

Spectral Analysis using the FFT

Brett Ninness

Department of Electrical and Computer Engineering
The University of Newcastle, Australia.

Having now considered the theoretical underpinnings of how spectral analysis of signals may be performed via using sampled versions of those signals, this section illustrates some of the practical issues associated with this topic by way of MATLAB example.

1 Computing the DFT, IDFT and using them for filtering

To begin this discussion on spectral analysis, let us begin by considering the question of trying to detect an underlying sinusoidal signal component that is buried in noise. Such problems occur, for example, very commonly in the area of telecommunications. Suppose that such a signal-in-noise situation gives rise to the observed data record as shown in figure 1.

Here the samples are collected at a spacing of $\Delta = 0.1$ second apart, and are contained in a MATLAB vector \mathbf{x} so that since the samples are plotted with the command

```
>> fs=10;  
>> t = (0:1:length(x)-1)/fs;  
>> plot(t,x)
```

then, as is the MATLAB plotting default, linear interpolation is performed between the sample points to give the impression of a continuous signal. Note also, in the above commands, the specification of fs as $f_s = 1/\Delta$ which is the sampling frequency in Hz.

Now the signal shown in figure 1 does not appear, by eye, to have any underlying sinusoidal signal component at all; instead it appears to be completely random and hence completely comprised of noise.

However, to check this, the Discrete Fourier Transform (DFT)

$$X(m\omega_s) = \sum_{k=0}^{N-1} x_k e^{-jm\omega_s k} \quad ; \omega_s \triangleq \frac{2\pi}{N} \quad (1)$$

could be computed to see if a spectral peak is present. For this purpose, MATLAB has the `fft` function, which performs the computation DFT computation (1) in an efficient manner, and hence is called the Fast Fourier Transform (FFT). Using it is very simple. Hand it a vector \mathbf{x} of time domain samples, and it returns a vector \mathbf{X} of samples $X(m\omega_s)$, $m = 0, 1, \dots, N - 1$ of the DFT computation:

```
>> X = fft(x);  
>> plot(abs(X));
```

These commands result in the plot shown in figure 2. There are some interesting aspects of this plot. Firstly, and most importantly, there are two clear spectral peaks, which indicates that there is more to this signal than just noise. The second point of interests arises by trying to interpret from this plot how many spectral components exist. Are there two clear ones?

In fact there is only one spectral component, since the frequency range used in the MATLAB DFT computation is as shown in figure 3. There it is illustrated that if the underlying continuous time signal has a spectrum $X(\omega) = \mathcal{F}\{x\}$, then as discussed in lectures, the DFT $X_N(\omega)$ will be a repeated version of this spectrum, with the repetition occurring every $\omega = 2\pi/\Delta$ seconds.

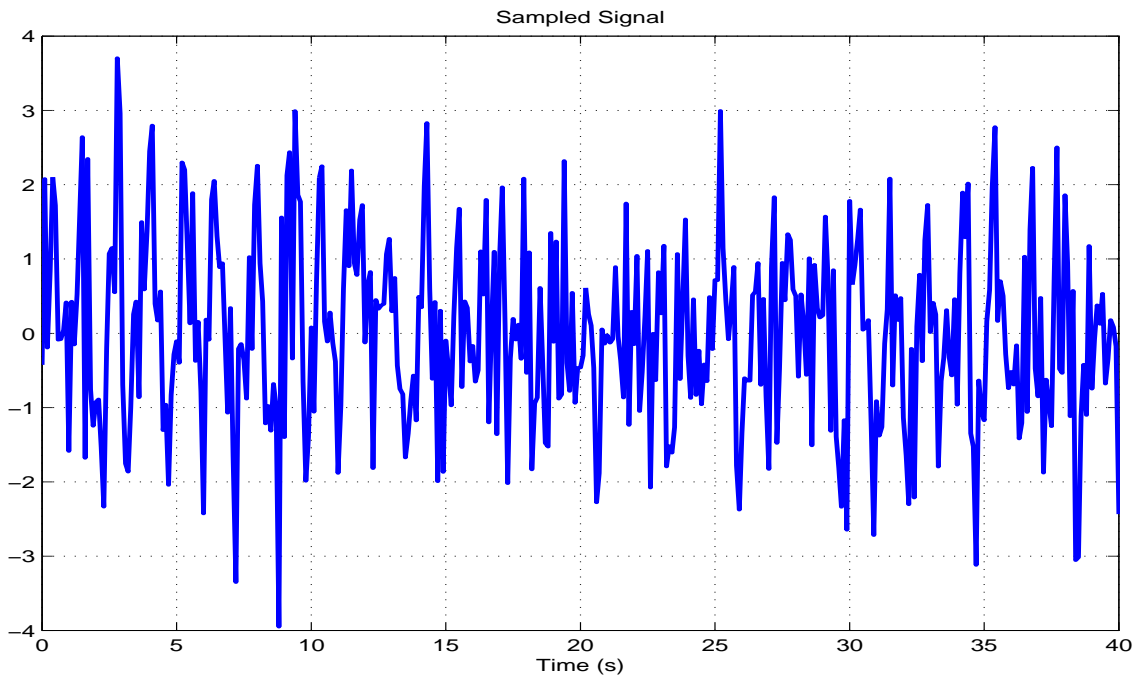


Figure 1: *Sampled signal record.*

Now, if $x(t)$ is real valued, then $X(\omega)$ will be symmetric, and hence the part of $X_N(\omega)$ in the range $\omega \in [\pi/\Delta, 2\pi/\Delta)$ is nothing more than the negative frequency part of $X(\omega)$.

