Mini Project Report

Course Title:

Discrete Time Signal Processing

Faculty Name:

Dr.Sabitha Ramakrishnan

Student Name:

P.Kabilan

Register No:

2018505517

Project Title:

Denoising Data signals using FFT

Date:

18/06/2021

Abstract:

The main objective of this project is that when we get any random noisy signal which has some information embedded into it we can differentiate the signals by calculating the power spectrum of the signal.This gives the individual powers of different frequencies and thus can be able to differentiate the original signal from the noisy signal.Then we can nullify the noisy frequency components to retrieve back the original signal.

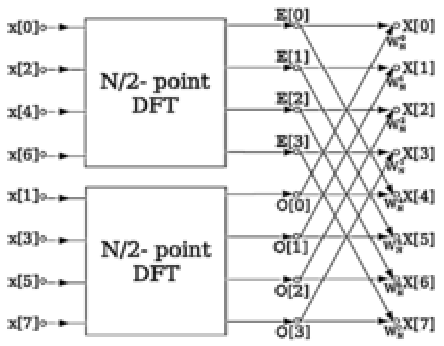
Software Used:

- Matlab (version-R2021a)

Theory:

Fast Fourier Transform:

The fast fourier transform is one of the most important algorithm being created in the past century by James.W.Cooley and John.W.Tukey.So much of the modern technology that we have today such as the wireless communication,gps,audio and image compression,satellite tv etc.The fft as you may all know as Fast Fourier Transform is not a transform but is an algorithm to compute discrete fourier transform in computing devices.FFT algorithms are speed and efficient to calculate.While the space complexity of discrete fourier transform is of the order O(n^2),the fast fourier transform can produce the same output of DFT with a space complexity of O(n*log(n)).This is way smaller and is linear and as the order of the matrix 'n' goes up higher and higher the value of log(n) would be not as important as 'n' where 'n'→Number of data points or the data size.This will be efficient while handling with very large data sets.



This is the FFT algorithm structure using a decomposition into half sized fft's.

Definition:

Let X_0, ………X_{N-1} be complex numbers.The DFT is defined by the formula

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N} \qquad k = 0, \dots, N-1,$$

where $e^{i2\pi/N}$ is a primitive Nth root of 1.

Evaluating this definition directly requires O(N^2) operations: there are N outputs Xk, and each output requires a sum of N terms. An FFT is any method to compute the same results in O(N*log N) operations. All known FFT algorithms require O(N*log N) operations, although there is no known proof that a lower complexity score is impossible.

To illustrate the savings of an FFT, consider the count of complex multiplications and additions for N=4096 data points. Evaluating the DFT's sums directly involves N^2 complex multiplications and N(N − 1) complex additions, of which O(N) operations can be saved by eliminating trivial operations such as multiplications by 1, leaving about 30 million operations. In contrast, the radix-2 Cooley–Tukey algorithm, for N a power of 2, can compute the same result with only $(N/2)\log_2(N)$ complex multiplications (again, ignoring simplifications of multiplications by 1 and similar) and $N \log_2(N)$ complex additions, in total about 30,000 operations – a thousand times less than with direct evaluation. In practice, actual performance on modern computers is usually dominated by factors other than the speed of arithmetic operations and the analysis is a complicated subject.

Cooley–Tukey algorithm:

By far the most commonly used FFT is the Cooley–Tukey algorithm. This is a divide-and-conquer algorithm that recursively breaks down a DFT of any composite size $N = N_1 N_2$ into many smaller DFTs of sizes N_1 and N_2 , along with O(N) multiplications by complex roots of unity traditionally called twiddle factors.

This method (and the general idea of an FFT) was popularized by a publication of Cooley and Tukey in 1965, but it was later discovered that those two authors had independently re-invented an algorithm known to Carl Friedrich Gauss around 1805 (and subsequently rediscovered several times in limited forms).

The best known use of the Cooley–Tukey algorithm is to divide the transform into two pieces of size N/2 at each step, and is therefore limited to power-of-two sizes, but any factorization can be used in general (as was known to both Gauss and Cooley/Tukey[1]). These are called the radix-2 and mixed-radix cases, respectively (and other variants such as the split-radix FFT have their own names as well). Although the basic idea is recursive, most traditional implementations rearrange the algorithm to avoid explicit recursion. Also, because the Cooley–Tukey algorithm breaks the DFT into smaller DFTs, it can be combined arbitrarily with any other algorithm for the DFT, such as those described below.

