ELECTRONOTES 222

Newsletter of the Musical Engineering Group

1016 Hanshaw Road, Ithaca, New York 14850

Volume 23, Number 222 June 2014

FFT – INTERPOLATION IN TIME AND FREQUENCY

-by Bernie Hutchins

INTRODUCTION

There are many methods of interpolating sequences, from those as simple as just drawing a straight line, to sync (low-pass) convolution, and so on. Depending strongly on the amount of data to be processed (is this a single graph for a paper to be published – once and done; or are we designing a CD player to work in real time – continuous data), we may be concerned with computational throughput. One method of interpolation is to use the FFT [1,2] where the interpolation is in time, but it can just as easily be in frequency.

Because we so much associate deliberate interpolation with the time domain, we may not have recognized that we have almost certainly interpolated in the frequency domain many times – virtually every time we have computed a digital filter's frequency response – the Discrete Time Fourier Transform (DTFT), which we have mentioned as just a Fourier Series (FS) for a periodic function of frequency [3]. Here we will be reviewing many issues with the FFT in addition to its use in interpolation, and will be indicating how this leads to familiar FS ideas.

The FFT (Fast Fourier Transform) is just a fast algorithm for computing the DFT (Discrete Fourier Transform). The computation is exact, not an approximation. The FFT can also be used to approximate a CTFT (Continuous Time Fourier Transform – the equations with integrals) and the Fourier Series (FS), as well as give samples of the DTFT (Discrete Time Fourier Transform – close relative of the DFT). Here we are first concerned with the FFT as it is often thought of as defining a spectrum, or as used for interpolation.

The famous equations for the DFT and Inverse-DFT (IDFT) are the finite length summations:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j(\frac{2\pi}{N})nk} \qquad k = 0,1,2,\ldots(N-1) \qquad (1a)$$

and:

$$x(n) = \left(\frac{1}{N}\right) \sum_{k=0}^{N-1} X(k) e^{j(\frac{2\pi}{N})nk} \qquad n = 0,1,2,\ldots(N-1) \qquad (1b)$$

We note the near perfect symmetry of this pair – the inverse has a (1/N) multiplier and a minus sign. Another notable fact is that these equations represent N linear equations in N unknowns. There are N (most often real) values of a time sequence x(n) and they are related to N (generally complex) values of a frequency description X(k) by either equation. The coefficients of these linear equations are the complex exponentials, not real numbers, and all in a nice sequence so they don't look like the coefficients of linear equations from high-school math. But it is true that they relate N values of x(n) to N values of X(k).

So here are the main questions for this note: **(1)** Is it true that if we have a length N time signal that we only get (approximately) N/2 frequency points? [More specifically, if N=7, the frequency values k=0, 1, 2, and 3 are "good" and if N=8, the points k=0,1,2,3,4 are "good"] **So did we lose half the information?** Not really because the input x(n) could have been complex (two real numbers for each n). **(2)** What happens when we interpolate to larger values of N? **Do we get extra information?** Not a chance, although the human looking at the data may get an easier idea what the FFT "had in mind". The information is NOT NEW if you can generate it, at any time, from the old information. So you don't lose, and you don't gain. Fair enough. **(3) How does FFT interpolation relate to the FS and the DTFT?** For the record, here are the FS and the DTFT:

$$f(t) = \sum_{k=-\infty}^{\infty} c(k) e^{2\pi jkt/P} \qquad (FOURIER\ SERIES) \qquad (1c)$$

$$X(\omega) = \sum_{n=-\infty}^{\infty} x(n) e^{-jn\omega T} \qquad (DTFT) \qquad (1d)$$

N Equations in N Unknowns

At some point (high school algebra?) we learn to set up and solve a set of simultaneous linear equations, often involving prices and numbers of things purchased such as candy bars and chewing gum. At least several ways of solving these are eventually offered (Gaussian elimination, matrix inversion, graphing). As mentioned, the summation formulas in equations (1a) and (1b) are exactly this problem. Indeed, it is usual to represent them in matrix form such as:

$$\underline{X} = D_N \underline{x} \qquad (2a)$$

$$\underline{x} = D_N^{-1} \underline{X} \qquad (2b)$$

where the element D_{nk} of the matrix D_N is:

$$D_{nk} = e^{-j(2\pi/N)nk} \qquad (2c)$$

The simplest N equations in N unknown problem is likely just that of saying that we are thinking of two numbers whose sum is 8 and whose difference is 2. In your head you immediately see that one of the numbers is 5 and the other is 3. Possibly you visualize the problem and solution as a flow graph such as Fig. 1a.

