**DAILY ASSESSMENT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **27 may 2020** | **Name:** | **Shreya poojary** |
| **Course:** | DSP | **USN:** | **4al16ec074** |
| **Topic:** | FFT, FIR, IIR filters CWT & DWT, welchs method and windowing, ECG signals analysis using MATLAB | **Semester & Section:** | **8-B** |
| **Github Repository:** | **Shreya-test** |  |  |

|  |
| --- |
|  |
| **Report:-** Fourier Transforms: The Fourier transform of a function *f* is traditionally denoted {\displaystyle {\hat {f}}}, by adding a [circumflex](https://en.wikipedia.org/wiki/Circumflex) to the symbol of the function. There are several [common conventions](https://en.wikipedia.org/wiki/Fourier_transform#Other_conventions) for defining the Fourier transform of an [integrable](https://en.wikipedia.org/wiki/Lebesgue_integration) function,{\displaystyle f:\mathbb {R} \to \mathbb {C} }    The statement that *f* can be reconstructed from {\displaystyle {\hat {f}}} is known as the [Fourier inversion theorem](https://en.wikipedia.org/wiki/Fourier_inversion_theorem), and was first introduced in [Fourier's](https://en.wikipedia.org/wiki/Joseph_Fourier)  although what would be considered a proof by modern standards was not given until much later. The functions *f* and {\displaystyle {\hat {f}}}f^ often are referred to as a *Fourier integral pair* or *Fourier transform pair*   Fast Fourier Transform: Let *x*0, …, *xN*−1 be [complex numbers](https://en.wikipedia.org/wiki/Complex_number). The [DFT](https://en.wikipedia.org/wiki/Discrete_Fourier_transform) is defined by the formula    where {\displaystyle e^{i2\pi /N}} is a [primitive](https://en.wikipedia.org/wiki/Primitive_root_of_unity) *N*th root of 1.        This compositional viewpoint immediately provides the simplest and most common multidimensional DFT algorithm, known as the **row-column** algorithm (after the two-dimensional case, below). That is, one simply performs a sequence of *d* one-dimensional FFTs (by any of the above algorithms): first you transform along the *n*1 dimension, then along the *n*2 dimension, and so on (or actually, any ordering works). This method is easily shown to have the usual O(*N* log *N*) complexity, where {\displaystyle N=N\_{1}\cdot N\_{2}\cdot \cdots \cdot N\_{d}}N= N1, N2,…………Nd is the total number of data points transformed. In particular, there are *N*/*N*1 transforms of size *N*1, etcetera, so the complexity of the sequence of FFTs is:    A **fast Fourier transform** (**FFT**) is an [algorithm](https://en.wikipedia.org/wiki/Algorithm) that computes the [discrete Fourier transform](https://en.wikipedia.org/wiki/Discrete_Fourier_transform) (DFT) of a sequence, or its inverse (IDFT). [Fourier analysis](https://en.wikipedia.org/wiki/Fourier_analysis) converts a signal from its original domain (often time or space) to a representation in the [frequency domain](https://en.wikipedia.org/wiki/Frequency_domain) and vice versa. The DFT is obtained by decomposing a [sequence](https://en.wikipedia.org/wiki/Sequence) of values into components of different frequencies.[[1]](https://en.wikipedia.org/wiki/Fast_Fourier_transform#cite_note-Heideman_Johnson_Burrus_1984-1) This operation is useful in many fields, but computing it directly from the definition is often too slow to be practical.    **wee.PNG** Simple and Easy Tutorial on FFT Fast Fourier Transform Matlab Fs = 1000;  T = 1/Fs;  L = 1500;  t = (0:L-1)\*T;  S = 0.7\*sin(2\*pi\*50\*t) + sin(2\*pi\*120\*t);  X = S + 2\*randn(size(t));  plot(1000\*t(1:50),X(1:50))  title('Signal Corrupted with Zero-Mean Random Noise')  xlabel('t (milliseconds)')  ylabel('X(t)')  Y = fft(X);  P2 = abs(Y/L);  P1 = P2(1:L/2+1);  P1(2:end-1) = 2\*P1(2:end-1);  f = Fs\*(0:(L/2))/L;  plot(f,P1)  title('Single-Sided Amplitude Spectrum of X(t)')  xlabel('f (Hz)')  ylabel('|P1(f)|')  Y = fft(S);  P2 = abs(Y/L);  P1 = P2(1:L/2+1);  P1(2:end-1) = 2\*P1(2:end-1);  plot(f,P1)  title('Single-Sided Amplitude Spectrum of S(t)')  xlabel('f (Hz)')  ylabel('|P1(f)|')  Fs = 100;  t = -0.5:1/Fs:0.5;  L = length(t);  X = 1/(4\*sqrt(2\*pi\*0.01))\*(exp(-t.^2/(2\*0.01)));  plot(t,X)  title('Gaussian Pulse in Time Domain')  xlabel('Time (t)')  ylabel('X(t)')  n = 2^nextpow2(L);  Y = fft(X,n);  f = Fs\*(0:(n/2))/n;  P = abs(Y/n);  plot(f,P(1:n/2+1))  title('Gaussian Pulse in Frequency Domain')  xlabel('Frequency (f)')  ylabel('|P(f)|')   Easy Introduction to Wavelets Wavelets, in general, are constructed by taking the dilations and translations of a single function with sufficient decay in both the time and frequency domains. The definition adopted here for “sufficient” decay is that a function Y(x) and its Fourier transform, denoted by Y(f) , both decay faster than I x I- 1 and I f I - r, respectively;      In most situations it is useful to restrict ψ to be a continuous function with a higher number *M* of vanishing moments, i.e. for all integer *m* < *M*         Simple audio denoising using wavelet decomposition and thresholding, wavelet denoising[ MATLAB ] rng default;  [X,XN] = wnoise('bumps',10,sqrt(6));  subplot(211)  plot(X); title('Original Signal');  AX = gca;  AX.YLim = [0 12];  subplot(212)  plot(XN); title('Noisy Signal');  AX = gca;  AX.YLim = [0 12];  xd = wdenoise(XN,4);  figure;  plot(X,'r')  hold on;  plot(xd)  legend('Original Signal','Denoised Signal','Location','NorthEastOutside')  axis tight;  hold off;  xdMODWT = wden(XN,'modwtsqtwolog','s','mln',4,'sym4');  figure;  plot(X,'r')  hold on;  plot(xdMODWT)  legend('Original Signal','Denoised Signal','Location','NorthEastOutside')  axis tight;  hold off;     Short-time Fourier Transform and the Spectogram The **Short-time Fourier transform** (**STFT**), is a [Fourier-related transform](https://en.wikipedia.org/wiki/List_of_Fourier-related_transforms) used to determine the sinusoidal frequency and phase content of local sections of a signal as it changes over time.