its constituent notes. The term *Fourier transform* refers to both the [frequency domain](https://en.wikipedia.org/wiki/Frequency_domain) representation and the [mathematical operation](https://en.wikipedia.org/wiki/Operation_(mathematics)) that associates the frequency domain representation to a function of time.  The Fourier transform of a function of time is a [complex-valued function](https://en.wikipedia.org/wiki/Complex-valued_function) of frequency, whose magnitude ([absolute value](https://en.wikipedia.org/wiki/Absolute_value#Complex_numbers)) represents the amount of that frequency present in the original function, and whose [argument](https://en.wikipedia.org/wiki/Argument_(complex_analysis)) is the [phase offset](https://en.wikipedia.org/wiki/Phase_offset) of the basic [sinusoid](https://en.wikipedia.org/wiki/Sine_wave) in that frequency. The Fourier transform is not limited to functions of time, but the [domain](https://en.wikipedia.org/wiki/Domain_of_a_function) of the original function is commonly referred to as the [*time domain*](https://en.wikipedia.org/wiki/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](https://en.wikipedia.org/wiki/Fourier_inversion_theorem).      A [sinusoidal](https://en.wikipedia.org/wiki/Sine_wave) curve, with peak amplitude (1), peak-to-peak (2), [RMS](https://en.wikipedia.org/wiki/Root_mean_square) (3), and wave period (4).        Illustration of phase shift *θ*.    {\displaystyle \scriptstyle f(t)}{\displaystyle \scriptstyle {\hat {f}}(\omega )}{\displaystyle \scriptstyle g(t)}{\displaystyle \scriptstyle {\hat {g}}(\omega )}{\displaystyle \scriptstyle t}{\displaystyle \scriptstyle \omega }{\displaystyle \scriptstyle t}{\displaystyle \scriptstyle \omega }  Linear operations performed in one domain (time or frequency) have corresponding operations in the other domain, which are sometimes easier to perform. The operation of [differentiation](https://en.wikipedia.org/wiki/Derivative) in the time domain corresponds to multiplication by the frequency, so some [differential equations](https://en.wikipedia.org/wiki/Differential_equation) are easier to analyze in the frequency domain. Also, [convolution](https://en.wikipedia.org/wiki/Convolution) in the time domain corresponds to ordinary multiplication in the frequency domain (see [Convolution theorem](https://en.wikipedia.org/wiki/Convolution_theorem)). After performing the desired operations, transformation of the result can be made back to the time domain. [Harmonic analysis](https://en.wikipedia.org/wiki/Harmonic_analysis) is the systematic study of the relationship between the frequency and time domains, including the kinds of functions or operations that are "simpler" in one or the other, and has deep connections to many areas of modern mathematics. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date:25/5/2020** |  | **Name: Gaurav** |  | |
| **Course: Python** |  | **USN:4AL15EC025** |  | |
| **Topic: Fixing programming errors, Application 3: Build a website blocker** |  | **Sem :8th sem**  **Section: A sec** |  | |
| **AFTERNOON SESSION DETAILS** | | | |
| **REPORT:**  Here is an another example for if not any(website in line for website in website\_list)   1. >>> lines  = ["trees are good", "pool is fresh", "face is round"] 2. >>> website\_list = ["face", "clock", "trend"] 3. >>> for line in lines: 4. ...     any(website in line for website in website\_list) 5. ... 6. False 7. False 8. True   We start iterating over the items of website\_list using a for loop. In the first iteration we would have:  any(website in "trees are good" for website in website\_list)  Inside the parenthesis of any() there's another loop that iterates over website\_list:  ("face" in "trees are good")   1. ("clock" in "trees are good") 2. ("trend" in "trees are good")   If any of the above is True you get the expression evaluated to True. In this case none of them is true, so you get False. If you want to return True (if all of them are True), use all() instead of any().  So, the part any(website in line for website in website\_list) will either be equal to True or False.  Scheduling a Python Program on a Server  Scheduling a Python program on a 24/7 server  Keeping your computer on 24-7 is not practical, so if you want to execute a Python script at a particular time every day, you probably need a computer that is on all the time.  PythonAnywhere gives you access to such a 24-7 computer. You can upload a Python script and schedule it to run at a certain time every day. This availability can be useful, for example, when you want to extract some values (e.g., weather data) from a website and generate a text file with the value or other reports every day.  To schedule a Python script for execution on PythonAnywhere, follow these simple steps:   1. Sign up for a free account at https://www.pythonanywhere.com. 2. Go to your Dashboard, Files, Upload a File, and upload the Python file you want to schedule for execution. 3. Go to Tasks and set the time of the day you want your script to be executed and type in the name of the Python file you  uploaded (e.g., myscript.py). Note that the time you enter should be in UTC. 4. Click the Create button and you’re done.   Your Python file will now be executed every day at your specified time. If you don't have a Python script and you’re still confused about the benefit of this, here is a very simple Python script that you can  use to try the above steps:  If you don’t have a Python script and you’re still confused about the benefits of this PythonAnywhere feature, here is a very simple Python script you can use to schedule for execution:   1. from datetime import datetime 2. with open(datetime.now().strftime("%Y-%m-%d-%H-%M-%S"), "w") as myfile: 3. myfile.write("Hi there!")   The above code creates a text file and writes the string “Hi there!”  in that text file. The name of the text file will be the current date and time. For example one file name example would be 2018-02-16-18-20-33.txt.  That name is generated by datetime.now() indicating the date and time the script was executed.  Every time the script is executed, the script generates a new text file with a different name. You will have a new text file created every day. | | | |