'Engineering style' explanation on Fast Fourier Transform (FFT).
If you already have the general idea then read the summary at the end.

Sampling using FT poses practical problems such as the need for ideal filters, which are unrealisable or are realisable only with infinite delays. So we move on to the FFT!

$X_r = \sum_{n=0}^{N_0-1} x_n e^{-jr\Omega_0 n}$

8.6 The Fast Fourier Transform (FFT)

<-- Linear Signals and Systems 2nd ed, by B.P. Lathi.

The number of computations required in performing the DFT was dramatically reduced by an algorithm developed by Cooley and Tukey in 1965.[5] This algorithm, known as the *fast Fourier transform* (FFT), reduces the number of computations from something on the order of N_0^2 to $N_0 \log N_0$. To compute one sample X_r from Eq. (8.22a), we require N_0 complex multiplications and $N_0 - 1$ complex additions. To compute N_0 such values (X_r for $r = 0, 1, \ldots, N_0 - 1$), we require a total of N_0^2 complex multiplications and $N_0(N_0 - 1)$ complex additions. For a large N_0, these computations can be prohibitively time-consuming, even for a high-speed computer. The FFT algorithm is what made the use of Fourier transform accessible for digital signal processing.

How Does FFT Reduce Number of Computations?

It is easy to understand the magic of FFT. The secret is in the linearity of the Fourier transform, and also of the DFT. Because of linearity, we can compute the Fourier transform of a signal $x(t)$ as a sum of the Fourier transforms of segments of $x(t)$ of shorter duration. The same principle applies to computation of DFT. Consider a signal of length $N_0 = 16$ samples. As seen earlier, DFT computation of this sequence requires $N_0^2 = 256$ multiplications and $N_0(N_0 - 1) = 240$ additions. We can split this sequence in two shorter sequences, each of length 8. To compute DFT of each of these segments, we need 64 multiplications and 56 additions. Thus, we need a total of 128 multiplications and 112 additions. Suppose, we split the original sequence in four segments of length 4 each. To compute the DFT of each segment, we require 16 multiplications and 12 additions. Hence, we need a total of 64 multiplications and 48 additions. If we split the sequence in eight segments of length 2 each, we need 4 multiplications and 2 additions for each segment, resulting in a total of 32 multiplications and 8 additions. Thus, we have been able to reduce the number of multiplications from 256 to 32 and the number of additions from 240 to 8. Moreover, some of these multiplications turn out to be multiplications by 1 or -1. All this fantastic economy in number of computations is realized by the FFT without any approximation! The values obtained by the FFT are identical to those obtained by DFT. In this example, we considered a relatively small value of $N_0 = 16$. The reduction in number of computations is much more dramatic for higher values of N_0.

The FFT algorithm is simplified if we choose N_0 to be a power of 2, although such a choice is not essential. For convenience, we define

$\Omega_0 = \frac{2\pi}{N_0}$

$$W_{N_0} = e^{-(j2\pi/N_0)} = e^{-j\Omega_0} \tag{8.37}$$

so that

$X_r = \sum_{n_0} x_n e^{-jr\Omega_0 n}$

$$X_r = \sum_{n=0}^{N_0-1} x_n W_{N_0}^{nr} \qquad 0 \le r \le N_0 - 1 \tag{8.38a}$$

and

$W_{N_0} = e^{-j\Omega_0}$

$$x_n = \frac{1}{N_0} \sum_{r=0}^{N_0-1} X_r W_{N_0}^{-nr} \qquad 0 \le n \le N_0 - 1 \tag{8.38b}$$

Although there are many variations of the Tukey–Cooley algorithm, these can be grouped into two basic types: *decimation in time* and *decimation in frequency*.

The Decimation-in-Time Algorithm

Here we divide the N_0-point data sequence into two $(N_0/2)$-point sequences consisting of even- and odd-numbered samples, respectively, as follows:

$$\underbrace{x_0, x_2, x_4, \ldots, x_{N_0-2}}_{\text{sequence } g_n}, \underbrace{x_1, x_3, x_5, \ldots, x_{N_0-1}}_{\text{sequence } h_n}$$

Then, from Eq. (8.38a),

$$X_r = \sum_{n=0}^{(N_0/2)-1} x_{2n} W_{N_0}^{2nr} + \sum_{n=0}^{(N_0/2)-1} x_{2n+1} W_{N_0}^{(2n+1)r} \tag{8.39}$$

Also, since

$$W_{N_0/2} = W_{N_0}^2 \tag{8.40}$$

we have

$$\begin{aligned} X_r &= \sum_{n=0}^{(N_0/2)-1} x_{2n} W_{N_0/2}^{nr} + W_{N_0}^r \sum_{n=0}^{(N_0/2)-1} x_{2n+1} W_{N_0/2}^{nr} \\ &= G_r + W_{N_0}^r H_r \qquad 0 \le r \le N_0 - 1 \end{aligned} \tag{8.41}$$

where G_r and H_r are the $(N_0/2)$-point DFTs of the even- and odd-numbered sequences, g_n and h_n, respectively. Also, G_r and H_r, being the $(N_0/2)$-point DFTs, are $(N_0/2)$ periodic. Hence

$$\begin{aligned} G_{r+(N_0/2)} &= G_r \\ H_{r+(N_0/2)} &= H_r \end{aligned} \tag{8.42}$$

Moreover,

$$W_{N_0}^{r+(N_0/2)} = W_{N_0}^{N_0/2} W_{N_0}^r = e^{-j\pi} W_{N_0}^r = -W_{N_0}^r \tag{8.43}$$

From Eqs. (8.41), (8.42), and (8.43), we obtain

$$X_{r+(N_0/2)} = G_r - W_{N_0}^r H_r \tag{8.44}$$

This property can be used to reduce the number of computations. We can compute the first $N_0/2$ points $(0 \le n \le (N_0/2) - 1)$ of X_r by using Eq. (8.41) and the last $N_0/2$ points by using Eq. (8.44) as

$$X_r = G_r + W_{N_0}^r H_r \qquad 0 \le r \le \frac{N_0}{2} - 1 \tag{8.45a}$$

$$X_{r+(N_0/2)} = G_r - W_{N_0}^r H_r \qquad 0 \le r \le \frac{N_0}{2} - 1 \tag{8.45b}$$

Thus, an N_0-point DFT can be computed by combining the two $(N_0/2)$-point DFTs, as in Eqs. (8.45). These equations can be represented conveniently by the *signal flow* graph depicted in Fig. 8.21. This structure is known as a *butterfly*. Figure 8.22a shows the implementation of Eqs. (8.42) for the case of $N_0 = 8$.

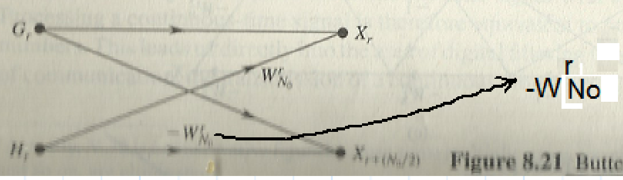


Figure 8.21 Butterfly signal flow graph.