It would actually make more sense to plot $X_N(\omega)$ in the range $\omega = [-\pi/\Delta, \pi/\Delta]$, and this would involve taking the ‘top half’ of the vector X computed above, and moving it to the front of the vector. MATLAB can do this simply by using the `fftshift` command as follows

```
>> N = length(x);
>> ws = 2*pi/N;
>> wnorm = -pi:ws:pi;
>> wnorm = wnorm(1:length(x));
>> w = wnorm*fs;
>> plot(w,abs(fftshift(X)))
>> axis([-30,30,0,160])
```

which results in the diagram shown in figure 4. Actually, this diagram illustrates something more than just the use of the `fftshift` command via `plot(w,abs(fftshift(X)))`, it also illustrates the computation of an appropriate x -axis labelling of frequency. In particular, notice that the ‘sampling’ spacing of the DFT in normalised frequency as

$$\omega_s = \frac{2\pi}{N}$$

is calculated in the variable `ws`, and this is used to generate a vector `wnorm` which is a frequency axis in the normalised frequency range $[-\pi, \pi]$, which in turn is converted to a ‘real’ frequency range $[-\pi/\Delta, \pi/\Delta]$ by multiplication with the sampling frequency f_s .

From figure 4 it is now clear that the signal vector x shown in figure 1 actually contains a clear sinusoidal component at around 5 rad/s. To find this frequency accurately, the `ginput` command may be used as

```
>> [freq,amp]=ginput
```

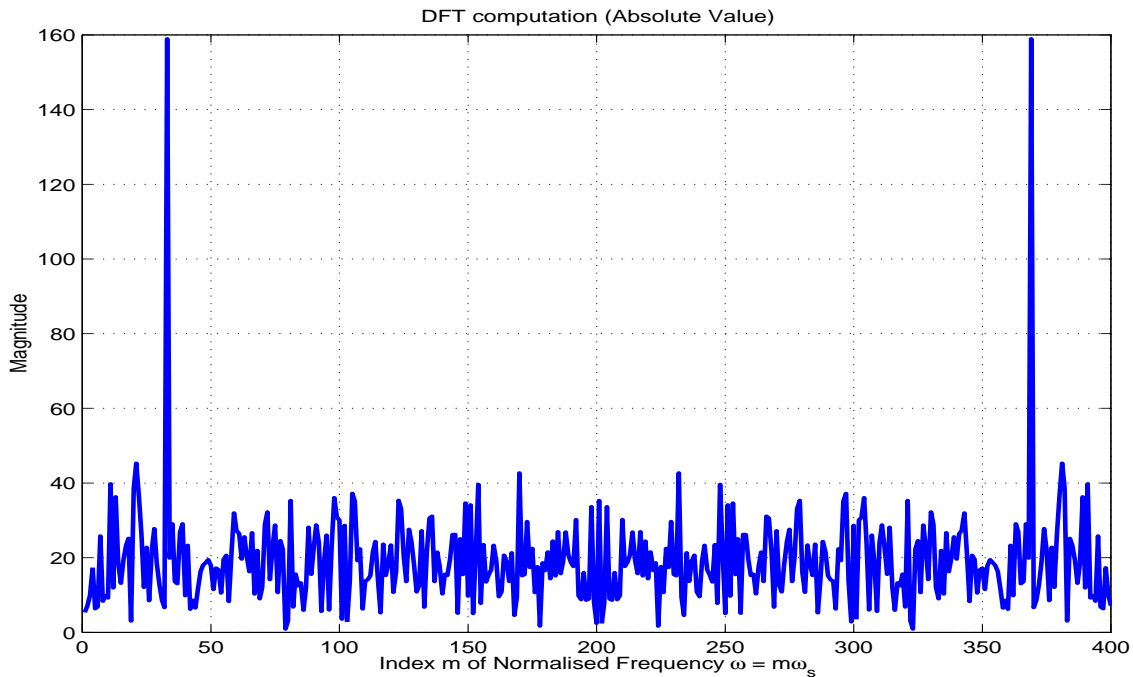


Figure 2: A plot of the DFT magnitude $|X_N(m\omega_s)|$ for the signal x shown in figure 1.

```
freq =
```

```
4.9770
```

```
amp =
```

```
159.0643
```

which creates mouse-controllable cross-hairs on the plot shown in figure 4 that can be placed at the tip of the spectral peak. When the left mouse button is then clicked, and the return key hit, the x and y axis values of the cross-hair position are reported.