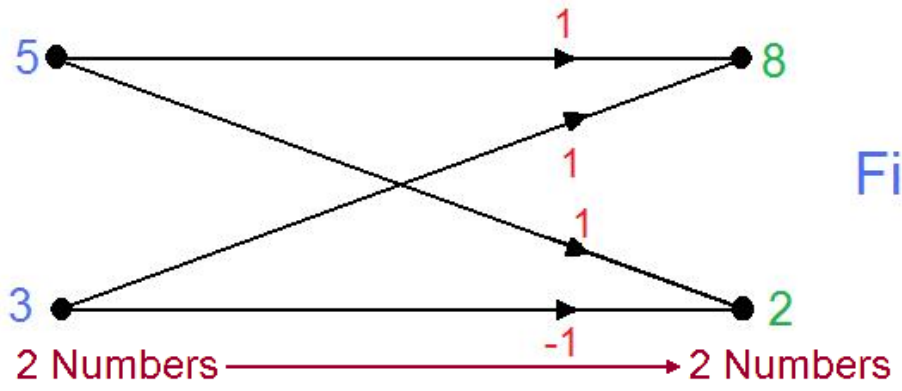


Fig. 1a

This is interesting because the flow graph happens to be a length N=2 DFT. This would be for:

$$D_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad (3)$$

Inverting D_2 by the usual matrix inversion we have:

$$D_2^{-1} = \left(\frac{1}{2}\right) \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad (4)$$

with a corresponding flow graph of Fig. 1b. We note that the original numbers are returned in Fig. 2b. This is a simple illustration of "conservation of numbers" (of information). Both Fig. 1a and Fig. 1b have two numbers in, and two numbers out.

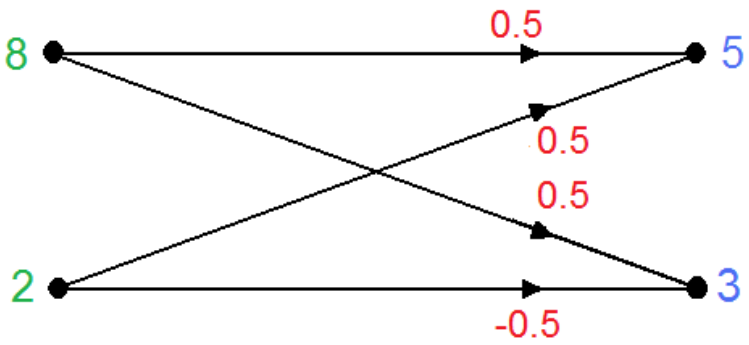


Fig. 1b

The path multipliers (often called “twiddle factors”) that appear in FFT flow-graphs (often called “butterflies” for obvious reasons) are not so simple in the general case. Above for N=2 we have all real multipliers. For N=4 all the multipliers are either ±1 or ±j. But it is perhaps N=3 that shows the first general case of some fully complex multiplies.

Real Time Data → Complex FFT (N=3)

In going to N=3 (Fig. 2a), if we continue to suppose that x(0), x(1), and x(2) are real, we see that X(0) will continue to be real (the sum of the three inputs), but X(1) and X(2) will in general now be complex. (There are input sequences, e.g., [1 2 2] where the FFT remains all real.) Thus we have the prospect here of the three input numbers being represented by five output numbers, X(0), and the real and imaginary parts of X(1) and X(2). A simple example will show better why this is not really so (Fig. 2b). In Fig. 2b we see that the output is represented by just three actual numbers: 6, -3/2, and $\sqrt{3}$ j/2. Or, just observe that X(2) is the complex conjugate of X(1). Indeed it is well established that for a real time