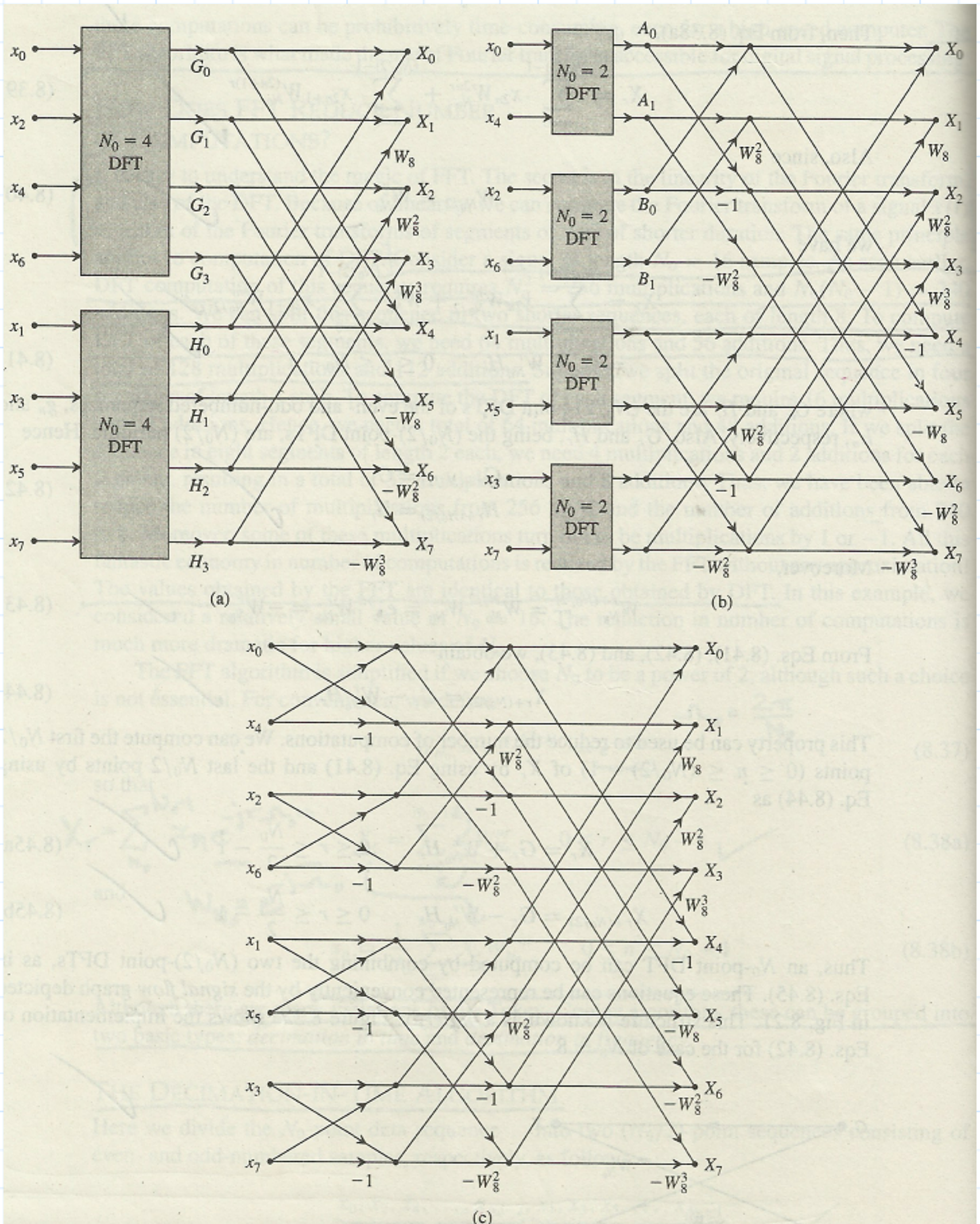


Figure 8.22 Successive steps in an 8-point FFT.

The next step is to compute the $(N_0/2)$-point DFTs G_r and H_r. We repeat the same procedure by dividing g_n and h_n into two $(N_0/4)$-point sequences corresponding to the even- and odd-numbered samples. Then we continue this process until we reach the one-point DFT. These steps for the case of $N_0 = 8$ are shown in Fig. 8.22a, 8.22b, and 8.22c. Figure 8.22c shows that the two-point DFTs require no multiplication.

To count the number of computations required in the first step, assume that G_r and H_r are known. Equations (8.45) clearly show that to compute all the N_0 points of the X_r, we require N_0 complex additions and $N_0/2$ complex multiplications† (corresponding to $W_{N_0}^r H_r$).

In the second step, to compute the $(N_0/2)$-point DFT G_r from the $(N_0/4)$-point DFT, we require $N_0/2$ complex additions and $N_0/4$ complex multiplications. We require an equal number of computations for H_r. Hence, in the second step, there are N_0 complex additions and $N_0/2$ complex multiplications. The number of computations required remains the same in each step. Since a total of $\log_2 N_0$ steps is needed to arrive at a one-point DFT, we require, conservatively, a total of $N_0 \log_2 N_0$ complex additions and $(N_0/2) \log_2 N_0$ complex multiplications, to compute the N_0-point DFT. Actually, as Fig. 8.22c shows, many multiplications are multiplications by 1 or -1, which further reduces the number of computations.

The procedure for obtaining IDFT is identical to that used to obtain the DFT except that $W_{N_0} = e^{j(2\pi/N_0)}$ instead of $e^{-j(2\pi/N_0)}$ (in addition to the multiplier $1/N_0$). Another FFT algorithm, the *decimation-in-frequency* algorithm, is similar to the decimation-in-time algorithm. The only difference is that instead of dividing x_n into two sequences of even- and odd-numbered samples, we divide x_n into two sequences formed by the first $N_0/2$ and the last $N_0/2$ digits, proceeding in the same way until a single-point DFT is reached in $\log_2 N_0$ steps. The total number of computations in this algorithm is the same as that in the decimation-in-time algorithm.

8.7 SUMMARY

<<------Read

A signal bandlimited to B Hz can be reconstructed exactly from its samples if the sampling rate $f_s > 2B$ Hz (the sampling theorem). Such a reconstruction, although possible theoretically, poses practical problems such as the need for ideal filters, which are unrealizable or are realizable only with infinite delay. Therefore, in practice, there is always an error in reconstructing a signal from its samples. Moreover, practical signals are not bandlimited, which causes an additional error (aliasing error) in signal reconstruction from its samples. When a signal is sampled at a frequency f_s Hz, samples of a sinusoid of frequency $(f_s/2) + x$ Hz appear as samples of a lower frequency $(f_s/2) - x$ Hz. This phenomenon, in which higher frequencies appear as lower frequencies, is known as aliasing. Aliasing error can be reduced by bandlimiting a signal to $f_s/2$ Hz (half the sampling frequency). Such bandlimiting, done prior to sampling, is accomplished by an antialiasing filter that is an ideal lowpass filter of cutoff frequency $f_s/2$ Hz.

The sampling theorem is very important in signal analysis, processing, and transmission because it allows us to replace a continuous-time signal with a discrete sequence of numbers. Processing a continuous-time signal is therefore equivalent to processing a discrete sequence of numbers. This leads us directly into the area of digital filtering (discrete-time systems). In the field of communication, the transmission of a continuous-time message reduces to the transmission of

†Actually, $N_0/2$ is a conservative figure because some multiplications corresponding to the cases of $W_{N_0}^r = 1, j$, and so on, are eliminated.

The sampling theorem is very important in signal analysis, processing, and transmission because it allows us to replace a continous-time signal with a discrete sequence of numbers.

a sequence of numbers. This opens doors to many new techniques of communicating continuous-time signals by pulse trains.

The dual of the sampling theorem states that for a signal timelimited to τ seconds, its spectrum $X(\omega)$ can be reconstructed from the samples of $X(\omega)$ taken at uniform intervals not greater than $1/\tau$ Hz. In other words, the spectrum should be sampled at a rate not less than τ samples/Hz.

To compute the direct or the inverse Fourier transform numerically, we need a relationship between the samples of $x(t)$ and $X(\omega)$. The sampling theorem and its dual provide such a quantitative relationship in the form of a discrete Fourier transform (DFT). The DFT computations are greatly facilitated by a fast Fourier transform (FFT) algorithm, which reduces the number of computations from something on the order of N_0^2 to $N_0 \log N_0$.

Starting with our tutorial textbook authors Derose and Veronis.
Textbook: Signals and Systems Using Prime/Mathcad.

5.1 Introduction

In Chapter 3, we introduced the Discrete Fourier Transform (DFT) and used MathCAD to compute the DFT of a given sequence. Although we did not compute the time it took to get the result, in practice, time is always an issue when computing a Fourier Transform of a sequence. In Chapter Five, we introduce another method of computing the Discrete Fourier Transform of a given sequence. This method is called the Fast Fourier Transform since it is faster or takes less time to produce the result. The Fast Fourier Transform is derived from the DFT equation. The Fast Fourier Transform (FFT) is faster than the Discrete Fourier Transform because it produces the same result with less operations, that is, the FFT algorithm requires less computational overhead compared to the DFT. It reduces the number of additions and multiplications to produce the same result.

To get a feel for the FFT, let's review the DFT equation. Let $x(n)$ be a given finite sequence; for example, we can assume a given sequence with a length of 8 (N = 8). Then the Discrete Fourier Transform of $x(n)$ is

$$X(k)=\sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi}{N}nk} \qquad \text{(Equ.5-1)}$$

where $k=0\ldots N-1$ and $n=0\ldots N-1$ as shown from the summation.

If we let

$$e^{-j\frac{2\pi}{N}} = W_N$$

we can rewrite the equation as follows

$$X(k)=\sum_{n=0}^{N-1} x(n)W_N^{nk} \qquad \text{(Equ.5-2)}$$

Now if we want to separate $x(n)$ into 2 sequences, one of them odd and one of them even, we can do so by letting $X(k)$ equal to 2 functions.