This confirms our estimate of the sine-wave frequency being at 5 rad/s. To estimate its amplitude, we need to remember that an N sample data record of an amplitude A sine/cosine wave, results in a DFT peak of $AN/2$, and therefore the estimated underlying sine-wave amplitude is

```
>> amp*2/N
```

```
ans =
```

```
0.7953
```

MATLAB also has an FFT function `ifft` that, in a numerically efficient (fast) fashion, performs the inverse DFT computation

$$x_k = \frac{1}{N} \sum_{m=0}^{N-1} X_N(m\omega_s) e^{jm\omega_s k}. \quad (2)$$

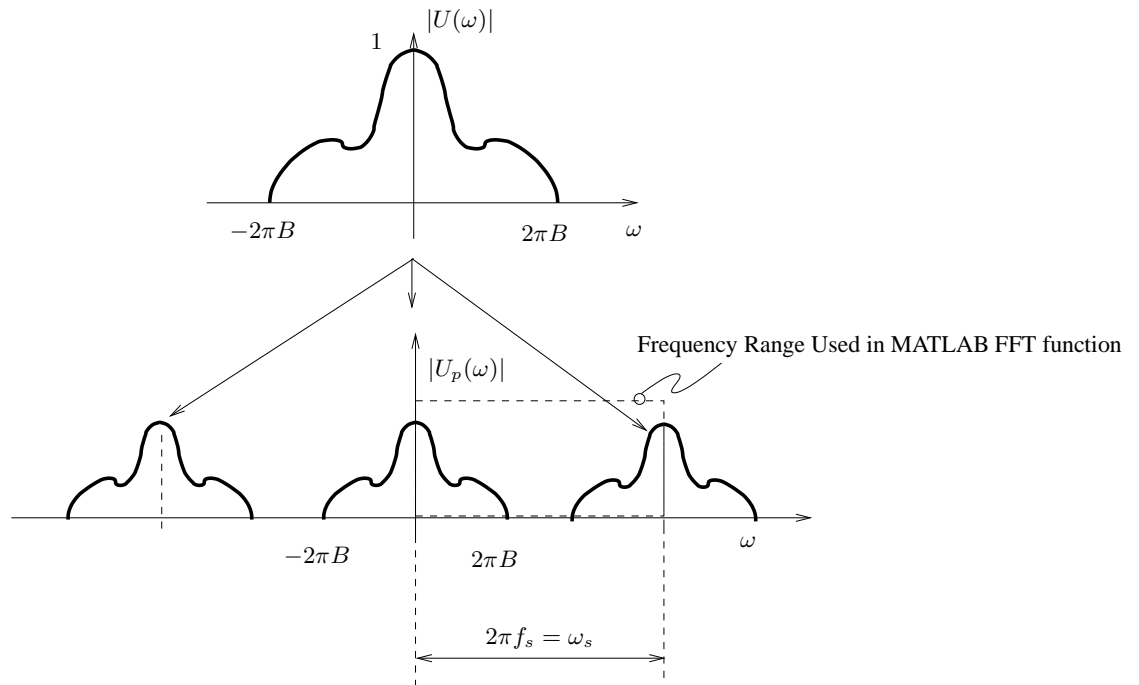


Figure 3: The frequency range of the DFT computed by the MATLAB FFT function is $\omega \in [0, 2\pi/\Delta)$

It could be used to perform filtering on the signal shown in figure 1 in order to recover the sinusoidal component that figure 4 indicates is present in the signal, but is buried in noise.

This filtering could be achieved by using the filtering vector

```
>> fil = [zeros(1,160), ones(1,15), zeros(1,50), ones(1,15), zeros(1,160)];
```

which, as shown in figure 5 by execution of the command

```
>> plot(w, abs(fftshift(X)), 'b', w, 160*fil, '-.r');
```

is one which, if multiplied by the vector representing $X_N(\omega)$ would retain the sinusoidal signal component of the spectrum, while eliminating the remainder that is due to noise. This filtering can be performed by execution of the commands

```
>> Xf = fil.*fftshift(X);
>> xf = ifft(fftshift(Xf));
>> plot(t, real(xf))
```

with the results shown in figure 6. Some important points to draw from the above commands are that firstly, the filtered spectrum vector Xf needs to have the function `fftshift` applied to it before being passed to the `ifft` function since the `ifft` function expects the frequency axis of figure 3 to be associated with its input; that is it expects the negative frequency components to be stacked in Xf after the positive frequency ones.

Secondly, although the result of the `xf = ifft(fftshift(Xf))` command should be real since Xf is symmetric, the limited numerical precision of the computing platform being used will result in small imaginary components of the order of 10^{-15} being present, and these need to be discarded by using the `real` command.