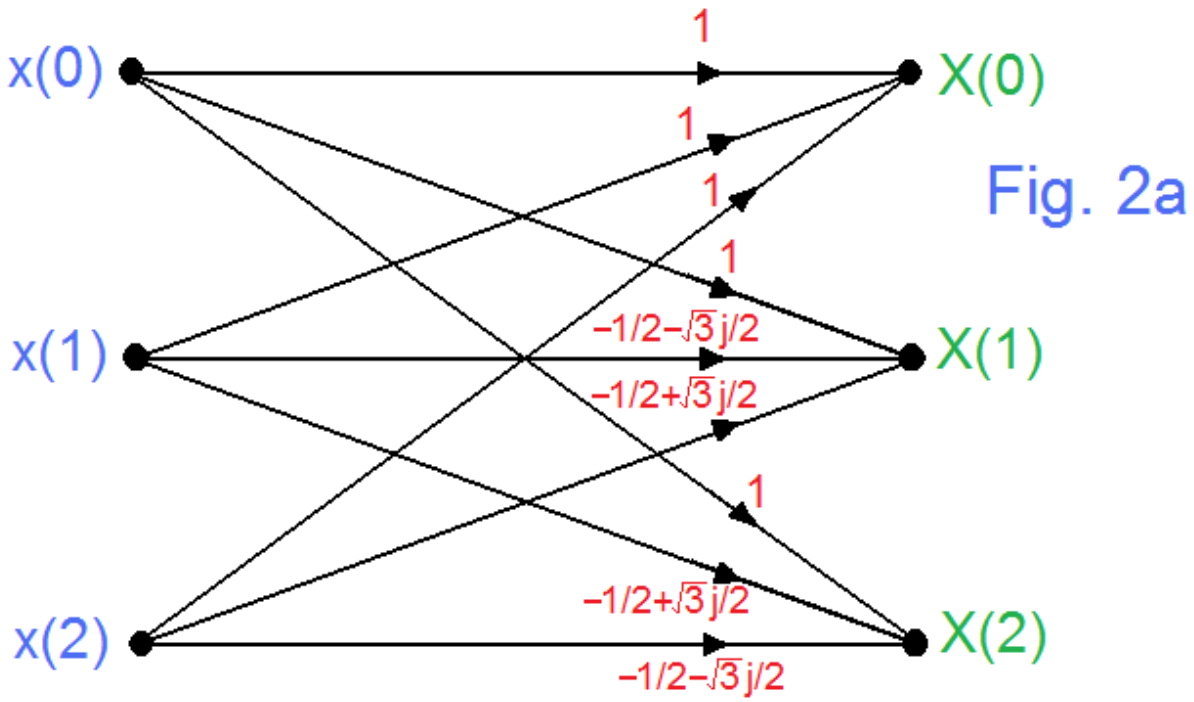
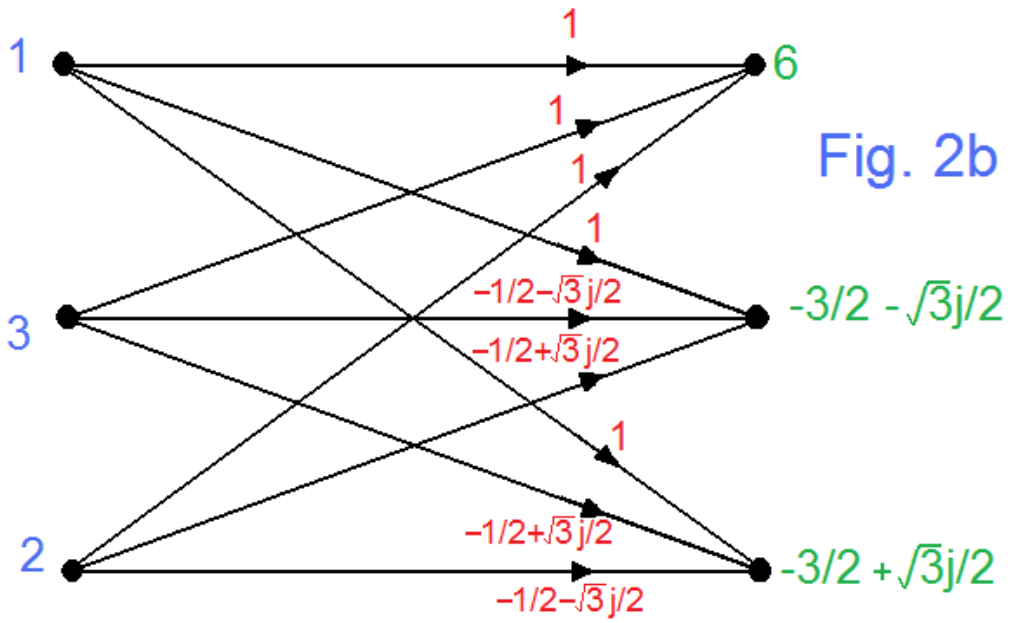


Fig. 2a



sequence, X(N-k) = X(k)* where the * represents a conjugate. So it is because the sequence is real and the resulting symmetries that we have reason to suspect that only the lower half of the spectrum is useful. In fact, for a real input, the real part of the output is even symmetrical, the imaginary part of the output is odd symmetrical, and the magnitude is even symmetrical. So is it the case that three input numbers goes to just two? No, we count a complex number as composed of two real numbers and a j.

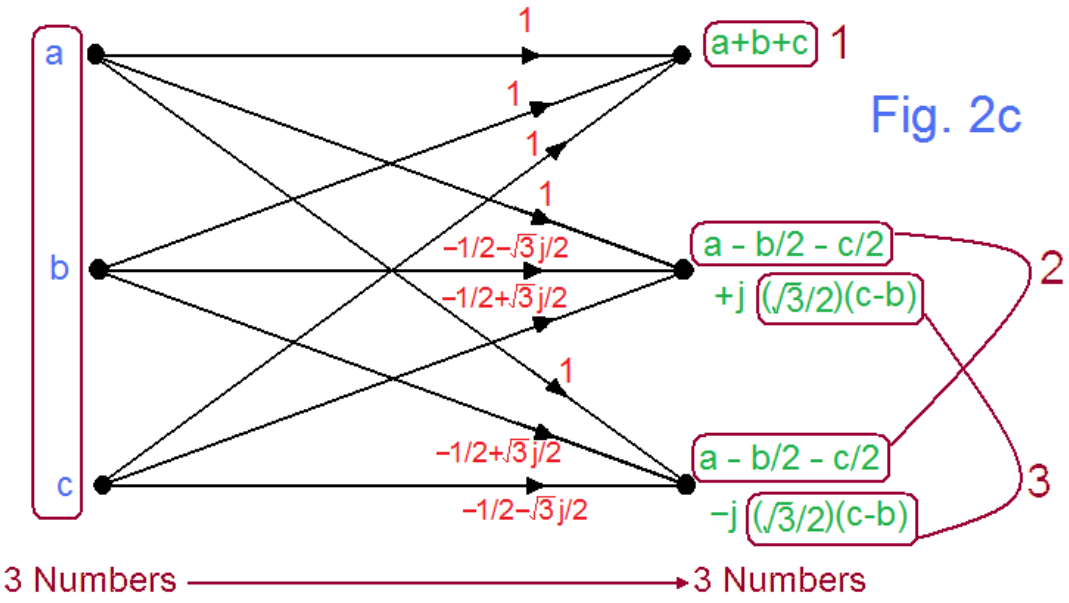


Fig. 2c repeats this example with general real inputs a, b, and c, and we note that the outputs are complex conjugates for X(1) and X(2). Likewise, for other N and for real time signal, the magnitude FFT is going to be symmetric with |X(N-k)| = |X(k)|. The DFT underperforms because we starved it at the input. This notion was addressed in the past