We can let

$x(n)=x_1(n)+x_2(n)$

where

$x_1(n)=x(2n)$, the even sequence of $x(n)$ and

$x_2(n)=x(2n+1)$, the odd sequence of $x(n)$

Since we partitioned $x(n)$ into 2 sequences, the length of each sequence is half the original length. Instead of going from 0 to N-1, now we go from $0\ldots\frac{N}{2}-1$. So, for $x_1(n)$ and $x_2(n)$, n goes from $n=0\ldots\frac{N}{2}-1$.

The original equation $X(k)=\sum_{n=0}^{N-1}x(n)W_N^{nk}$ can be rewritten now as

$$X(k)=\underbrace{\sum_{n=0}^{\frac{N}{2}-1}x(2n)W_N^{2nk}}_{\text{Even}}+\underbrace{\sum_{n=0}^{\frac{N}{2}-1}x(2n+1)W_N^{(2n+1)k}}_{\text{Odd}} \qquad \text{(Equ.5-3)}$$

or, as given by Equation 5-4 below

$$X(k)=\sum_{n=0}^{\frac{N}{2}-1}x_1(n)W_N^{2nk}+\sum_{n=0}^{\frac{N}{2}-1}x_2(n)W_N^{(2n+1)k} \qquad \text{(Equ.5-4)}$$

By knowing that W_N^{2nk}, which is equal to $e^{-j\frac{2\pi}{N}nk}$ is periodic or can give the same result for many values of n and k, it is possible to reduce the number of operations. We call this process the radix-2 FFT, since the given sequence is further partitioned in half to produce the result.

5.2 MathCAD Approach to the DFT

<<------ The procedure explained for Prime/Mathcad

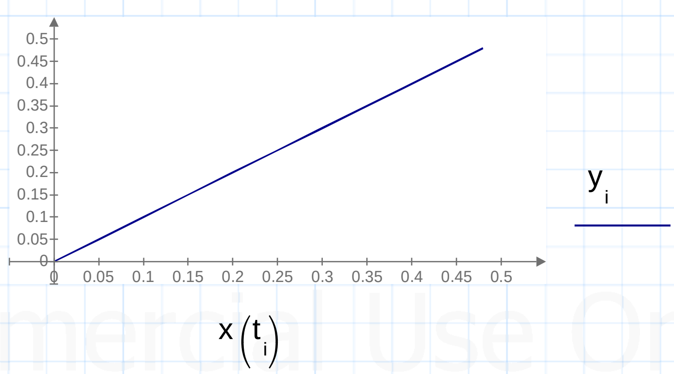
In MathCAD, there is a built - in FFT function that implements the DFT equation. As mentioned above, MathCAD uses radix-2 FFT to compute the DFT of a given sequence. Lets discuss an example. As we recall our example from the continuous time Fourier transform, where we were given a signal and were asked to find its Fourier transform, we can use the same approach in this example as well. Let's start with a continuous time signal and convert it to a discrete time signal. We can proceed as follows. As part of the discussion, we mention that , before we take the FFT of a function, we must put it in terms of vector. We cannot take the FFT of $x(t)$ if $x(t)$ is not a vector. In MathCAD we can convert a signal to a vector by using a subscript. For instance, consider $t := 0..10$ and $x(t) := \sin(t)$; $x(t)$ is not a vector; we cannot define a subscript i and equate $y_i := \sin(t)$; in this case, y is not a vector; but $t_i := \frac{i}{20}$ and $x(t) := \sin(t)$, $y_i := x(t_i)$, then y is a vector. A vector $y(t)$ can also be defined by setting t as vector and call $x(t) := \sin(t)$. In this case we can also take the FFT of $x(t)$ directly by calling $y := fft(x(t))$.

<<--- vector - indexing shown below uses matrix

$\text{ORIGIN} := 0$

$i := 0..10$ $\quad t_i := \frac{i}{20}$ $\quad x(t) := \sin(t)$ $\quad y_i := x(t_i)$ $\quad$ using 1 column matrix for vector

$$y_i = \begin{bmatrix} 0 \\ 0.049979 \\ 0.099833 \\ 0.149438 \\ 0.198669 \\ 0.247404 \\ 0.29552 \\ 0.342898 \\ 0.389418 \\ 0.434966 \\ 0.479426 \end{bmatrix}$$



Example 5.1

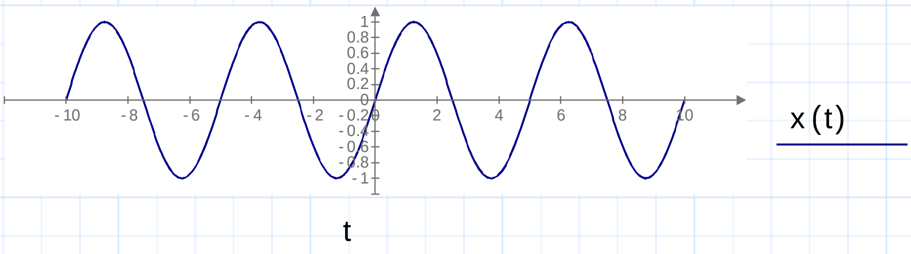
clear (x)   clear (t)

$f_0 := 50$

$T_0 := \frac{1}{f_0}$

$\omega_0 := \frac{2 \cdot \pi}{T_0}$ fundamental frequency in radians

$x(t) := \sin(\omega_0 \cdot t)$ defining the signal



Next the sample process.
Since Prime (Mathcad) uses the radix-2 FFT, the length of the signal N must be given as the power of 2.

$N := 2^4$ radix 2 for Prime FFT

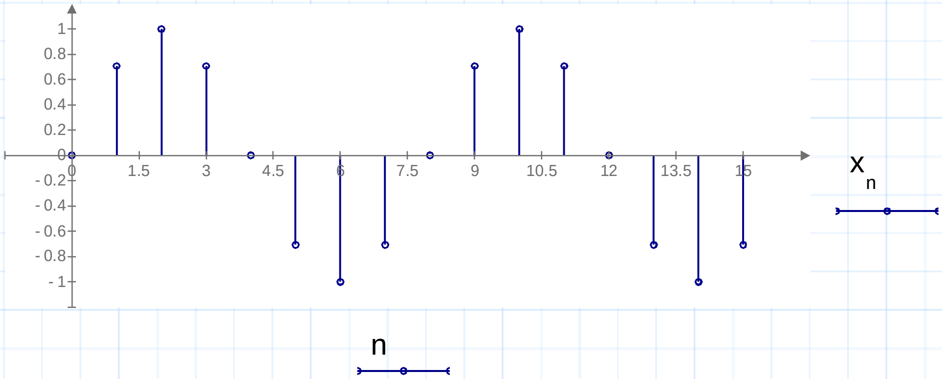
$\omega_s := 8$ sampling angular frequency

$n := 0 .. N-1$ the indexing; 0 to 2^4 - 1 = 0 - 15

$T_s := \frac{2 \cdot \pi}{\omega_s}$ sampling time/period/interval in radians

$x_n := \sin(n \cdot T_s)$ defining the sampling signal in discrete form
n is a matrix vector (single column)

plot the signal in discrete form (stem plot)



Above is the plot of discrete samples of the analog signal
n from 0 to 15 (2^4 -1) in plot above for the radix-2

$$X := \frac{fft(x)}{\sqrt{N}}$$

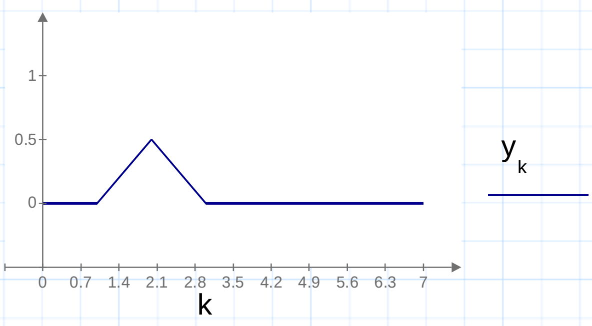
Applying the FFT function to the sampling signal x_n. Do not need to show x_n just the variable x.
Sqrt(N) is the normalisation factor. Its not necessary because it is normalsing to the squareroot of the number of samples N.
Most publications skip the normalisation sqrt(N)

$$k := 0 .. \left(\frac{N}{2} - 1\right)$$

partition the indexing of k to be half of the original index
- as the theory states (even and odd partition)

$$y_k := \left|X_k\right|$$

take the magnitude of X to remove the imaginary constraints

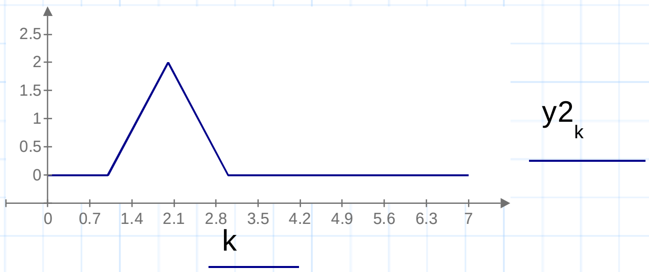


Remember as if almost seeking this plot shape from previous exercise(s) in another file.