The results of the filtering are shown in figure 6, where a 5 rad/s tone of average amplitude around 0.8 is shown.

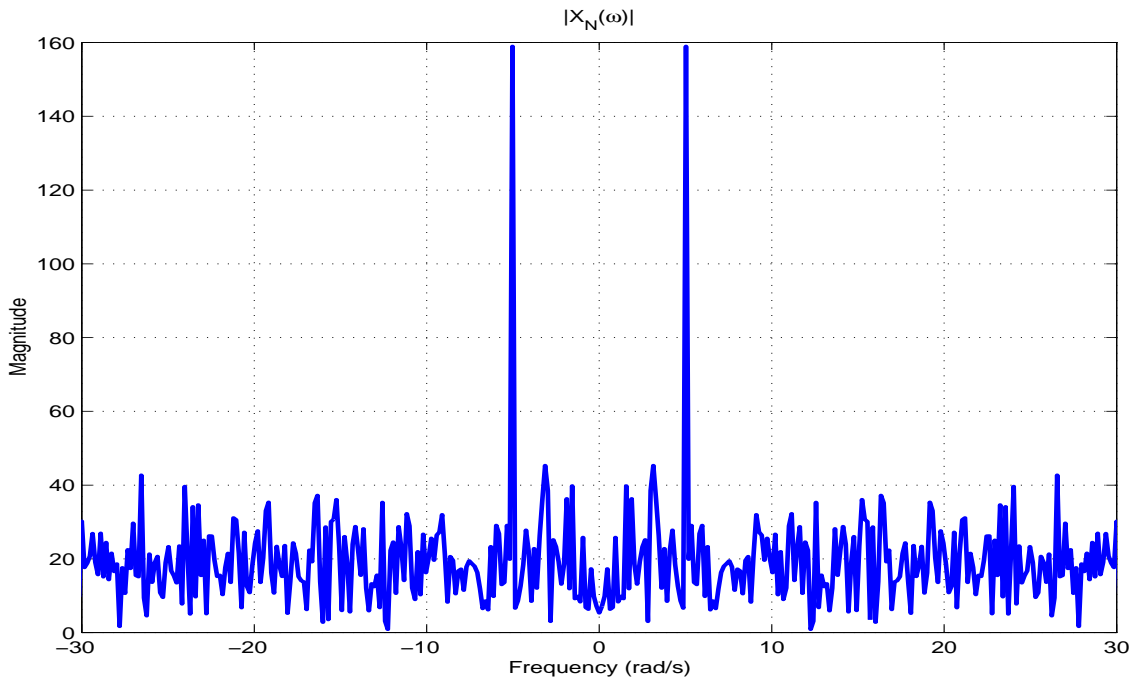


Figure 4: DFT magnitude $|X_N(m\omega_s)|$ plotted over the frequency range $[-\pi/\Delta, \pi/\Delta]$.

2 Spectral Resolution and Windowing

Now suppose that we have available a new 40 second data record x as shown in figure 7(a). This looks very similar in nature to the original data record shown in figure 1, and indeed the DFT analysis performed by

```
>> N = length(x);
>> ws = 2*pi/N;
>> wnorm = -pi:ws:pi;
>> wnorm = wnorm(1:length(x));
>> w = wnorm*fs;
>> X=fft(x);
>> plot(w,abs(fftshift(X)))
>> axis([-30,30,0,500])
```

produces a spectral plot shown in figure 8 which indicates (as the plot did in figure 4) a sine component at 5 rad/s. However, the data record shown in figure 7(b) is of the same signal, except observed for a longer 120 second duration, and the spectral analysis of this signal gives the spectral plot shown in figure 9 (the command `axis([-10,10,0,500])` has been used to ‘zoom in’ on the spectral peak region) This indicates not one, but two spectral peaks that are close together at approximately 4.7 and 5 rad/s. Why are they noticeable in the spectral analysis of the long data-record signal, but not in the short data-record one?

The answer is to do with windowing. Remember from lectures that the finite data record DFT $X_N(\omega)$ is related to the infinite data record sampled spectrum $X_p(\omega)$ according to a convolution

$$X_N(\omega) = \frac{1}{2\pi} [U_p \otimes W](\omega)$$

where $W(\omega) = \mathcal{F}\{w\}$ is the Fourier transform of the effective ‘window function’ $w(t)$ which models the truncation of the infinite record to the finite one by multiplication.

