**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **26-05-2020** | **Name:** | **Karthik J** |
| **Course:** | **DSP** | **USN:** | **4AL16EC030** |
| **Topic:** | Fourier Series | **Semester & Section:** | **8TH A** |
| **GitHub Repository:** | Karthik-J |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session**      **Fourier and Wavelet Transforms**  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 modernworld than any other algorithm to date.  With increasingly complex problems, data sets, and computational geometries, simple Fourier sine and cosine bases have given way to tailored bases, such as the data-driven SVD. In fact, the SVD basis can be used as a direct analogue of the Fourier basis for solving PDEs with complex geometries, as will be discussed later. In addition, related functions, called wavelets, have been developed for advanced signal processing and compression efforts. In this chapter, we will demonstrate a few of the many uses of Fourier and wavelet transforms.  **Fourier transforms**  Before describing the computational implementation of Fourier transforms on vectors of data, here we introduce the analytic Fourier series and Fourier trans-form, defined for continuous functions. Naturally, 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.  The Fourier series is defined for periodic functions, so that outside the domain of definition, the function repeats itself forever. The Fourier transform integral is essentially the limit of a Fourier series as the length of the domain goes to infinity, which allows us to define a function defined on (−∞,∞) without repeating  Inner products of functions and vectors  Hermitian inner product for functions f(x)and g(x)defined for x on a domain x∈[a,b]: f(x),g(x)=∫baf(x) g(x)dx(2.1)where ̄g denotes the complex conjugate.  f(x) = F −1 ˆf(ω) = 1 2π Z ∞ −∞ ˆf(ω)e iωx dω (2.18a) ˆf(ω) = F (f(x)) = Z ∞ −∞ f(x)e −iωx dx.  Fourier series  A fundamental result in Fourier analysis is that iff(x)is periodic and piece wise smooth, then it can be written in terms of a Fourier series, which is an infinite sum of cosines and sines of increasing frequency Fourier series approximation to a hat function 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),holdon % 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  output    Fourier series for a discontinuous hat function  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),holdonplot(x,fFS,'r-','LineWidth',1.2)  output |
|  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Date:** | | **26-05-2020** | **Name:** | **Karthik J** |  |
| **Course:** | | [Programming with Python: Hands-on Introduction for Beginners](https://www.udemy.com/course/python-programming-beginners/) | **USN:** | **4AL16EC030** |  |
| **Topic:** | |  | **Semester & Section:** | **8th A** |  |
|  | **AFTERNOON SESSION DETAILS** | | | | |
|  | **Image of session** | | | | |
|  | Python - Lists The most basic data structure in Python is the **sequence**. Each element of a sequence is assigned a number - its position or index. The first index is zero, the second index is one, and so forth.  Python has six built-in types of sequences, but the most common ones are lists and tuples, which we would see in this tutorial.  There are certain things you can do with all sequence types. These operations include indexing, slicing, adding, multiplying, and checking for membership. In addition, Python has built-in functions for finding the length of a sequence and for finding its largest and smallest elements. Python Lists The list is a most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.  Creating a list is as simple as putting different comma-separated values between square brackets. For example −  list1 = ['physics', 'chemistry', 1997, 2000];  list2 = [1, 2, 3, 4, 5 ];  list3 = ["a", "b", "c", "d"]  Similar to string indices, list indices start at 0, and lists can be sliced, concatenated and so on Accessing Values in Lists To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index.  **Example**  list1 = ['physics', 'chemistry', 1997, 2000];  list2 = [1, 2, 3, 4, 5, 6, 7 ];  print "list1[0]: ", list1[0]  print "list2[1:5]: ", list2[1:5] Updating Lists You can update single or multiple elements of lists by giving the slice on the left-hand side of the assignment operator, and you can add to elements in a list with the append() method.  **Example**  list = ['physics', 'chemistry', 1997, 2000];  print "Value available at index 2 : "  print list[2]  list[2] = 2001;  print "New value available at index 2 : "  print list[2] Delete List Elements To remove a list element, you can use either the del statement if you know exactly which element(s) you are deleting or the remove() method if you do not know.  **Example**  list1 = ['physics', 'chemistry', 1997, 2000];  print list1  del list1[2];  print "After deleting value at index 2 : "  print list1 Basic List Operations Lists respond to the + and \* operators much like strings; they mean concatenation and repetition here too, except that the result is a new list, not a string. | | | | |