Now lets do without the sqrt(N) normalisation

$$X2 := fft(x) \qquad k := 0 .. \left(\frac{N}{2} - 1\right) \qquad y2_k := \left|X2_k\right|$$

Plot without normalisation below, it has an amplitude 4 times the plot with normalisation.



$Amp4 := \sqrt{N} = 4$

The amplitude with normalisation divided the FFT by 4, removing it multiplies the amplitude by 4.
So the amplitude increases from 0.5 to 2.
Currently at this stage I like the normalisation because the amplitude is under 1.0.

We used the FFT function from Prime/Mathcad which made it much simpler.

Next an example demonstrating the usefulness of FFT in practical applications.

Example 5.2

This very good example illustrates the usefulness and the application of the FFT function. Assume that we were given a signal buried in noise; we don't know the frequency component of the signal. To solve this problem, we can apply the FFT function to determine the frequency of the signal. By doing so, if the signal had several frequency components, we could see each individual frequency. To understand this process more clearly, we can use a Fourier series signal from a previous chapter. In this example, we will see the final signals with all the frequencies.

Example 5.2

clear(x) clear(t) clear(X)

$N := 256$ Number of points for the sample iterations

$f := 5$ frequency

$\omega_0 := 2 \cdot \pi$

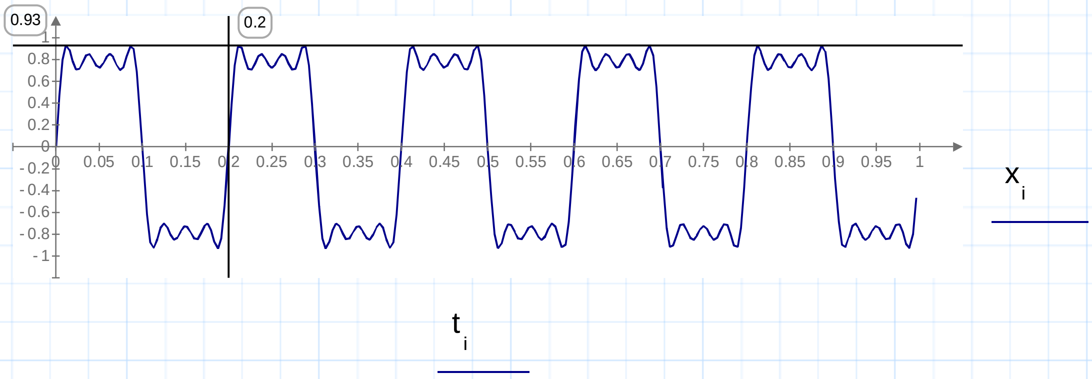
$i := 0..N-1$ sets the interval of sampling or interval time

$t_i := \frac{i}{N}$ sampling time ti - i is a vector set by matrix operation t'['i

Create signal to be sampled:

$$x_i := \sin\left(\omega_0 \cdot f \cdot t_i\right) + \frac{1}{3} \cdot \sin\left(3 \cdot \omega_0 \cdot f \cdot t_i\right) + \frac{1}{5} \cdot \sin\left(5 \cdot \omega_0 \cdot f \cdot t_i\right) + \frac{1}{7} \cdot \sin\left(7 \cdot \omega_0 \cdot f \cdot t_i\right)$$

Plot the signal to be sampled (this uses the index method of plotting):

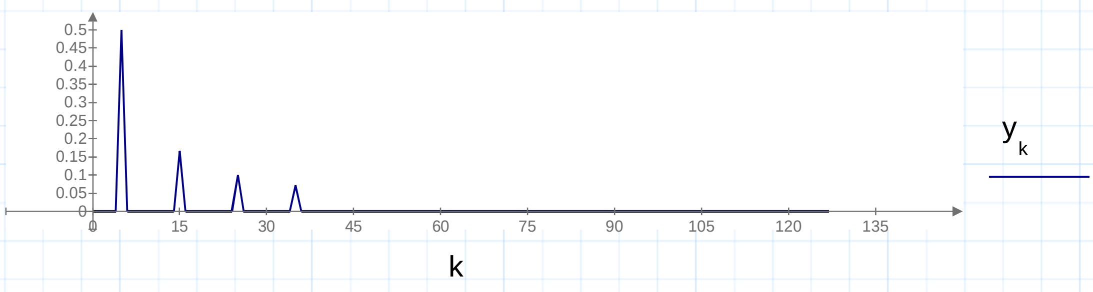


$X := \dfrac{fft(x)}{\sqrt{N}}$ the FFT of the signal to be sampled $X2 := X$ X2 for example 5.4

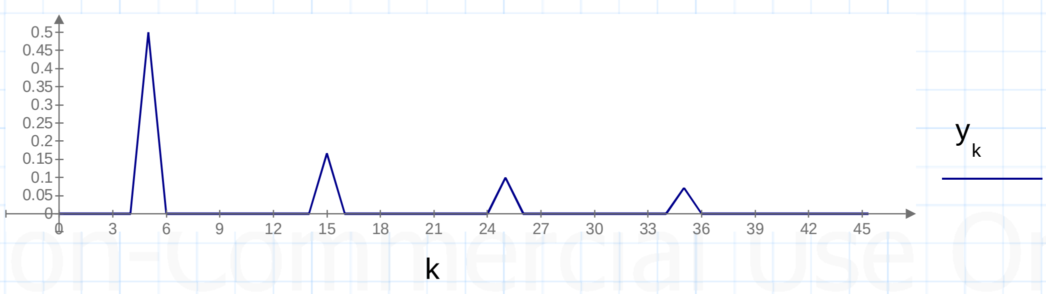
$k := 0 .. \left(\dfrac{N}{2} - 1\right)$ range of the frequency - partition the index

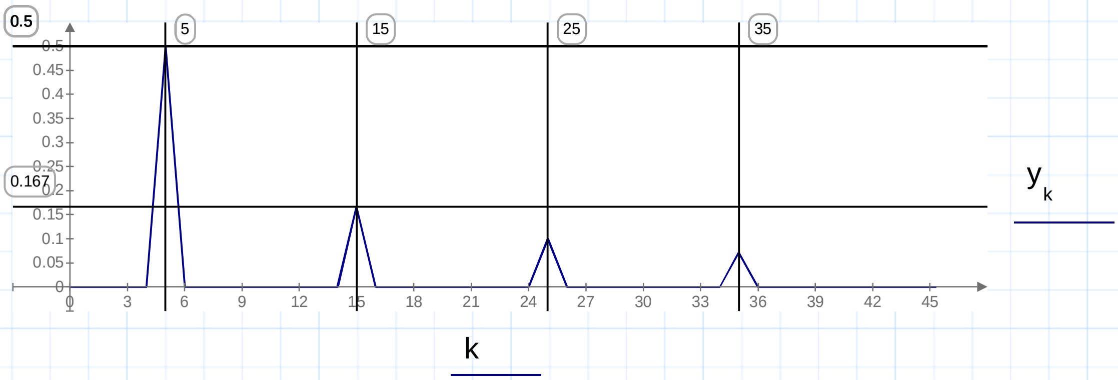
$y_k := \left|X_k\right|$ apply magnitude to remove imaginary constraint

Plot of the FFT signal:



$PlotRange := \dfrac{N}{2} - 1 = 127$ Plot range is from 1 - 127, as set.





The original signal had 4 sinusoidal terms and each term has its frequency shown above.
Signal to be sampled or original signal below

$$x_i := \sin\left(\omega_0 \cdot f \cdot t_i\right) + \frac{1}{3} \cdot \sin\left(3 \cdot \omega_0 \cdot f \cdot t_i\right) + \frac{1}{5} \cdot \sin\left(5 \cdot \omega_0 \cdot f \cdot t_i\right) + \frac{1}{7} \cdot \sin\left(7 \cdot \omega_0 \cdot f \cdot t_i\right)$$

$f_1 := 1 \cdot f = 5$ $\quad$ $amp1 := 0.5$ $\quad$ from each term above

$f_2 := 3 \cdot f = 15$ $\quad$ $amp2 := \left(\frac{1}{3}\right) \cdot amp1 = 0.167$

$f_3 := 5 \cdot f = 25$

$f_4 := 7 \cdot f = 35$

The FFT shows the same frequencies on the plot above at 5, 15, 25 and 35, with corresponding amplitudes. So here mission accomplished per example requirements.