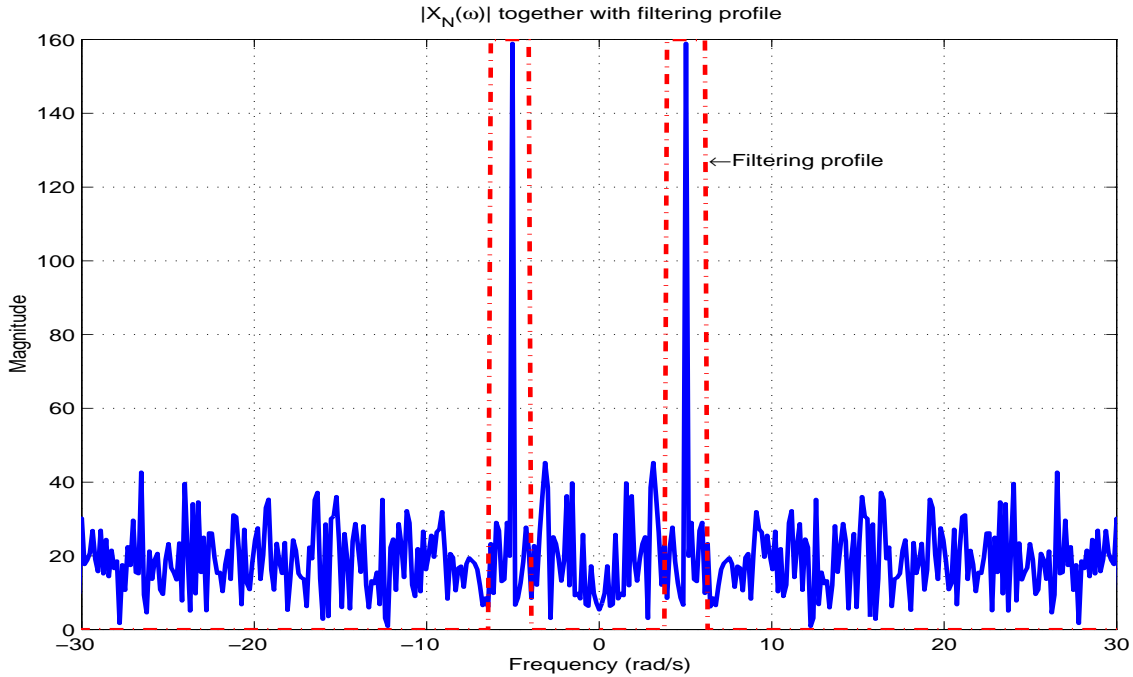


Figure 5: DFT magnitude $|X_N(m\omega_s)|$ together with proposed filtering profile.

Now in the cases shown in figures 7(a),(b), the truncation function $w(t)$ is simply of rectangular shape and of the form

$$w(t) = \begin{cases} 1 & ; t \in [0, T] \\ 0 & ; \text{Otherwise} \end{cases}$$

This means that $|W(\omega) = \mathcal{F}\{w\}|$ will be of the form

$$|W(\omega)| = \frac{2 \sin \omega T/2}{\omega T}$$

which is a sinc-type function that has slowly decaying ‘side-lobes’ of effective width approximately $2\pi/T \times 4$ rad/s. For the short data record with $T = 40$, the side-lobe width is

$$\frac{2\pi}{40} * 4 \approx 0.63 \text{rad/s}$$

which is greater than the sinusoid separation of 0.3 rad/s, and hence the ‘smearing’ interaction of the side-lobes blends the two peaks into one. However, for the longer data record of $T = 120$ seconds

$$\frac{2\pi}{120} * 4 \approx 0.21 \text{rad/s}$$

which is less than the sine-wave frequency separation, and hence the ‘smearing’ action does not blend the peaks and we can resolve them in figure 8

The frequency resolution can be enhanced by use of a window function other than a rectangular one and which has a Fourier Transform with faster decaying side-lobes than the sinc-type function. MATLAB implements more sophisticated sorts of windows via the functions `triang` (triangular window), `hamming`, `hanning` and others. T

For example to generate a hamming window of length N samples, the command

```
>> win = hamming(N);
```