[[1]](https://en.wikipedia.org/wiki/Short-time_Fourier_transform#cite_note-1) In practice, the procedure for computing STFTs is to divide a longer time signal into shorter segments of equal length and then compute the Fourier transform separately on each shorter segment. This reveals the Fourier spectrum on each shorter segment. One then usually plots the changing spectra as a function of time, known as a [spectrogram](https://en.wikipedia.org/wiki/Spectrogram) or waterfall plot.  If the DFT coefficients of each frame are placed into a separate column of a matrix, the STFT can be represented as a matrix of coefficients, where the column index represents time and the row index is associated with the frequency of the respective DFT coefficient. If the magnitude of each coefficient is computed, the resulting matrix can be treated as an image and, as a result, it can be visualized. This image is known as the [spectrogram](https://www.sciencedirect.com/topics/engineering/spectrogram) of the signal and presents the evolution of the signal in the time-frequency domain. To generate the spectrogram, we can use the magnitude or the squared magnitude of the STFT coefficients on a linear or [logarithmic scale](https://www.sciencedirect.com/topics/engineering/logarithmic-scale) (dB). In MATLAB, the spectrogram of a signal is implemented in the spectrogram () function, which can plot the spectrogram and return the matrix of STFT coefficients, along with the respective time and frequency axes. In this book, we will mainly use the spectrogram as a visualization tool. The STFT coefficients will be extracted, when required, by means of a more general function that we have developed for short-term processing purposes.   Power Spectrum Estimation Examples: Welch's Method   **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.   ECG Signal Analysis Using MATLAB A real-time QRS detection algorithm, which references [1, lab one], [3] and [4], is developed in Simulink with the assumption that the sampling frequency of the input ECG signal is always 200 Hz (or 200 samples/s). However, the recorded real ECG data may have different sampling frequencies ranging from 200 Hz to 1000 Hz, e.g., 360 Hz in this example. To bridge the different sampling frequencies, a sample rate converter block is used to convert the sample rate to 200 Hz. A buffer block is inserted to ensure the length of the input ECG signal is a multiple of the calculated decimation factor of the sample-rate converter block.  The ECG signal is filtered to generate a windowed estimate of the energy in the QRS frequency band. The filtering operation has these steps:  **1.** FIR Bandpass filter with a pass band from 5 to 26 Hz  **2.** Taking the derivative of the bandpass filtered signal  **3.** Taking the absolute value of the signal  **4.** Averaging the absolute value over an 80 ms window  The QRS detection block detects peaks of the filtered ECG signal in real-time. The detection threshold is automatically adjusted based on the mean estimate of the average QRS peak and the average noise peak. The detected peak is classified as a QRS complex or as noise, depending on whether it is above the threshold.  The following QRS detection rules reference the PIC-based QRS detector implemented in [4].  **Rule 1.** Ignore all peaks that precede or follow larger peaks by less than 196 ms (306bpm).  **Rule 2.** If a peak occurs, check to see whether the raw signal contains both positive and negative slopes. If true, report a peak being found. Otherwise, the peak represents a baseline shift.  **Rule 3.** If the peak is larger than the detection threshold, classify it as a QRS complex. Otherwise classify it as noise.  **Rule 4.** If no QRS has been detected within 1.5 R-to-R intervals, but there is a peak that was larger than half the detection threshold, and that peak followed the preceding detection by at least 360ms, classify that peak as a QRS complex.  **1.** Open the [example model](matlab:ex_ecg_sigprocessing).  **2.** Change your current folder in MATLAB® to a writable folder.  **3.** On the model tool strip, click **Run** to start the simulation. Observe the **HeartRate** display and the raw and filtered ECG signal in the scope, which also illustrates the updating of peaks, threshold and estimated mean heart rate.  **4.** Open the dialog of **ECG Signal Selector** block. Select the ECG signal mean heart rate in the drop down menu. Click **Apply** and observe the real-time detection results in the scopes and **HeartRate** display.  **5.** Click **Stop** to end simulation.  **6.** After selecting target hardware, you can generate code from the **ECGSignalProcessing** subsystem and deploy it to the target.   |  |  |  |  | | --- | --- | --- | --- | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  | | | | | |