Inverse FFT

5.3 The Inverse Fast Fourier Transform

The Inverse Discrete Fourier Transform (IDFT) can be evaluated from Equation 5-5.

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{j\frac{2\pi}{N}nk} \qquad \text{(Equ.5-5)}$$

or

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W_N^{-nk} \qquad \text{(Equ.5-6)}$$

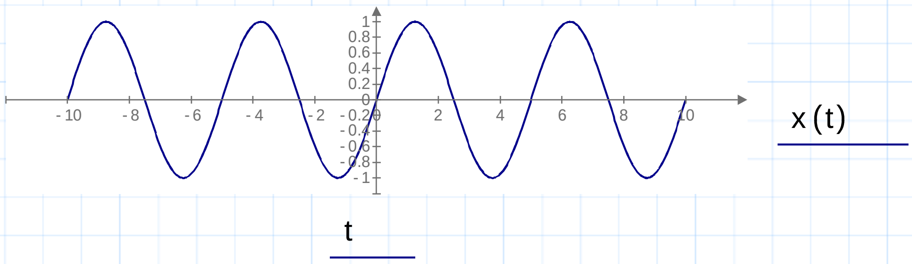
In practice, the FFT algorithm is used to compute the inverse FFT. In MathCAD, the iFFT is used to compute the inverse FFT function. As an example, let's compute the inverse FFT of the two examples we have given above. We assume that the signal is in the frequency domain, and we wish to find the time domain of that signal. We proceed as follows:

Example 5.3

Using the FFT of Example 5.1 - Repeated here

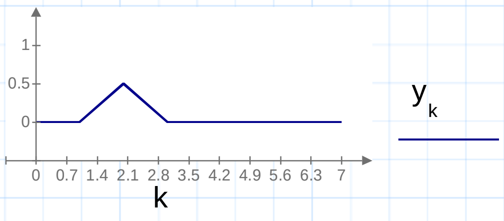
clear (x) clear (t) clear (X) clear (n)

$f_0 := 50$ $T_0 := \frac{1}{f_0}$ $\omega_0 := \frac{2\cdot\pi}{T_0}$ $x(t) := \sin(\omega_0 \cdot t)$



$N := 2^4$ $\omega_s := 8$ $n := 0..N-1$ $T_s := \frac{2\cdot\pi}{\omega_s}$

$x_n := \sin(n \cdot T_s)$ $X1 := \frac{fft(x)}{\sqrt{N}}$ $k := 0..\left(\frac{N}{2}-1\right)$ $y_k := |X1_k|$

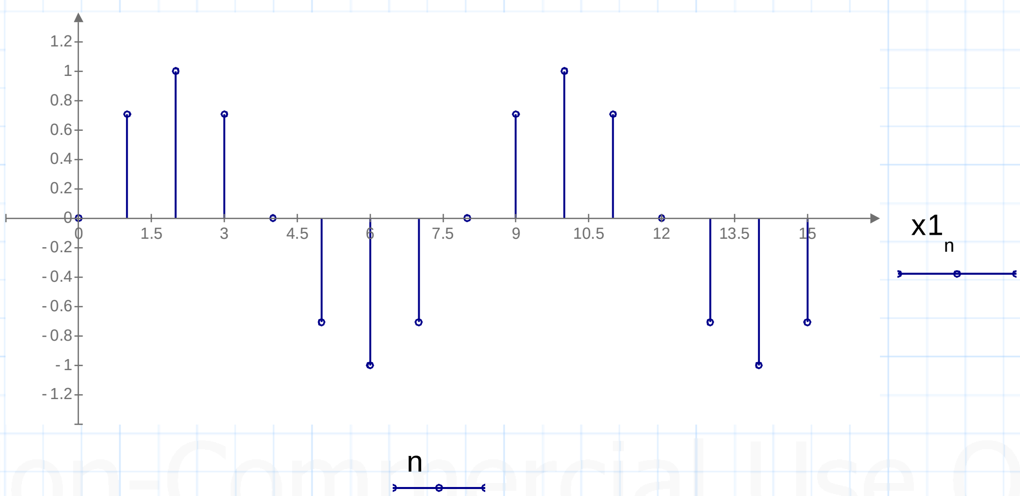


Frequency domain plot.

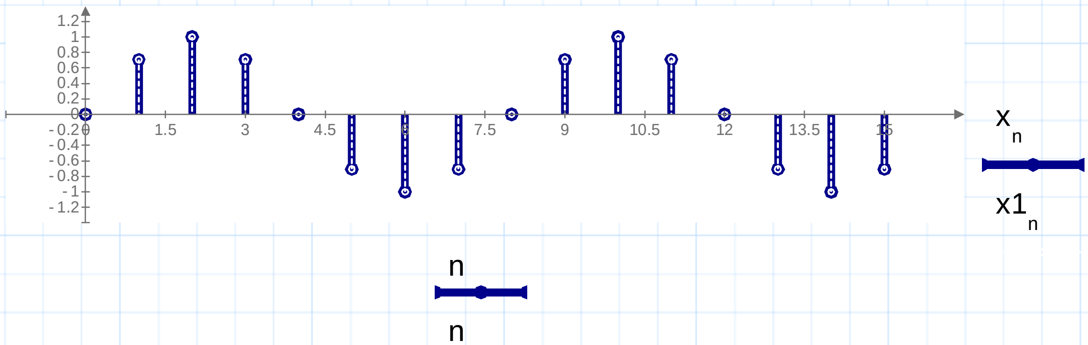
Proceed with inverse Fast Fourier Transform ifft:

Do NOT place n in variable x1 below, it has to be a scalar, in plot place n for vector plot.

$x1 := ifft(X1) \cdot \sqrt{N}$ remember in reverse remove divison by sqrt N, by multipling sqrt N



Both the time domain in discrete time - blue, and the inverse fast fourier transform plot - white

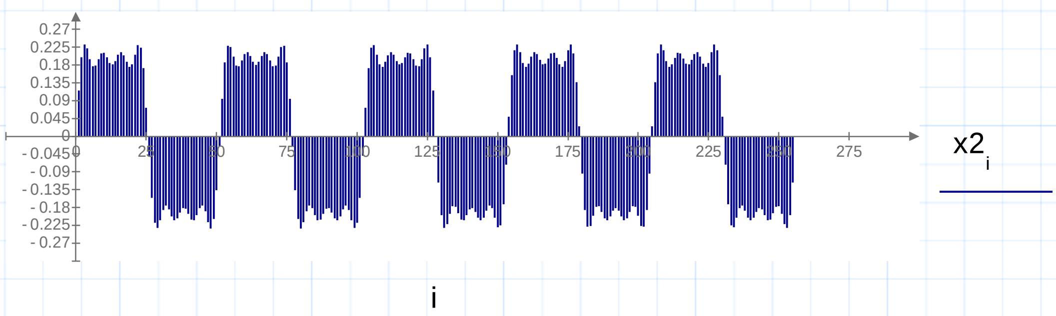


Example 5.4

In example 5.2 X2 was set equal to X so that we do not have to repeat the steps here as we did in example 5.3.

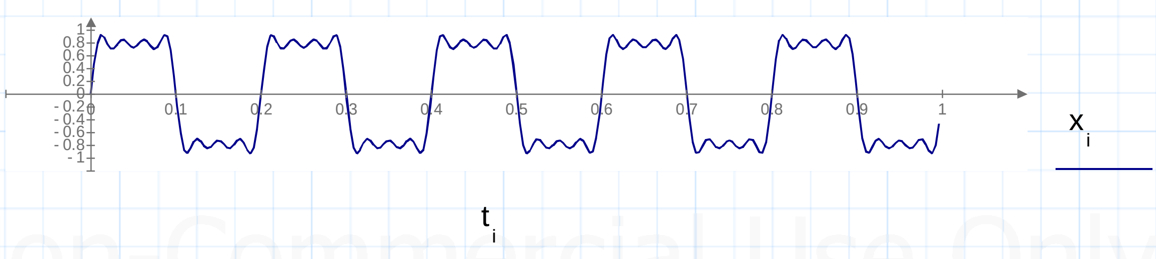
$x2 := \mathrm{ifft}(X2) \cdot \sqrt{N}$

subscript i was used in example 5.2 for indexing - signal x2-i matches the time domain x_i



$N := 256 \qquad i := 0..N-1 \qquad t_i := \frac{i}{N} \qquad f := 5 \qquad \omega_0 := 2 \cdot \pi$

$x_i := \sin(\omega_0 \cdot f \cdot t_i) + \frac{1}{3} \cdot \sin(3 \cdot \omega_0 \cdot f \cdot t_i) + \frac{1}{5} \cdot \sin(5 \cdot \omega_0 \cdot f \cdot t_i) + \frac{1}{7} \cdot \sin(7 \cdot \omega_0 \cdot f \cdot t_i)$



Fast Convolution.