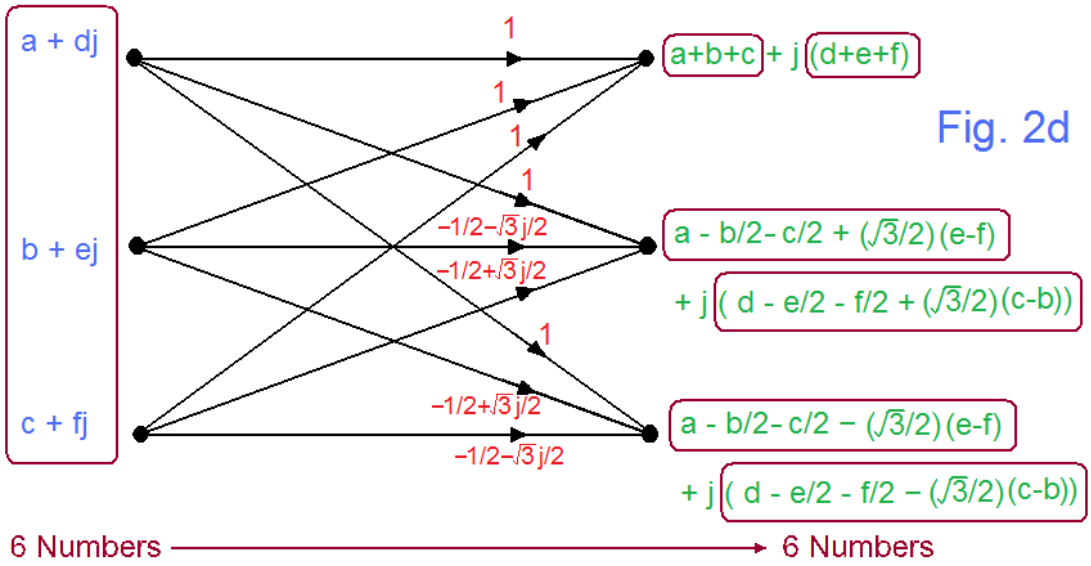


Fig. 2d

when it was suggested that you could do two FFTs at the same time with one FFT run. You could make one time signal the real part of the input, and a second time signal could be added as an imaginary part. Run the FFT of the now complex signal, and take the results apart with symmetry operations. This never caught on (except as a student homework exercise). It would seem that the saving by using this would be immense if it were DFTs that were being computed. But - likely the overhead "bookkeeping" might be comparable to just doing a second FFT.

What we have shown to this point is the origin of the complex output from a purely real input, and the symmetry properties that result in a redundancy in the output for a purely real input. This has shown a conservation of information (conservation of numbers) such that no information is added or lost. It is true that for a purely real input (almost always?) we make use mainly of half the FFT output in the form of its magnitude. It is also true that except for educational purposes and program writing [4], we don't make much direct use of butterflies of the type we have shown.

A Whole Bunch of Example Plots

When we compute the FFT of a time sequence and want to see plotted results, (which we intend to interpret as a spectrum in the frequency domain), we are dealing with discrete sequences, and often will plot the results as spikes or stems (lollypops) instead of as continuous functions. This decision to plot as discrete points is very important. (While it is true that for a very dense set of discrete samples, the result as plotted may look continuous, we generally understand this fact as relating to a plotting limitation). By showing discrete points, we are acknowledging an ignorance of what might be between the samples (when that notion of "between" has a meaning). We avoid, for example, connecting discrete points with a straight line – the idea being presumptuous. Still we traverse the FFT landscape knowing that we are involved here with discrete samples of something that might actually be associated with a continuous curve. What might that be?

This means we want to look at ideas of interpolation. That is, making various specific assumptions, what lies between the given samples? First however, we need to be very clear on the ideas of linearity (superposition), symmetry, complex sequences, and bandlimiting. So next we will present a series of graphs showing time/frequency sequences based on the FFT. The first of these examples are elementary, while some later ones probably are seldom shown. The program we use (see appendix) will plot five graphs for each example: the real and imaginary parts of the time sequence (upper left and right), the magnitude of the FFT (middle left), and the real and imaginary parts of the FFT (lower left and right). Most commonly we see the spectrum of a real signal presented as the magnitude of the FFT, and this is often reasonable and useful.

We begin with the simplest case – a length N=30 signal that is purely real, even symmetric, and comprised of exactly three full cycles of a cosine (Fig. 3a). These three cycles have a purely real FFT (bottom left) that is symmetric with respect to the middle (k=15) and is non-zero only for k=3, and at the upper side (corresponding to negative frequencies if you prefer) at k=N-3 = 27. The magnitude in this case is exactly the same as the real part. Quite lovely – but just a start.