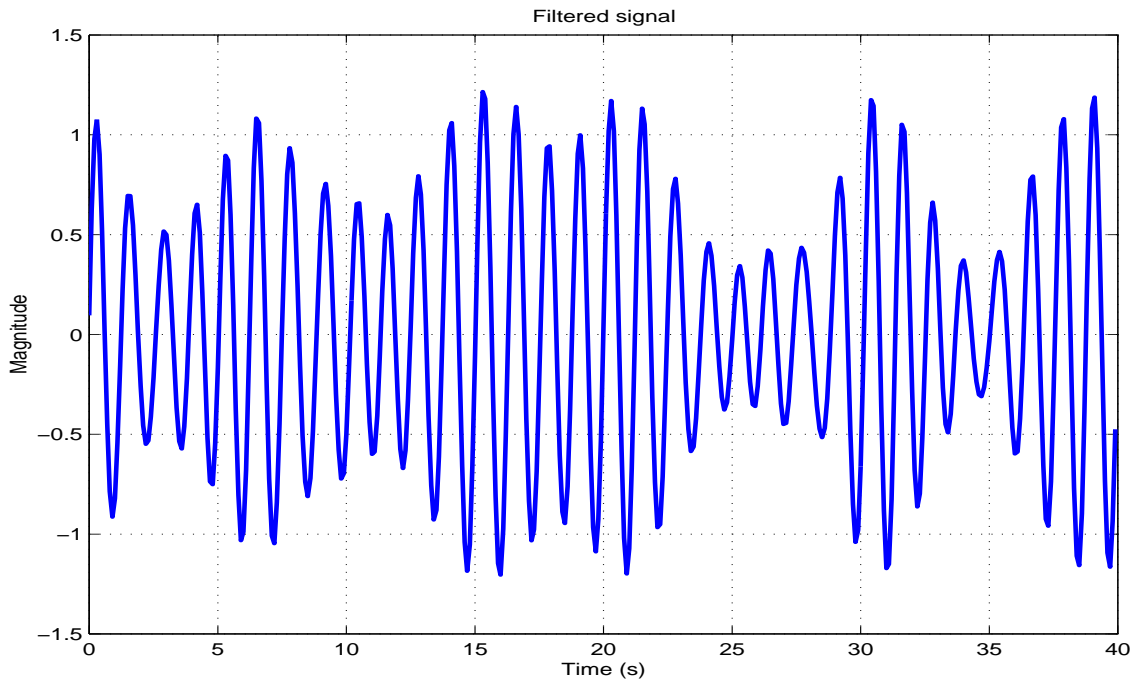


Figure 6: *Filtered version of signal shown in figure 1 with the filtering achieved by use of a DFT/IDFT concatenation.*

which produces a vector `win` which when plotted via `plot(win)` results in the diagram shown in figure 10. This can then be used to weight or window the data as

```
>> xw = win(:).*x(:);
```

(note the use of the colon notation `x(:)` to ensure both vectors are of column type before point-wise multiplication) before the DFT is calculated and plotted via

```
>> xw = win(:).*x(:);
>> Xw=fft(xw);
>> plot(w,abs(fftshift(Xw)))
>> axis([-10,10,0,80])
```

to produce the diagram shown in figure 11. Notice that with the use of this appropriate choice of Hamming window, the spectral resolution enhanced to the point where the two sine-wave components at 4.7 and 5 rad/s can be resolved.

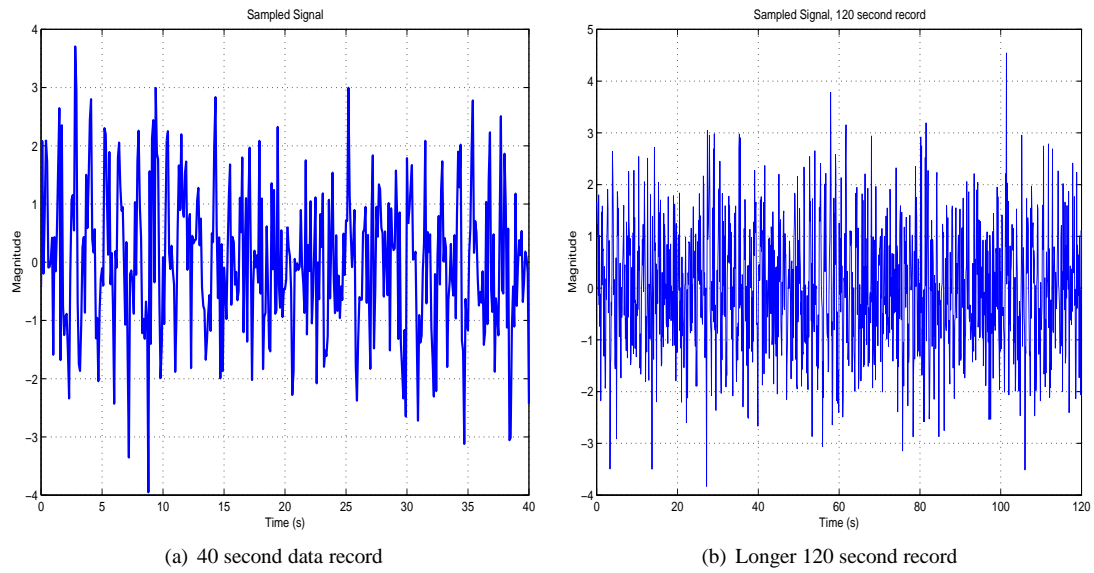


Figure 7: A short 40 second, and a longer 120 second record of the same signal.