Some other Applications of the FFT - Fast Convolution

We have thus determined that we can use the FFT function to look at the frequency component of a signal buried in noise. It would have been impossible to identify the frequency of that signal without using the Fourier transform. There are numerous other applications where the FFT can be used. For instance, in Chapter Two, we used convolution to analyze the response of a given signal. To make this possible, we used the convolution sum to plot the signal response. There is another way we could have done it by using the FFT function instead We can use the FFT function to perform convolution; the process is called Fast Convolution. In Example 5-5, we use MathCAD to carry out fast convolution.

Explanation of the steps for Fast Fourier Transform:

The input of a linear system is given as

$$x(n)=u(n)-u(n-4) \qquad \text{(Equ.5-7)}$$

also the impulse response of the system is given as

$$h(n)=u(n)-u(n-4) \qquad \text{(Equ.5-8)}$$

We can use the FFT function to compute the output response of the system $y(n)=x(n)*h(n)$. Since we know that convolution in the time domain corresponds to multiplication in the frequency domain, we can take the FFT of both signals in the time domain, multiply them together to produce a result., and finally take the inverse FFT of the result. Before we proceed further with the example, the following steps in fast convolution must be understood. Table 5-1 holds true both for time - and frequency – domain.

Table 5-1

Time Domain	Frequency Domain
$y(n)=x(n)*h(n)$	$Y(\omega)=X(\omega)H(\omega)$
$y(n)=x(n)h(n$	$Y(\omega)=X(\omega)*H(\omega)$

1. We must pad with zeroes, the length of N of the sequence $x(n)$ and $h(n)$, so that the length of the sequence $x(n)$ and $h(n)$ should be $length[x(n)]+length[h(n)]-1$. For $\boxed{\text{example,}}$ if $x(n)=[1\ \ 1\ \ 1\ \ 1]$ and $h(n)=[1\ \ 1\ \ 1\ \ 1]$. The new $x(n)$ and $h(n)$ are $x_1(n)$ and $h_1(n)$

   $x_1(n)=[1\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0]$

   $h_1(n)=[1\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0]$

2. We calculate the FFT of $x_1(n)$ and $h_1(n)$ in the form of $X_1=fft(x_1)$ and $H_1=fft(h_1)$
3. We multiply both results of the FFT in the form of $Y=X_1\cdot H_1$
4. We take the inverse FFT of the result in Step 3 in the form of $y=ifft(Y)$.
5. We take the real part of the result from Step 4 in the form of $y_1=\mathrm{Re}(y)$; this is the result of the convolution, and it yields the same result as the convolution sum.
6. We must also scale the result properly to get the right amplitude by multiplying it by $\sqrt{length(y)}$

To better understand the process of fast convolution, lets study Example 5-6 First, let's plot the given signal as shown below. For this example, we will not apply all the steps described above, but we will apply all of them in the next one.

→ 5-6

→ 5.7

Example 5.6

clear (x)   clear (n)   clear (x1)   clear (x2)   clear (y)

$ORIGIN := -4$ We want to add zeros later so we shift the origin to -4

The input of a linear system is given as

$x(n) = u(n) - u(n-4)$

also the impulse response of the system is given as

$h(n) = u(n) - u(n-4)$

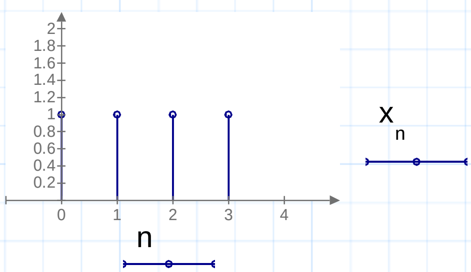
$N := 4$

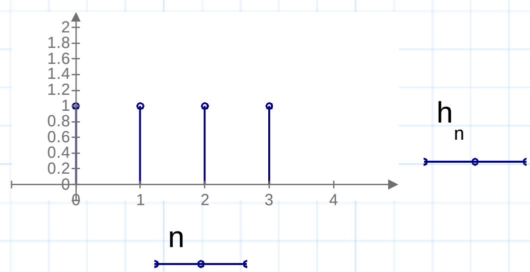
$n := 0 .. N-1$

$u_n := if(n \geq 0, 1, 0)$ defining the discrete unit step function

$x_n := u_n - u_{n-4}$ defining the input signal - using matrix vector

$h_n := u_n - u_{n-4}$ defining the impulse response





$ORIGIN := 0$ Change the origin back to zero to start at 0
We want to start at zero to add the 0 values

$j := 0 .. 7$ defining the index of the 0 length vector

$x1_j := 0$ $h1_j := 0$ defining two 0 length vectors

Defining 2 new vectors to combine the 0 length vectors

$x2 := stack(x1, x)$ vector x1_j + x_n $length(x2) = 16$ no of elements
$h2 := stack(h1, h)$ vector h1_j + h_n $length(h2) = 16$ in vector x2 and h2

stack(A , B, C, ...)—Returns an array formed by placing A, B, C, ... top to bottom.

Next perform the convolution of both the signals:
Take the FFT of both signals then multiply them together.

$X2 := \mathrm{fft}(x)$ the fft of x[n is fft(x)

$H2 := \mathrm{fft}(h)$ the fft of h[n is fft(h)

$$X2 = \begin{bmatrix} 1.41 \\ -0.35-0.85j \\ 0 \\ -0.35-0.15j \\ 0 \end{bmatrix} \qquad H2 = \begin{bmatrix} 1.41 \\ -0.35-0.85j \\ 0 \\ -0.35-0.15j \\ 0 \end{bmatrix}$$

$Y := \overrightarrow{X2 \cdot H2}$ perform an element by element multiplication for the vectors - vectorisation

$$Y = \begin{bmatrix} 2 \\ -0.6+0.6j \\ 0 \\ 0.1+0.1j \\ 0 \end{bmatrix}$$

$y := \mathrm{ifft}(Y)$ inverse FFT of Y

$\mathrm{length}(y) = 8$

$n := 0 .. \mathrm{length}(y) - 1$

defining a new index, length function returns the number of elements in a vector
n was first defined as 0..N-1 where N = 4
now n: 0 - (8-1), n: 0 - 7.

$M := \sqrt{\mathrm{length}(y)}$

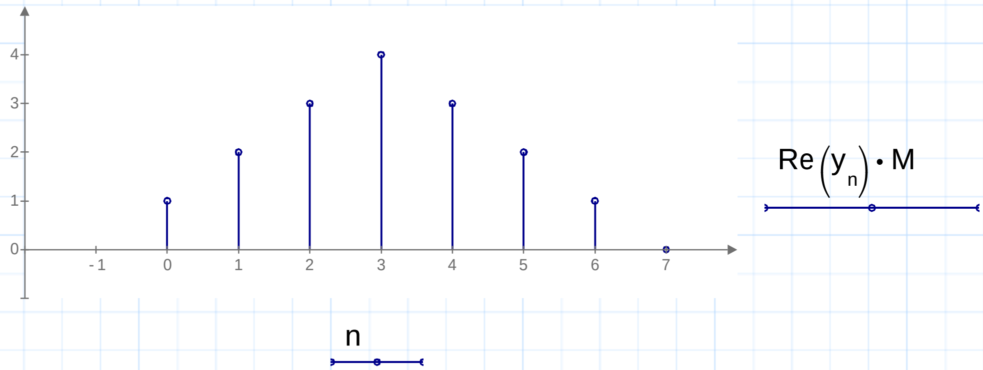
$M = 2.83$

$$n = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{bmatrix} \qquad y = \begin{bmatrix} 0.354 \\ 0.707 \\ 1.061 \\ 1.414 \\ 1.061 \\ 0.707 \\ 0.354 \\ 0 \end{bmatrix}$$