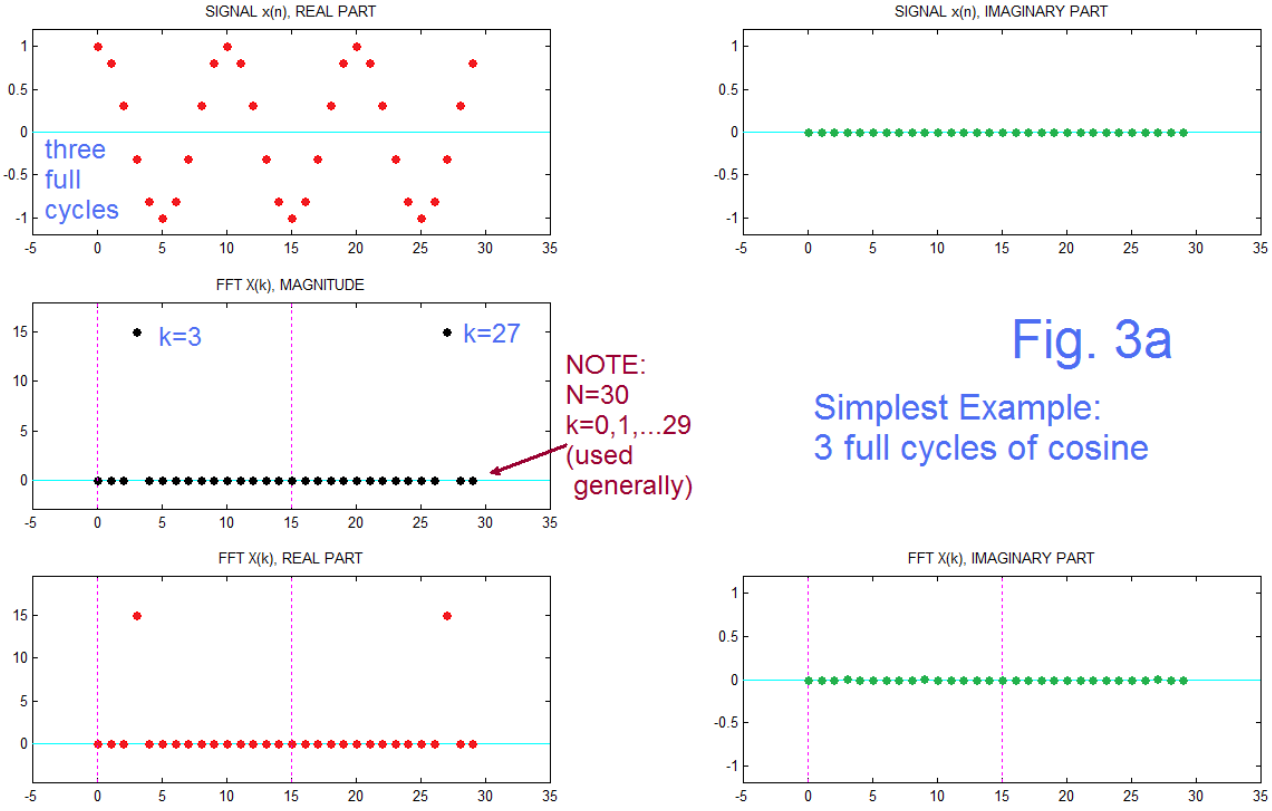
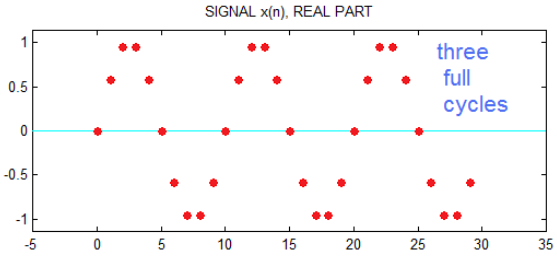


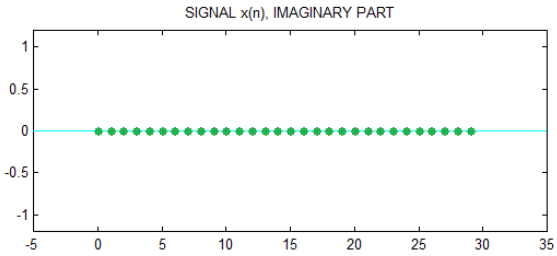
Fig. 3a

Simplest Example: 3 full cycles of cosine

As we noted above, the FFT (while real here) is in general complex. The FFT being linear, if we were to make this test signal purely imaginary (multiply it by j), we would have a purely imaginary FFT (see Fig. 4b ahead). A different way to get a purely imaginary FFT would be to change the symmetry of the input waveform. This is shown in Fig. 3b where

we use three cycles of a sine instead of a cosine. Keep in mind that we are talking about symmetry with respect to n=0, the waveform being always seen to be periodic.





