**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **23 May 2020** | **Name:** | **Srinidhi J C** |
| **Course:** | **Digital Signal Processing** | **USN:** | **4al16ec078** |
| **Topic:** | |  | | --- | | Introduction to Fourier Series & Fourier Transform | | Fourier Series – Part 1 | | Fourier Series – Part 2 | | Inner Product in Hilbert Transform | | Complex Fourier Series | | Fourier Series using Matlab  (Use Octave to execute the code) | | Fourier Series using Python  (Experience implementation using Python) | | Fourier Series and Gibbs Phenomena Using Matlab | | **Semester & Section:** | **8th-Sem, B-Sec** |
| **Github Repository:** | **https://github.com/alvas-education-foundation/SrinidhiJC078.git** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session**  **A close up of a map  Description automatically generatedA close up of a logo  Description automatically generatedA close up of a map  Description automatically generatedA screenshot of a cell phone  Description automatically generatedA picture containing drawing, sitting  Description automatically generatedA close up of a map  Description automatically generated**  **A picture containing text, blackboard, man, swinging  Description automatically generatedA screen shot of a person  Description automatically generated**  **A screen shot of a person  Description automatically generatedA screen shot of a person  Description automatically generated** |
| **Report -**  Fourier’s seminal work provided the mathematical foundation for Hilbert spaces, operator theory, approximation theory, and the subsequent revolution in analytical and computational mathematics. Fast forward two hundred years, and the fast Fourier transform has become the cornerstone of computational mathematics, enabling real-time image and audio compression, global communication networks, modern devices and hardware, numerical physics and engineering at scale, and advanced data analysis. Simply put, the fast Fourier transform has had a more signiﬁcant and profound role in shaping the modern world than any other algorithm to date.  **Fourier series and Fourier transforms**  The discrete and continuous formulations should match in the limit of data with inﬁnitely ﬁne resolution. The Fourier series and transform are intimately related to the geometry of inﬁnite-dimensional function spaces, or Hilbert spaces, which generalize the notion of vector spaces to include functions with inﬁnitely many degrees of freedom. Thus, we begin with an introduction to function spaces.  **Inner products of functions and vectors**  In this section, we will make use of inner products and norms of functions. In particular, we will use the common Hermitian inner product for functions f(x) and g(x) deﬁned for x on a domain x ∈ [a, b]: <f(x),g(x )> = --------------🡪 (2.1)  where ¯ g denotes the complex conjugate. The inner product of functions may seem strange or unmotivated at ﬁrst, but this deﬁnition becomes clear when we consider the inner product of vectors of data. If we discretize the functions f(x) and g(x) into vectors of data, as in we would like the vector inner product to converge to the function inner product as the sampling resolution is increased. The inner product of the data vectors f = [f1 f2 ··· fn]T and g =[g1 g2 ··· gn]T is  deﬁned by:  <f,g> = g∗f = ∑nk=1 fkgk= ∑ nk=1f(xk) g(xk).-------------------🡪 (2.2)  The magnitude of this inner product will grow as more data points are added; i.e., as n increases. Thus, we may normalize by ∆x = (b−a)/(n−1):  b−a /n−1 <f,g> = ∑n  k=1 f(xk)¯ g(xk)∆x, ---------------------🡪(2.3) whichistheRiemannapproximationtothecontinuousfunctioninnerproduct. It is now clear that as we take the limit of n → ∞ (i.e., inﬁnite data resolution, with ∆x → 0), the vector inner product converges to the inner product of functions in (2.1).  This inner product also induces a norm on functions, given by  ||f||2 = (<f,f>)1/2 = =( )1/2 .-------------------🡪 (2.4)  The set of all functions with bounded norm deﬁne the set of square integrable functions, denoted by L2([a,b]); this is also known as the set of Lebesgue integrable functions. The interval [a,b] may also be chosen to be inﬁnite (e.g., (−∞,∞)), semi-inﬁnite (e.g., [a,∞)), or periodic (e.g., [−π,π)). A fun example of a function in L2([1,∞)) is f(x) = 1/x. The square of f has ﬁnite integral from 1 to ∞, although the integral of the function itself diverges. The shape obtained by rotating this function about the x-axis is known as Gabriel’s horn, as the volume is ﬁnite (related to the integral of f2), while the surface area is inﬁnite (related to the integral of f). As in ﬁnite-dimensional vector spaces, the inner product may be used to project a function into an new coordinate system deﬁned by a basis of orthogonal functions. A Fourier series representation of a function f is precisely a projection of this function onto the orthogonal set of sine and cosine functions with integer period on the domain [a,b]. This is the subject of the following sections.  **Fourier series**  f(x) =a0/ 2+∑∞ k=1(ak cos(kx) + bk sin(kx))  Thus, the functions ψk = eikx for k ∈ Z (i.e., for integer k) provide a basis for periodic, complex-valued functions on an interval [0,2π). It is simple to see that these functions are orthogonal:    Thecoefﬁcientsaregivenbyck = 1 2πhf(x),ψk(x)i. Thefactorof1/2π normalizestheprojectionbythesquareofthenormof ψk;i.e.,kψkk2 = 2π. Thisisconsistent withourstandardﬁnite-dimensionalnotionofchangeofbasis,asinFig.2.2. A vector \* f maybewritteninthe (\* x, \* y) or (\* u,\* v) coordinatesystems,viaprojection onto these orthogonal bases:  A picture containing object, clock  Description automatically generated   * Fourier series approximation to a hat function.   % Define domain  dx = 0.001; L = pi;  x = (-1+dx:dx:1)\*L;  n = length(x); nquart = floor(n/4);  % Define hat function  f = 0\*x; f(nquart:2\*nquart) = 4\*(1:nquart+1)/n;  f(2\*nquart+1:3\*nquart) = 1-4\*(0:nquart-1)/n;  plot(x,f,’-k’,’LineWidth’,1.5), hold on  % Compute Fourier series  CC = jet(20);  A0 = sum(f.\*ones(size(x)))\*dx;  fFS = A0/2;  for k=1:20  A(k) = sum(f.\*cos(pi\*k\*x/L))\*dx; % Inner product  B(k) = sum(f.\*sin(pi\*k\*x/L))\*dx;  fFS = fFS + A(k)\*cos(k\*pi\*x/L) + B(k)\*sin(k\*pi\*x/L);  plot(x,fFS,’-’,’Color’,CC(k,:),’LineWidth’,1.2)  end   * The truncated Fourier series is plagued by ringing oscillations, known as Gibbs phenomena, around the sharp corners of the step function. This example highlights the challenge of applying the Fourier series to discontinuous functions:   dx = 0.01; L = 10;  x = 0:dx:L;  n = length(x); nquart = floor(n/4);  f = zeros(size(x)); f(nquart:3\*nquart) = 1;  A0 = sum(f.\*ones(size(x)))\*dx\*2/L;  fFS = A0/2;  for k=1:100  Ak = sum(f.\*cos(2\*pi\*k\*x/L))\*dx\*2/L;  Bk = sum(f.\*sin(2\*pi\*k\*x/L))\*dx\*2/L;  fFS = fFS + Ak\*cos(2\*k\*pi\*x/L) + Bk\*sin(2\*k\*pi\*x/L);  end  plot(x,f,’k’,’LineWidth’,2), hold on  plot(x,fFS,’r-’,’LineWidth’,1.2)  **A screenshot of a cell phone  Description automatically generated** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date:** | **23 May 2020** | **Name:** | **Srinidhi J C** | |
| **Course:** | **Python** | **USN:** | **4al16ec078** | |
| **Topic:** | |  | | --- | | **Fixing Programming Errors** | | **Application 3: Build a Website Blocker** | | **Semester & Section:** | **8Th -Sem, B-Sec** | |
| **AFTERNOON SESSION DETAILS** | | | |
|  | | | |
| **Report –**  Common error types in python.  **Index Error** is thrown when trying to access an item at an invalid index.  >>> L1=[1,2,3] >>> L1[3] Traceback (most recent call last): File "<pyshell#18>", line 1, in <module>  L1[3] IndexError: list index out of range  **ModuleNotFoundError** is thrown when a module could not be found.  >>> import notamodule Traceback (most recent call last): File "<pyshell#10>", line 1, in <module>  import notamodule ModuleNotFoundError: No module named 'notamodule'  **KeyError** is thrown when a key is not found.  >>> D1={'1':"aa", '2':"bb", '3':"cc"} >>> D1['4'] Traceback (most recent call last): File "<pyshell#15>", line 1, in <module>  D1['4'] KeyError: '4'  **ImportError** is thrown when a specified function can not be found.  >>> from math import cube Traceback (most recent call last): File "<pyshell#16>", line 1, in <module>  from math import cube ImportError: cannot import name 'cube'  **StopIteration** is thrown when the next() function goes beyond the iterator items.  >>> it=iter([1,2,3]) >>> next(it) 1 >>> next(it) 2 >>> next(it) 3 >>> next(it) Traceback (most recent call last): File "<pyshell#23>", line 1, in <module>  next(it) StopIteration  **TypeError** is thrown when an operation or function is applied to an object of an inappropriate type.  >>> '2'+2 Traceback (most recent call last): File "<pyshell#23>", line 1, in <module>  '2'+2 TypeError: must be str, not int  **ValueError** is thrown when a function's argument is of an inappropriate type.  >>> int('xyz') Traceback (most recent call last): File "<pyshell#14>", line 1, in <module>  int('xyz') ValueError: invalid literal for int() with base 10: 'xyz'  **NameError** is thrown when an object could not be found.  >>> age Traceback (most recent call last): File "<pyshell#6>", line 1, in <module>  age NameError: name 'age' is not defined  **ZeroDivisionError** is thrown when the second operator in the division is zero.  >>> x=100/0 Traceback (most recent call last): File "<pyshell#8>", line 1, in <module>  x=100/0 ZeroDivisionError: division by zero  **KeyboardInterrupt** is thrown when the user hits the interrupt key (normally Control-C) during the execution of the program.  >>> name=input('enter your name') enter your name^c Traceback (most recent call last): File "<pyshell#9>", line 1, in <module>  name=input('enter your name') KeyboardInterrupt  >>> lines  = ["trees are good", "pool is fresh", "face is round"]  >>> website\_list = ["face", "clock", "trend"]  >>> for line in lines:  ...     any(website in line for website in website\_list)  ...  False  False  True  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")  ("clock" in "trees are good")  ("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 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:  from datetime import datetime  with open(datetime.now().strftime("%Y-%m-%d-%H-%M-%S"), "w") as myfile:  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. | | | |