Applications of FFT:

The FFT is used in digital recording, sampling, additive synthesis and pitch correction software.
The FFT's importance derives from the fact that it has made working in the frequency domain equally computationally feasible as working in the temporal or spatial domain. Some of the important applications of the FFT include:

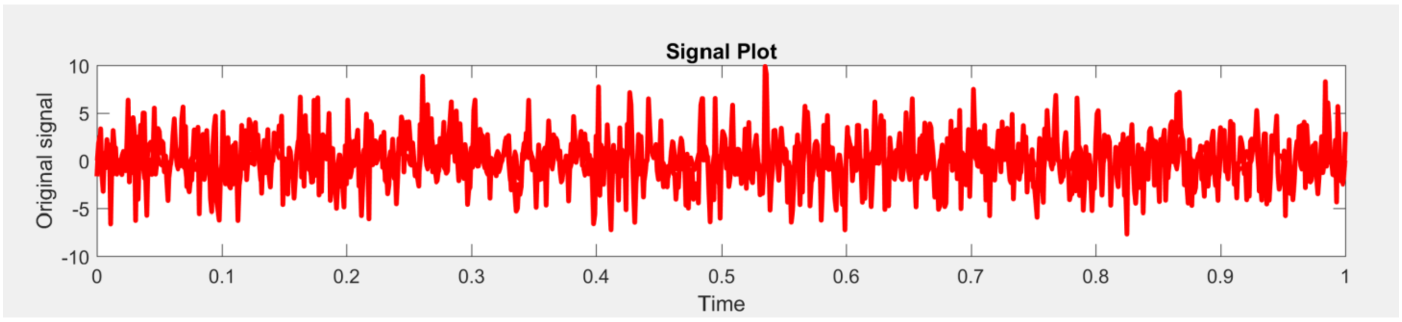
- fast large-integer and polynomial multiplication,
- efficient matrix–vector multiplication for Toeplitz, circulant and other structured matrices,
- filtering algorithms (see overlap–add and overlap–save methods),
- fast algorithms for discrete cosine or sine transforms (e.g. fast DCT used for JPEG and MPEG/MP3 encoding and decoding),
- fast Chebyshev approximation,
- solving difference equations,
- computation of isotopic distributions.
- modulation and demodulation of complex data symbols using orthogonal frequency division multiplexing (OFDM) for 5G, LTE, Wi-Fi, DSL, and other modern communication systems.

Code & Graph Explanation:

Below is the code used in matlab for denoising data signals,

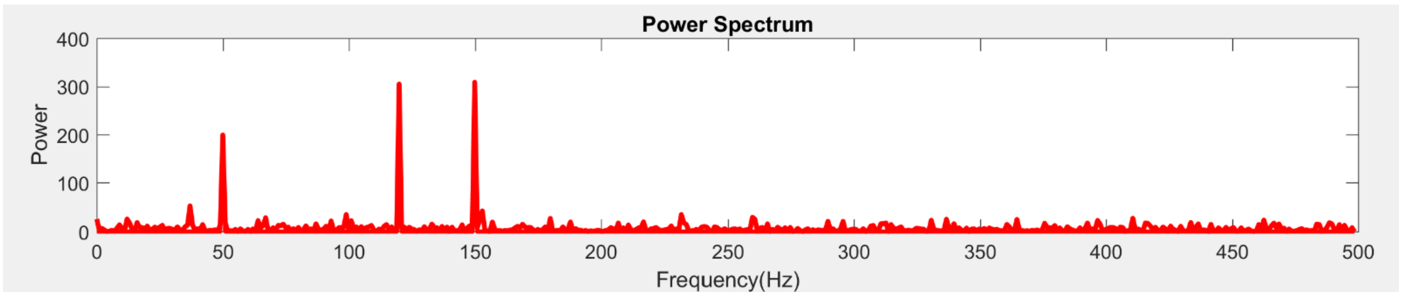
```
clc,clear all,close all
%creating a signal with three frequencies
dt=.001;
t=0:dt:1;
fclean=sin(2*pi*50*t)+sin(2*pi*120*t)+sin(2*pi*150*t);%sum of three frequencies
f=fclean+2.5*randn(size(t));%adding some noise

subplot(3,1,1)
plot(t,f,'c','LineWidth',3),hold on
plot(t,fclean,'Color',[.5 .1 0],'LineWidth',2.5)
xlabel('Time');
ylabel('Original signal');
title('Signal Plot');
l1=legend('Noisy','Clean');set(l1,'FontSize',12)
ylim([-10 10]); set(gca,'FontSize',12)

%%Computing the Fast Fourier Transform (FFT)
n=length(t);
fhat=fft(f,n);%Compute the fast fourier transform
PSD=fhat.*conj(fhat)/n;%Power spectrum calculation(power per frequency)
freq=1/(dt*n)*(0:n);%Create x-axis of frequencies in Hz
L=1:floor(n/2);%Only plot the first half of frequencies

subplot(3,1,2);set(gca,'FontSize',12)
plot(freq(L),PSD(L),'c','LineWidth',3),hold on
set(gca,'FontSize',12)

%%Use the PSD to filter out noise
indices=PSD>100;%find all frequencies with power greater than 100
PSDclean=PSD.*indices;%this zeroes out the freqs having power less than 100
fhat=indices.*fhat;%Zero out small fourier coeffs in Y
ffilt=ifft(fhat);%Inverse FFT for filterd time signal

plot(freq(L),PSDclean(L),'-','Color',[.5 .1 0],'LineWidth',2.5)
xlabel('Frequency(Hz)');
ylabel('Power');
title('Power Spectrum');
l1=legend('Noisy','Filtered');set(l1,'FontSize',12)
subplot(3,1,3);set(gca,'FontSize',12)
plot(t,ffilt,'-','Color',[.5 .1 0],'LineWidth',2.5)
xlabel('Time');
ylabel('Filtered signal');
title('Final plot');
l1=legend('Filtered');set(l1,'FontSize',12)
ylim([-10 10]);set(gca,'FontSize',12)
```

Firstly when we get a data signal like this,

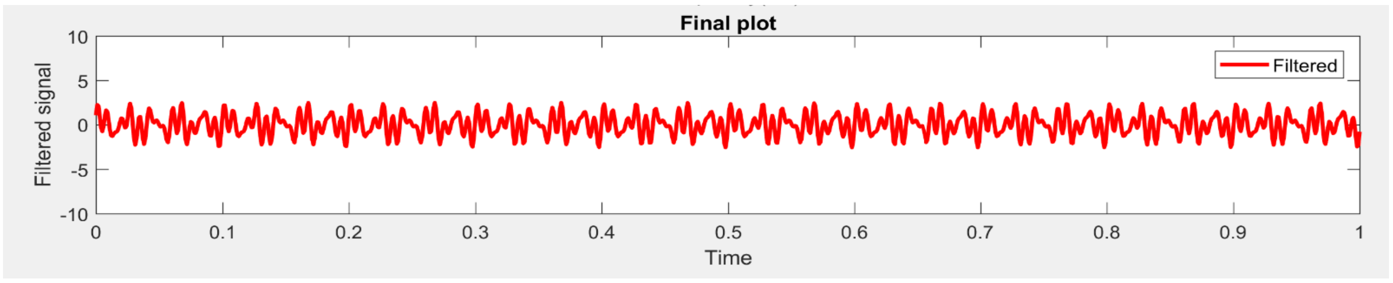


Don’t panic because we have FFT on our side to analyse its power spectrum and do appropriate calculations to retrieve the final signal.

In this code we have given a clean signal of sine waves at three different frequencies and also added a noise of amplitude 2.5.Due to this we get a signal like most real world scenarios do.Calculate the fast fourier transform and then check the power spectrum to find out where our clean signals frequencies lie.Below is the plot of power spectrum.