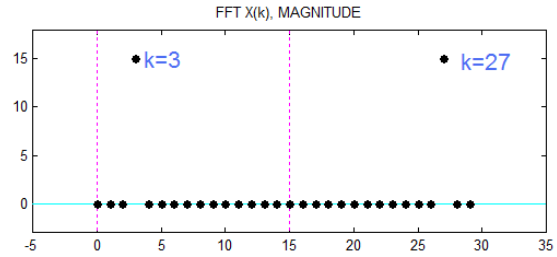
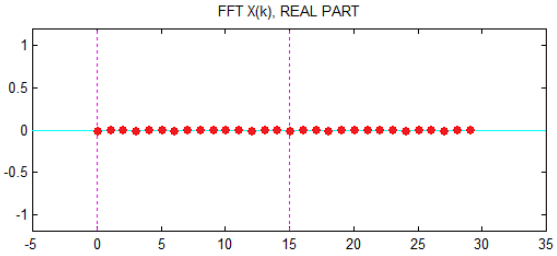
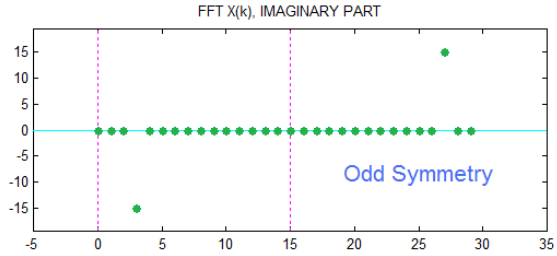
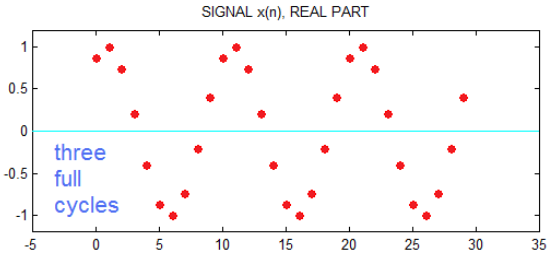


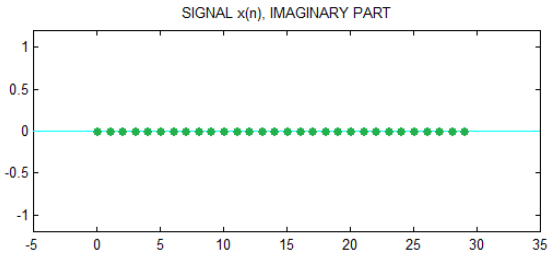
Fig. 3b

Changing to a sine phase makes the FFT pure imaginary
- still 3 full cycles









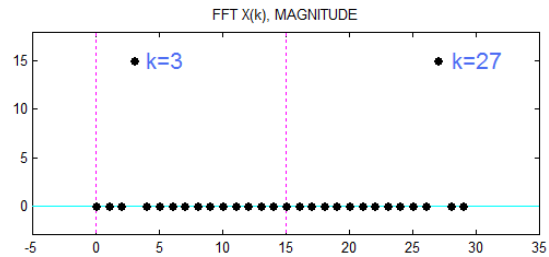
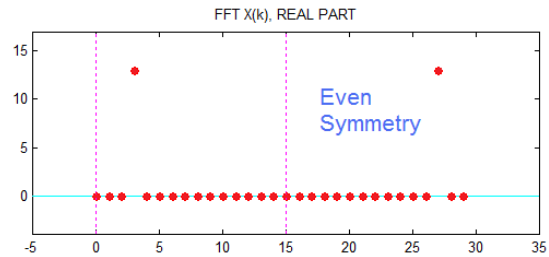
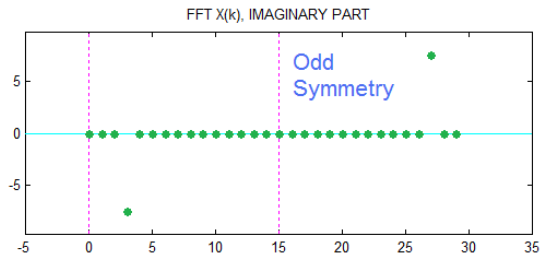
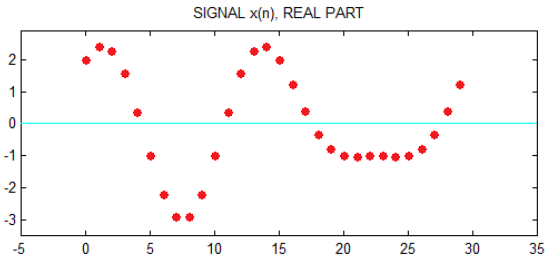


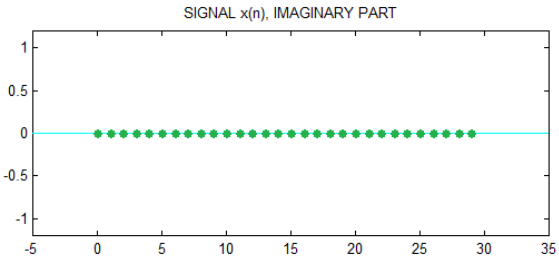
Fig. 3c

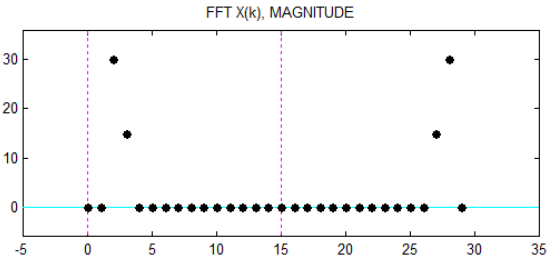
Starting first sample at 60^{o} makes FFT complex
- still 3 full cycles

