**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **27-May-2020** | **Name:** | **Raziya Banu** |
| **Course:** | **DSP** | **USN:** | **4AL16EC058** |
| **Topic:** | **Fourier Transforms** | **Semester & Section:** | **8th sem & ‘B’ section** |
| **Github Repository:** |  |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session** |
| **Report –**  In my first session today I have studied about the DSP Fourier transform.  **Introduction**  The frequency analysis is the one of the most popular methods in signal processing. It is a tool for signal decomposition for further filtration, which is in fact separation of signal components from each other.  Although, the process of crossing the border between these two worlds (time and frequency domain) may be considered as advanced case, it is worth doing it, as this other world gives us new opportunities and simplifies many issues.  The article presents implementation of the various versions of calculating Discrete Fourier Transform, starting with definition of Fourier Transform, by reduced calculation algorithm and finishing with Cooley-Tukey method of Fast Fourier Transform.  In addition, it presents the importance of the simplest operations performed on the signal spectrum and their impact on the time domain.  **Signals**  Signal can be defined as a variability of any physical value, that can be described as a function of a single or multiple arguments. In this article we will be interested in one-dimensional time functions.  In real world, time functions that can be met are placed in continuous domain. However, the development of computer science, caused that analog signal processing became rare. It is much more cost-effective to create, implement and test signal processing algorithms in digital world, then to project and develop analog (electronic) devices.  **From continuous to discrete domain**  In order to receive digital representation of analog signal it needs to be turned into discrete-time domain and quantized.  The Nyquist-Shannon sampling theorem is the link between continuous-time signals and discrete-time signals. This theorem can be also known as The Whittaker-Nyquist-Kotielnikov-Shannon sampling theorem – the choice of the authors names depending on the country in which we talk about this issue. To be above this, we will call it simply the sampling theorem.  The sampling theorem answers the question of how to sample a continuous-time signal to obtain a discrete-time signal, from which you can restore original (continuous-time) signal. According to this statement to obtain a properly sampled discrete-time signal, the sampling frequency must be at least twice of highest frequency that can be observed in original signal.  **Signal decomposition into frequency domain**  It all started in 1807 when the French mathematician and physicist, Joseph Fourier, introduced the trigonometric series decomposition (nowadays know as Fourier series method) to solve the partial differential heat equation in the metal plate. Fourier's idea was to decomposed complicated periodic function into to sum of the simplest oscillating functions - sines ans cosines.  If the function f(x) is periodic with period of T and intergrable (its integral is finite) on an interval [x0, x0 + T], then it can be transformed into series:  f(x)=a02+∑n=1∞(an⋅cos(2nπTx)+bn⋅sin(2nπTx))f(x)=a02+∑n=1∞(an⋅cos(2nπTx)+bn⋅sin(2nπTx))  where  an=2T∫x0+Tx0f(x)⋅cos(2nπTx)dxbn=2T∫x0+Tx0f(x)⋅sin(2nπTx)dxan=2T∫x0x0+Tf(x)⋅cos(2nπTx)dxbn=2T∫x0x0+Tf(x)⋅sin(2nπTx)dx    **Discrete Fourier Transform**  Definition  Fourier series can be named a progenitor of Fourier Transform, which, in case of digital signals (Discrete Fourier Transform), is described with formula:  X(k)=1N∑n=0N−1x(n)⋅e−j2πNknX(k)=1N∑n=0N−1x(n)⋅e−j2πNkn    Fourier transformation is reversible and we can return to time domain by calculation:    x(n)=∑k=0N−1X(k)⋅ej2πNknx(n)=∑k=0N−1X(k)⋅ej2πNkn    In some notations we can observe that divide by N is transterred to inverse calculation - it does not disrupt the calculation unless we apply divide by N both in forward and inverse Fourier Transform.    Forward and inverse Discrete Fourier Transform implementations are shown below.  /// <summary>  /// Calculates Discrete Fourier Transform of given digital signal x  /// </summary>  /// <param name="x">Signal x samples values</param>  /// <returns>Fourier Transform of signal x</returns>  public Complex[] DFT(Double[] x)  {      int N = x.Length; // Number of samples      Complex[] X = new Complex[N];  for (int k = 0; k < N; k++)      {          X[k] = 0;          for (int n = 0; n < N; n++)          {              X[k] += x[n] \* Complex.Exp(-Complex.ImaginaryOne \* 2 \* Math.PI \* (k \* n) / Convert.ToDouble(N));          }          X[k] = X[k] / N;      }      return X;  }  /// <summary>  /// Calculates inverse Discrete Fourier Transform of given spectrum X  /// </summary>  /// <param name="X">Spectrum complex values</param>  /// <returns>Signal samples in time domain</returns>  public Double[] iDFT(Complex[] X)  {  int N = X.Length; // Number of spectrum elements  Double[] x = new Double[N];  for (int n = 0; n < N; n++)  {  Complex sum = 0;  for (int k = 0; k < N; k++)  {  sum += X[k] \* Complex.Exp(Complex.ImaginaryOne \* 2 \* Math.PI \* (k \* n) / Convert.ToDouble(N));  }  x[n] = sum.Real; // As a result we expect only real values (if our calculations are correct imaginary values should be equal or close to zero)  }  return x;}    The mistake often committed in code implementation of inverse Fourier Transform is to transfer complex value to real by using its magnitude value - in such a case we will receive |x(n)| instead of x(n). |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date:** | **27-May-2020** | **Name:** | **Raziya Banu** | |
| **Course:** | **Udemy** | **USN:** | **4AL16EC058** | |
| **Topic:** | **Append Lists in Python** | **Semester & Section:** | **8th sem & ‘B’ section** | |
| **AFTERNOON SESSION DETAILS** | | | |
| **Image of session** | | | |
| **Python List append, insert, extend etc…** In Python, use list methods append(), extend(), and insert() to add items to a list or combine other lists. You can also use the + operator to combine lists, or use slices to insert itemss at specific positions.   * Add an item to the end: append() * Combine lists: extend(), + operator * Add an item at specified index: insert() * Add another list or tuple at specified index: slice   Sponsored Link  **Add an item to the end: append()**  You can add an item to the end of the list with append(). If you want to add to positions other than the end, such as the beginning, use insert() described later.  l = list(range(3))  print(l)  # [0, 1, 2]  l.append(100)  print(l)  # [0, 1, 2, 100]  l.append('new')  print(l)  # [0, 1, 2, 100, 'new']  A list is also added as one item, not combined.  l.append([3, 4, 5])  print(l)  # [0, 1, 2, 100, 'new', [3, 4, 5]]  **Combine lists: extend()**  You can combine another list or tuple at the end with extend(). All itemss are added to the end of the original list.  l = list(range(3))  print(l)  # [0, 1, 2]  l.extend([100, 101, 102])  print(l)  # [0, 1, 2, 100, 101, 102]  l.extend((-1, -2, -3))  print(l)  # [0, 1, 2, 100, 101, 102, -1, -2, -3]  In the case of a string, note that each character is added one by one.  l.extend('new')  print(l)  # [0, 1, 2, 100, 101, 102, -1, -2, -3, 'n', 'e', 'w']  It is also possible to combine using the + operator instead of extend().  In the case of the + operator, a new list is returned. You can also add to the existing list with +=.  l2 = l + [5, 6, 7]  print(l2)  # [0, 1, 2, 100, 101, 102, -1, -2, -3, 'n', 'e', 'w', 5, 6, 7]  l += [5, 6, 7]  print(l)  # [0, 1, 2, 100, 101, 102, -1, -2, -3, 'n', 'e', 'w', 5, 6, 7]  **Add an item at specified index: insert()**  You can add an item at the specified index (position) by insert().  Set the index for the first parameter and the item to be inserted for the second parameter. The beginning is 0. For negative values, -1 means one before the end.  l = list(range(3))  print(l)  # [0, 1, 2]  l.insert(0, 100)  print(l)  # [100, 0, 1, 2]  l.insert(-1, 200)  print(l)  # [100, 0, 1, 200, 2]  Like append(), the list is added as a single item, not combined.  l.insert(0, [-1, -2, -3])  print(l)  # [[-1, -2, -3], 100, 0, 1, 200, 2]  **Add another list or tuple at specified index: slice**  If you specify a range using slice and assign another list or tuple, all itemss will be added.  l = list(range(3))  print(l)  # [0, 1, 2]  l[1:1] = [100, 200, 300]  print(l)  # [0, 100, 200, 300, 1, 2]  You can also replace the original item. All items in the specified range are replaced.  l = list(range(3))  print(l)  # [0, 1, 2]  l[1:2] = [100, 200, 300]  print(l)  # [0, 100, 200, 300, 2] | | | |