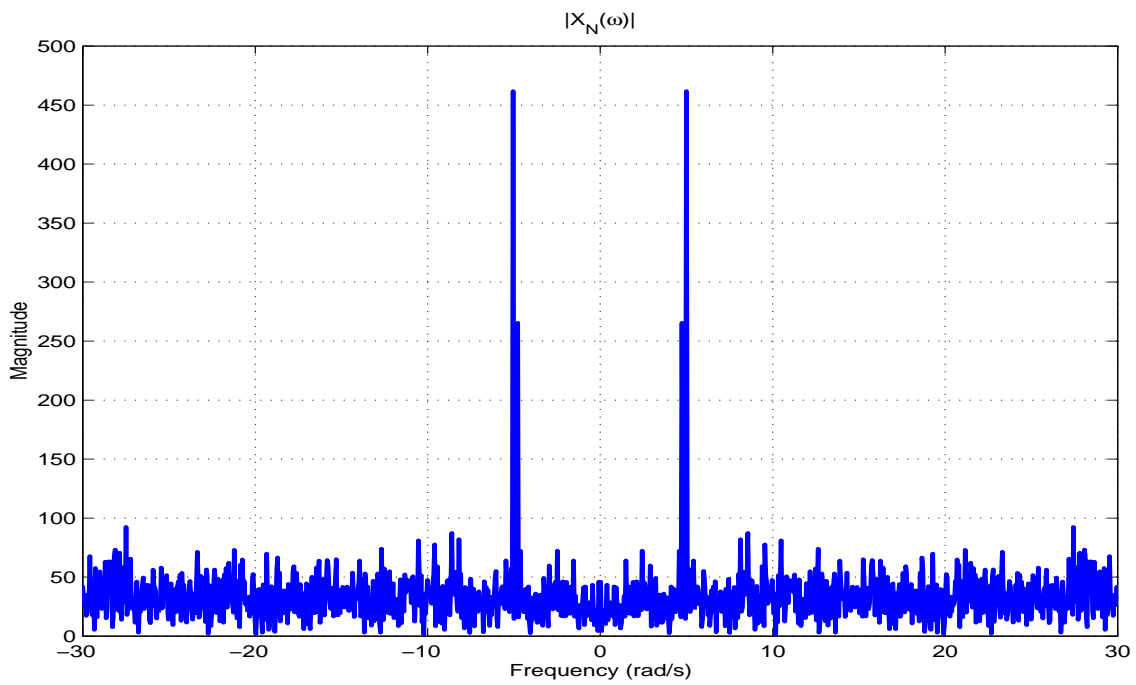


Figure 8: Spectrum of signal shown in figure 7(a).