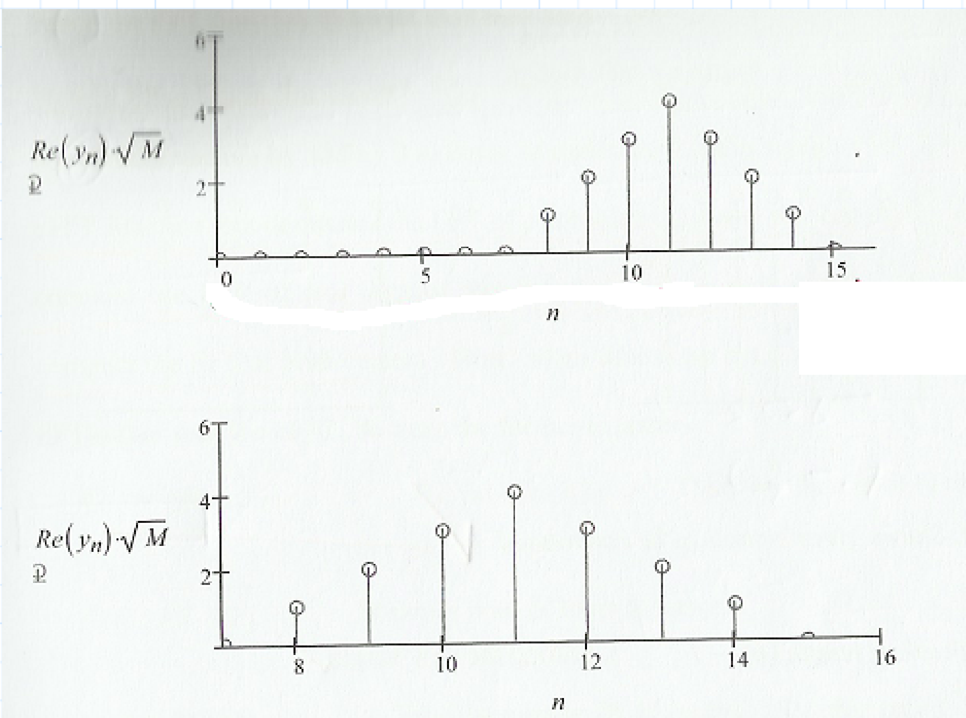
$y_0 = 0.35$
$y_1 = 0.71$
$y_2 = 1.06$
$y_3 = 1.41$
$y_4 = 1.06$
$y_5 = 0.71$
$y_6 = 0.35$
$y_7 = 0$

$$y1 := y \cdot M = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 3 \\ 2 \\ 1 \\ 0 \end{bmatrix}$$

Plot of response y the result of convolution.
Plot is in discrete time domain.



Textbook plot range of n is from n = 0 to 15
With values of 0 for n= 0 to 7, then continues as shown in plot above from 8 to 15.
The results are ok but the plot range is not matching, maybe an error somewhere. Next example has a similar situation but with an improvement.



Textbook plots shown above.

Example 5.7

clear(x) clear(h) clear(y) clear(x1) clear(x2) clear(u) clear(N) clear(i) clear(n)

vector x = [1 1 1 1 1 1 1 1]
vector h = [1 1 1 1]

$ORIGIN := -8$

$N := 8$

$n := 0 .. N-1$

$u_n := if(n \geq 0, 1, 0)$ defining the discrete unit step function

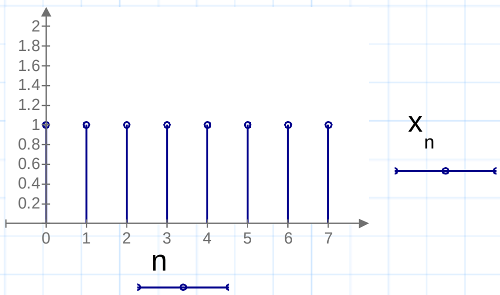
$x_n := u_n - u_{n-8}$ defining the input signal - using matrix vector

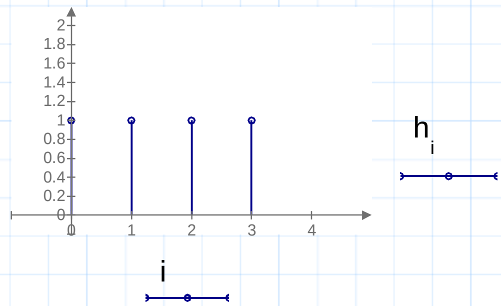
$i := 4$

$i := 0 .. N-5$

$u_i := if(i \geq 0, 1, 0)$ defining the discrete unit step function

$h_i := u_i - u_{i-4}$ defining the impulse response





The signal plots above are correct.

$length(x) := 8$ 0-7 is 8 positions

$length(h) := 4$ 0-3 is 4 position

Zero overlaps, one zero position has to be removed

$N := 8 + 4 - 1$

$N = 11$

$N1 := N - length(x)$

$N1 := N - 8 = 3$ defining the length of the 1st 0 vector

$j := 0 .. N1 - 1$ defining the index for the 1st 0 vector

$N2 := N - length(h)$ defining the length of the 2nd 0 vector

$N2 := N - 4 = 7$

$k := 0 .. N2 - 1$ defining the index for the 2nd 0 vector

$x1_j := 0$ defining the 1st 0 vector

$h1_k := 0$ defining the 2nd 0 vector

Next pack the vectors together using stack:

$x2 := \text{stack}(x1, x)$
$h2 := \text{stack}(h1, h)$

Note:
Prime/Mathcad uses radix-2 FFT, so the vector has to be in a length of power 2.
We can add zeros to make the length 16 (2^4).

However Prime has a function cFFT that computes the FFT of complex data.
We can use this function also to compute real data of any length.

So we use the cFFT function to compute the FFT of both vectors.
Hint: when time is a issue/criteria use FFT rather than cFFT because FFT is faster.

$X2 := \text{cfft}(x2)$ Take the fft of both of them.
$H2 := \text{cfft}(h2)$

$Y := \overrightarrow{X2 \cdot H2}$

$y := \text{icfft}(Y)$ inverse cfft

Continued next page where vectors elements are shown.

So the cfft and icfft is used when the length of the data is not to the power of 2.

The vectors Y and y shown below

$$Y = \begin{bmatrix} 1.19 \\ -0.06-0.99j \\ -0.53+0.06j \\ 0.02+0.11j \\ -0.09+0.02j \\ 0.03+0.09j \\ 0.02-0.01j \\ 0 \\ -0.03+0.01j \\ 0.02+0.03j \\ 0 \\ 0.02+0.02j \\ 0.02-0.02j \\ 0 \\ 0 \\ 0.02+0.02j \\ 0.02-0.02j \\ 0 \\ 0.02-0.03j \\ -0.03-0.01j \\ 0 \\ 0.02+0.01j \\ 0.03-0.09j \\ -0.09-0.02j \\ 0.02-0.11j \\ -0.53-0.06j \\ -0.06+0.99j \end{bmatrix} \qquad y = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0.19 \\ 0.38 \\ 0.58 \\ 0.77 \\ 0.77 \\ 0.77 \\ 0.77 \\ 0.77 \\ 0.58 \\ 0.38 \\ 0.19 \\ 0 \end{bmatrix}$$

number of elements: -8 to 18
-8 - 0 + 0 - 18
= 27 positions in the range
N = 27-1 = 26

$\text{length}(Y) = 4$ Incorrect! $\text{length}(y) = 4$ Incorrect!

We had applied the transform and inverse transform, we need to define the new index next:

$\text{clear}(n)$

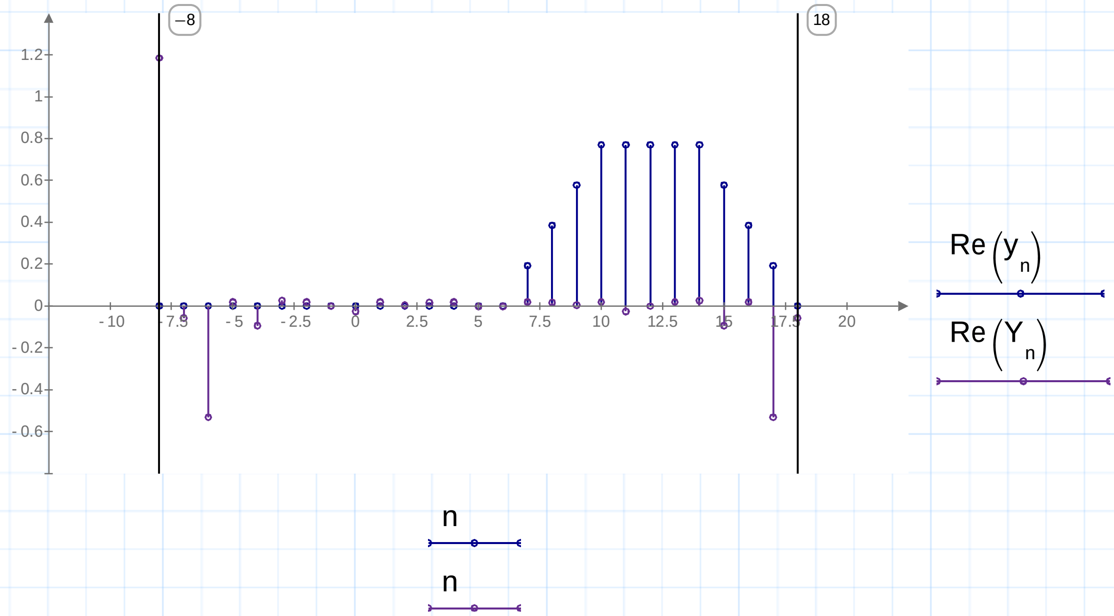
$\text{length\_y} := 26$ set manually from -8 to 18 is 26, because length(y) returned 4.