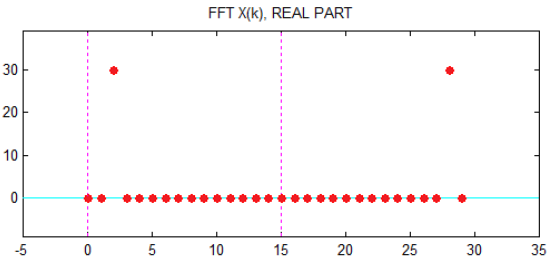












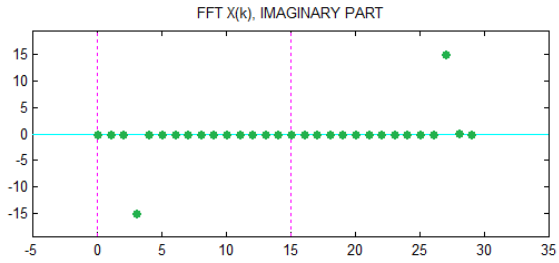
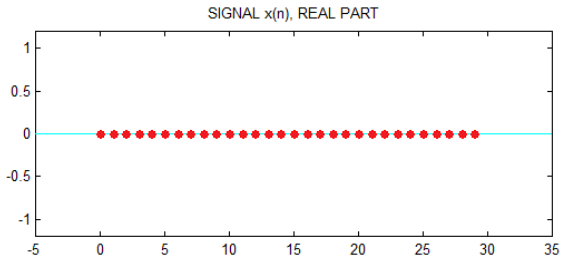
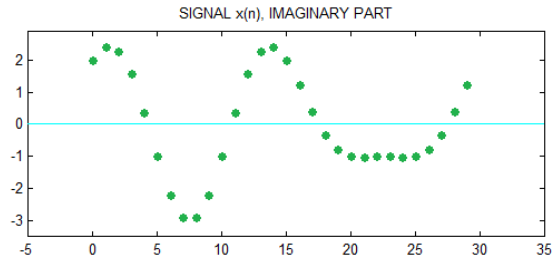
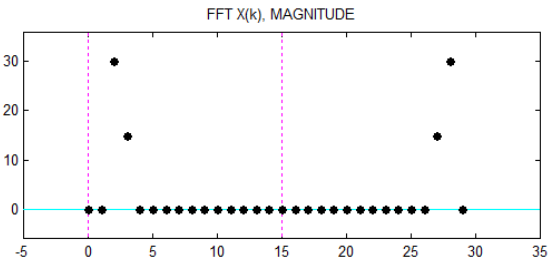


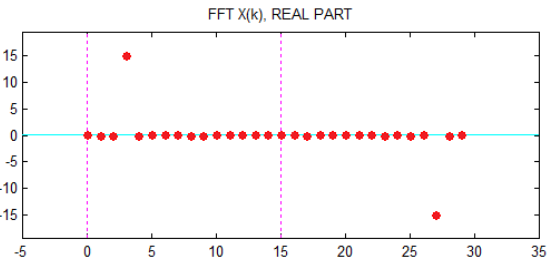
Fig. 4a

Linearity: x(n) is the sum of two cycles of a cosine (amplitude 2) plus three cycles of a sine (amplitude 1). FFT is complex.









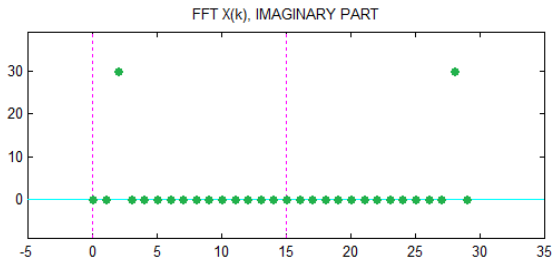


Fig 4b

Linearity: Same as Fig. 4a except x(n) is multiplied by j so it becomes purely imaginary. Symmetry conditions flip.

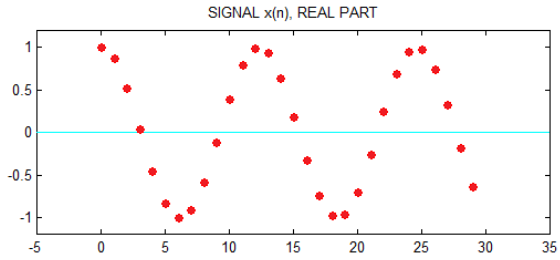
The key now is to look at an intermediate case between the sine and the cosine, and this is shown in Fig. 3c where the cosine sequence of Fig. 3a is multiplied by cos(30°) and added to the sine sequence of Fig. 3b as multiplied by sin(30°). Thus we have just done a phase shift of the sine by 60°. The particular angle is not important. We do note that this phase shift makes the FFT fully complex (unlike Fig. 3a and Fig. 3b), as it will be in general. Thus, Fig. 3c illustrates an example where we have a purely real sequence with no simple phase, so its FFT is complex, and the real part of the FFT is even symmetric while the imaginary part of the FFT is odd symmetric. Just as we have always been told. Note that we have still restricted our sequences to be an integer number of cycles, so we do not have anything general yet.

