**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **27-05-2020** | **Name:** | **Karthik J** |
| **Course:** | **DSP** | **USN:** | **4AL16EC030** |
| **Topic:** | Fourier Series | **Semester & Section:** | **8TH A** |
| **GitHub Repository:** | Karthik-J |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session**     The Fast Fourier Transform How the FFT works  The FFT is a complicated algorithm, and its details are usually left to those that specialize in such things. This section describes the general operation of the FFT, but skirts a key issue: the use of *complex numbers*. If you have a background in complex mathematics, you can read between the lines to understand the true nature of the algorithm. Don't worry if the details elude you; few scientists and engineers that use the FFT could write the program from scratch.  In complex notation, the time and frequency domains each contain *one signal* made up of *N* *complex points*. Each of these complex points is composed of two numbers, the real part and the imaginary part. For example, when we talk about complex sample *X*[42], it refers to the combination of *ReX*[42] and *ImX*[42]. In other words, each complex variable holds two numbers. When two complex variables are multiplied, the four individual components must be combined to form the two components of the product (such as in Eq. 9-1). The following discussion on *"How the FFT works"* uses this jargon of complex notation. That is, the singular terms: *signal, point, sample*, and *value*, refer to the *combination* of the real part and the imaginary part.  The FFT operates by decomposing an *N* point time domain signal into *N* time domain signals each composed of a single point. The second step is to calculate the *N* frequency spectra corresponding to these *N* time domain signals. Lastly, the *N* spectra are synthesized into a single frequency spectrum.  separate stages. The first stage breaks the 16-point signal into two signals each consisting of 8 points. The second stage decomposes the data into four signals of 4 points. This pattern continues until there are *N* signals composed of a single point. An interlaced decomposition is used each time a signal is broken in two, that is, the signal is separated into its even and odd numbered samples. The best way to understand this is by inspecting Fig. 12-2 until you grasp the pattern. There are *Log*2*N* stages required in this decomposition, i.e., a 16 point signal (24) requires 4 stages, a 512 point signal (27) requires 7 stages, a 4096 point signal (212) requires 12 stages, etc. Remember this value, *Log*2*N*; it will be referenced many times in this chapter.  Introduction to WT  MATLAB includes built-in mathematical functions fundamental to solving engineering and scientific problems, and an interactive environment ideal for iterative exploration, design, and problem solving. Through product demonstrations, you will see how this combination allows you to quickly explore ideas, gain insight into your data, and document and share your results.  CWT & DWT  The wavelet transform is a relatively new concept (about 10 years old), but yet there are quite a few articles and books written on them. However, most of these books and articles are written by math people, for the other math people; still most of the math people don't know what the other math people are talking about (a math professor of mine made this confession). In other words, majority of the literature available on wavelet transforms are of little help, if any, to those who are new to this subject (this is my personal opinion). When I first started working on wavelet transforms, I have struggled for many hours and days to figure out what was going on in this mysterious world of wavelet transforms, due to the lack of introductory level text(s) in this subject. Therefore, I have decided to write this tutorial for the ones who are new to the topic. I consider myself quite new to the subject too, and I have to confess that I have not figured out all the theoretical details yet.  Welch's method and windowing  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) 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) 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. |
|  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Date:** | | **27-05-2020** | **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 - Tuples A tuple is an immutable sequence of Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.  Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses also.  For example −  tup1 = ('physics', 'chemistry', 1997, 2000);  tup2 = (1, 2, 3, 4, 5 );  tup3 = "a", "b", "c", "d"; Accessing Values in Tuples To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index.  For example −  tup1 = ('physics', 'chemistry', 1997, 2000);  tup2 = (1, 2, 3, 4, 5, 6, 7 );  print "tup1[0]: ", tup1[0];  print "tup2[1:5]: ", tup2[1:5]; Updating Tuples Tuples are immutable which means you cannot update or change the values of tuple elements. You are able to take portions of existing tuples to create new tuples  Example  tup1 = (12, 34.56);  tup2 = ('abc', 'xyz');  # Following action is not valid for tuples  # tup1[0] = 100;  # So, let's create a new tuple as follows  tup3 = tup1 + tup2;  print tup3; Delete Tuple Elements Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.  To explicitly remove an entire tuple, just use the **del** statement.  example −  tup = ('physics', 'chemistry', 1997, 2000);  print tup;  del tup;  print "After deleting tup : ";  print tup;  This produces the following result. Note an exception raised, this is because after **del tup** tuple does not exist any-more − Basic Tuples Operations Tuples respond to the + and \* operators much like strings; they mean concatenation and repetition here too, except that the result is a new tuple, not a string. Indexing, Slicing, and Matrixes Because tuples are sequences, indexing and slicing work the same way for tuples as they do for strings. Assuming following input −  L = ('spam', 'Spam', 'SPAM!') 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] | | | | |