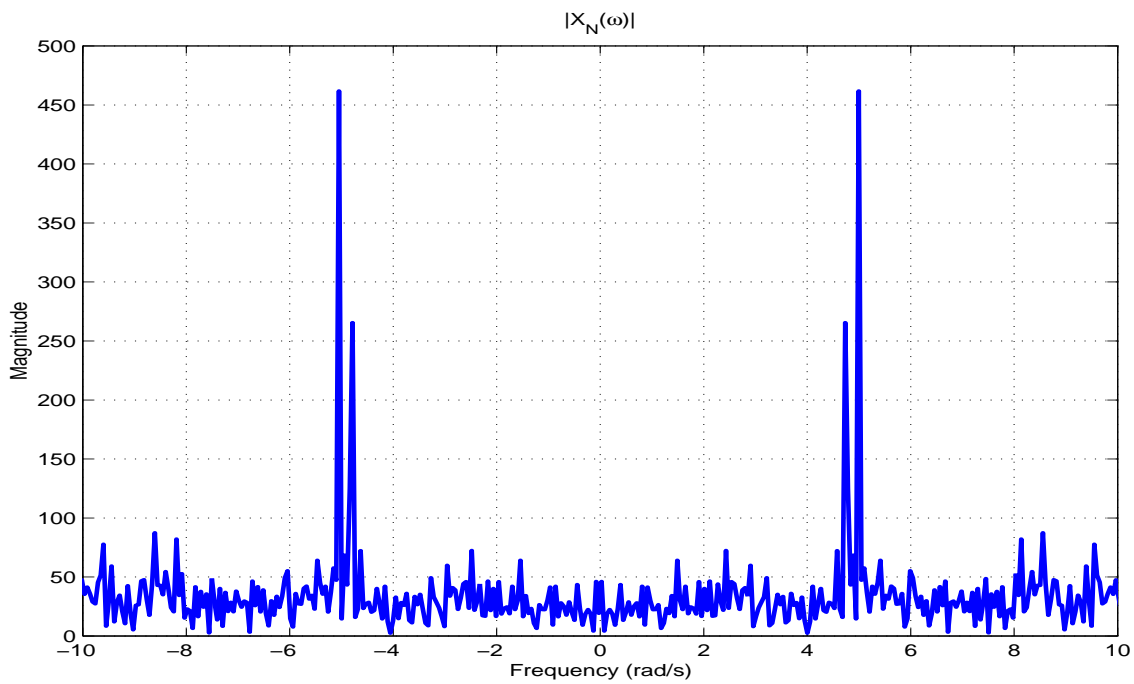


Figure 9: *Spectrum of signal shown in figure 7(b).*

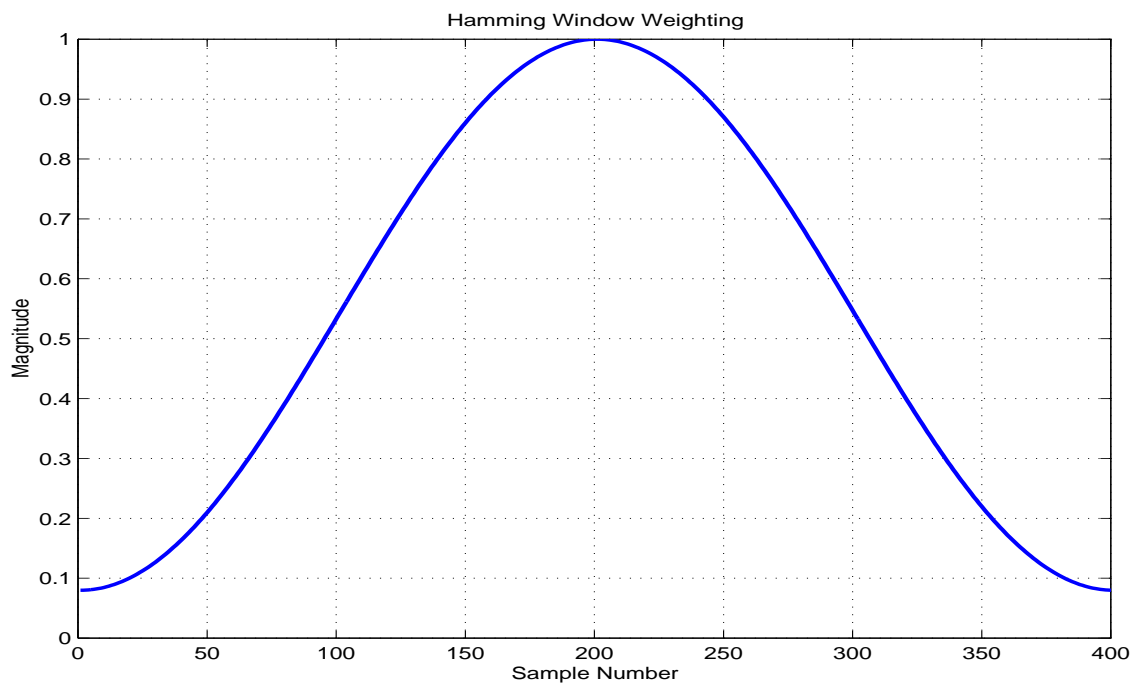


Figure 10: *Hamming Window Weighting.*

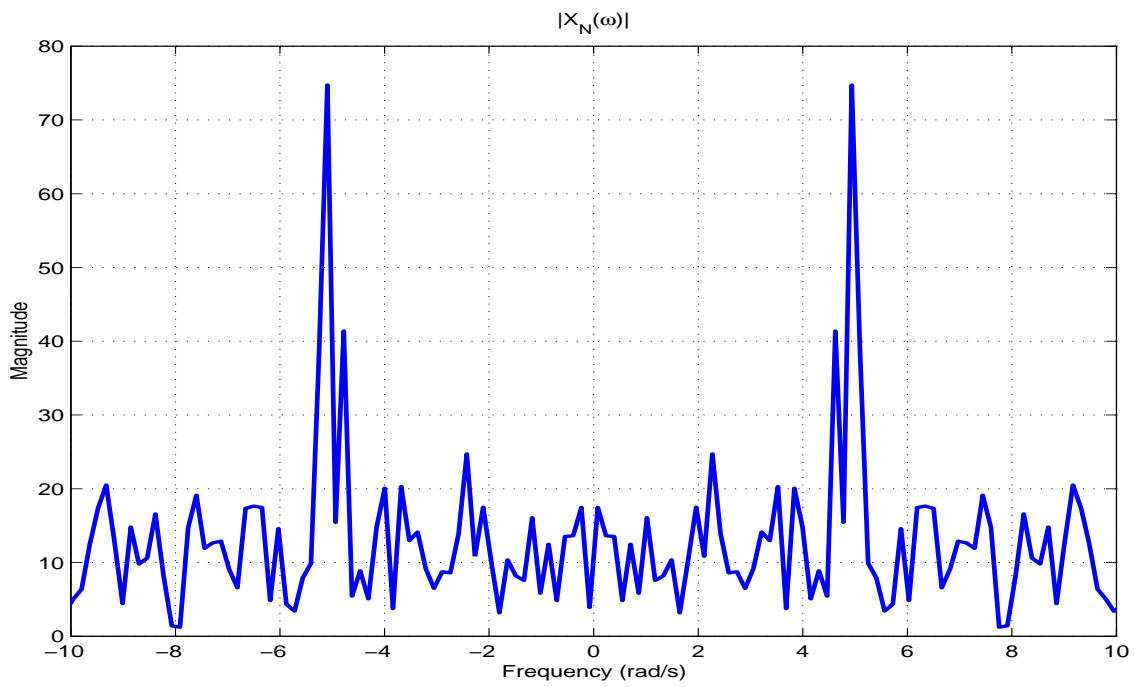


Figure 11: *Spectrum of Short 40 second data record when Hamming window is used.*