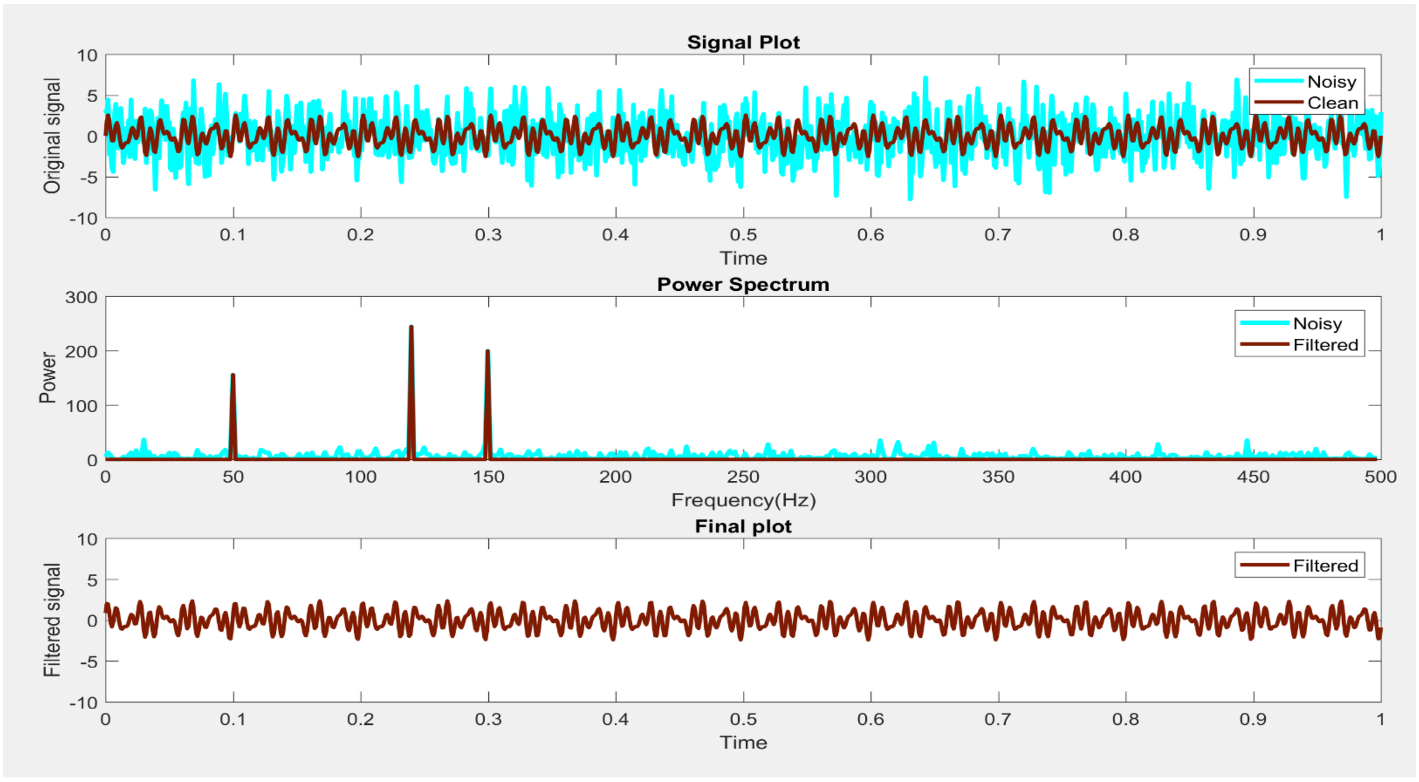
Here we see that there are three peaks at each of the three different frequencies.Even in real cases this type of plot with peaks will be retrieved so that we can eliminate the noise not completely but mostly.Therefore I can set a threshold in power at 100 and any frequency components that are larger than 100,I am going to keep it and rest are nullified.So I find the indices in power where the power is greater than 100 and find those corresponding frequencies in fft frequency components and perform inverse fast fourier transform over it to retrieve the original signal.Therefore after we did this our retrieved signal plot is,



Therefore this is our retrieved signal.You can play with any number of frequencies and also the main thing here is that to make sure to find the correct threshold in power spectrum so that we can get the retrieved signal completely.

Output Graph:

For demonstration purposes I have differentiated the clean signal and the whole signal to compare it with the final plot.



Result:

It is seen that the Fast Fourier Transform is one of the best and efficient algorithms created and is used for computing discrete fourier transform very efficiently and also with less space complexity.The final plot is almost retrieved when compared with the clean signal.