$n := -8..18$ set manually because the length(y) returns 4 which is NOT correct.

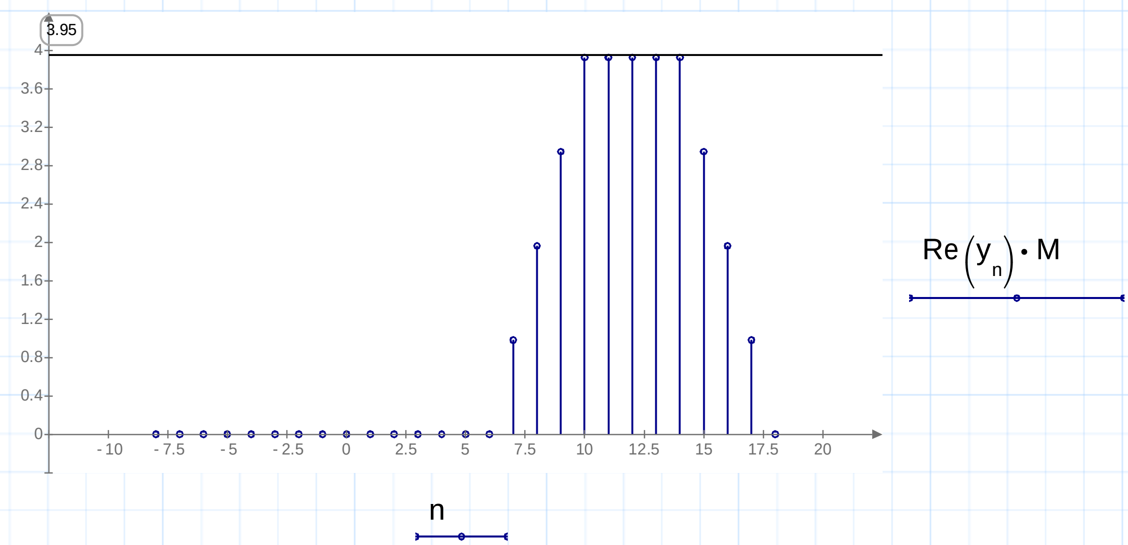
$M := \sqrt{\text{length\_y}}$ To scale the amplitude

$M = 5.09902$

Plots of Y and y

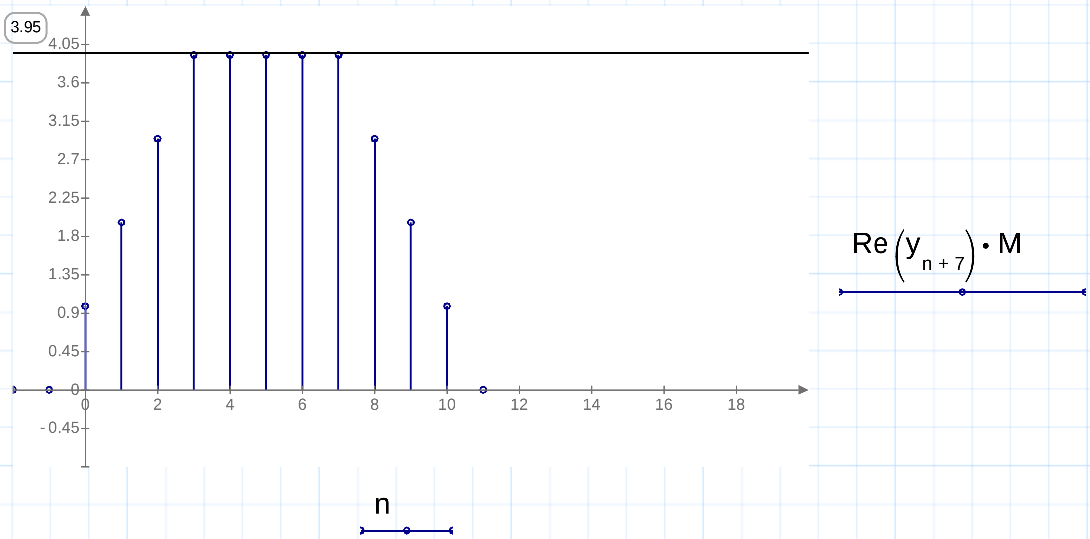


Plot of y multiplied by M, except for the range values the plot is the same as the textbook.

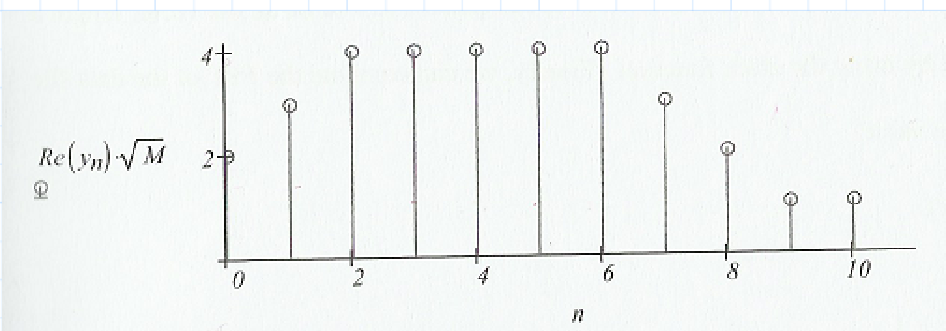


Plot of y multiplied by M, except for the range values the plot is the same as the textbook. textbook has a range of 0 to 10 for the same plot.

This was achieved by moving n to the positive side by 7, n+7. N was in the beginning of the solution set at 7, N=7. This brings the first vector element to start at 0. Also the horizontal range was set to begin at 0.



Textbook plot shown below.



Next page the textbook example for accessing a data file.

5.5 More FFT Examples

The FFT function is very useful when we wish to find the frequency component of a signal. Most of the time in real life, you are going to be given signals as data points rather than functions. It may not be possible to look at the plot of the signal and determine the frequency component, but you can use the FFT to get the frequency component of the signal. We can use the **READPRN** function in MathCAD to read a data file into a vector and take the FFT of that vector. Here is an example of how we can do that.

We use the **READPRN** function to retrieve the text file to a variable:

$x := READPRN(\text{"c:\\data.txt"})$

We compute the length of the data:

$N := length(x)$

$N = 256$

If the length of the data is not a power of 2, we can use the cFFT function instead of the FFT. If we wish to use the FFT function, we can pad 0's to x to make the vector length a power of 2 by using the stack function. Finally, we can compute the FFT of the data file and plot the value.

$y := fft(x)$

$k := 0..\frac{N}{2} - 1$

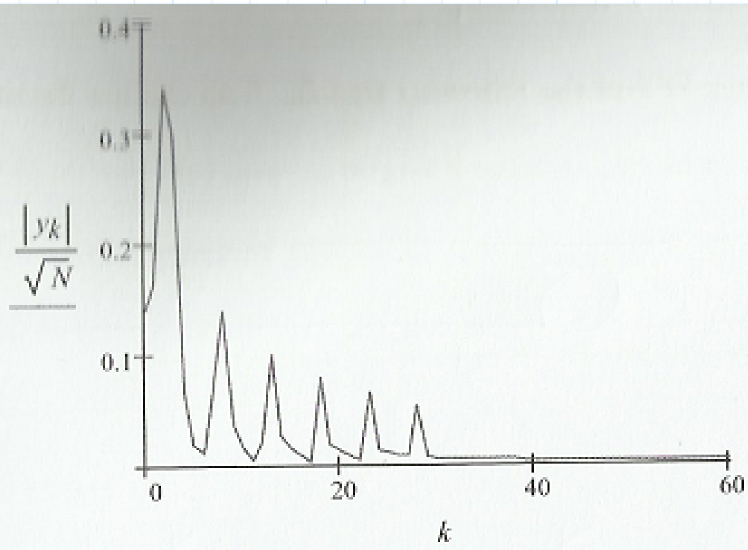


Figure 5.11: Shows the FFT of the data file

EoF - End of File.