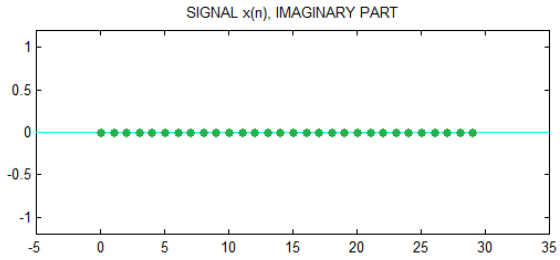
Figure 4a and Fig. 4b relate to the linearity of the FFT. That is, we can add inputs and add the corresponding outputs (Fig. 4a), and/or we can multiply inputs (and correspondingly, outputs) by any constants (Fig. 4b, the constant being j in this example). Here we have continued to restrict ourselves to signals with integer numbers of cycles. In Fig. 4a, one component of the input has cosine phase, amplitude 2, and two full cycles while the other component has sine phase, amplitude 1, and three full cycles. We see that the output result is simply the sum of what the individual inputs would be.

Fig. 4b, in addition to serving as a second illustration of linearity, also introduces a signal that is purely imaginary. The only thing different here is that the signal from Fig. 4a is multiplied by j. We note that the red input curve of Fig. 4a (purely real) simply moves to the right side and becomes the green input curve of Fig. 4b (purely imaginary). Note well that the green curve is still the same combination of components (including phase) as Fig. 4a. As should be true, the outputs FFT plots are simply reversed with respect to real and imaginary parts, while the magnitude is unchanged. We can correctly conclude that a purely imaginary signal has a real part that is odd symmetric and an imaginary part that is even symmetric. Both signals (Fig. 4a and Fig. 4b) have complex FFTs as a consequence of the phases chosen. So it is not the multiplication by j that creates a complex FFT. Multiplication by j just reverses the real and imaginary positionings.

Fig. 5 is an important example that needs to be here for comparison, but which primarily reminds us of the topic of “leakage” of the FFT that results when we do not have exact integer numbers of cycles – the general case. This we have looked at extensively [5]. We saw that even with an integer number of cycles, for a general phase we got back a complex FFT. Here too we get a complex FFT from a real signal (with the expected symmetry properties), but the spectral energy smears (leaks) into all the FFT “bins” (values of k). Quite arbitrarily we chose a frequency of “k” = 2.4414, between 2 and 4, and it is true that the peak at k=2 (the closest) is largest and the peak at k=3 is also large, with energy in all other values of k. This is well understood and is addressed practically when FFT is used as a spectrum analysis tool. Here our main interest is that at the middle, k=15, the FFT is not zero (there is energy at half the sampling frequency), and this relates to considerations of bandlimiting.

Fig. 6 is another example for our collection of reference cases. In this case we have made a random, complex signal. As one expects, this is quite the mess. Linearity still applies, so the output is the sum of the outputs that would have been obtained if the real





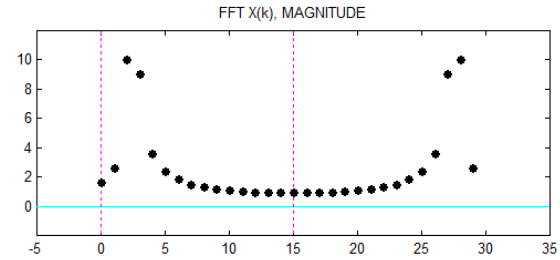
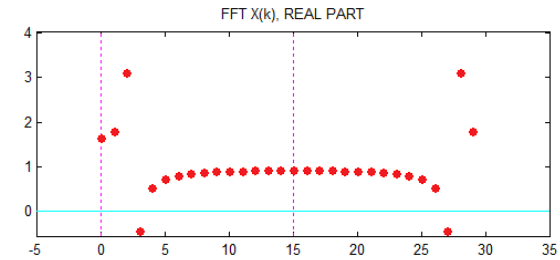
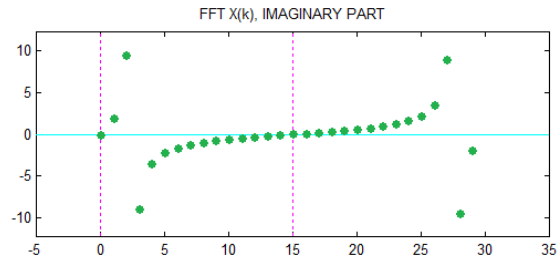
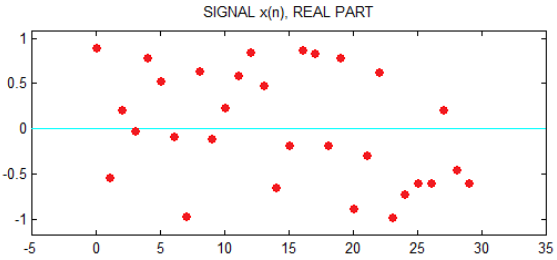


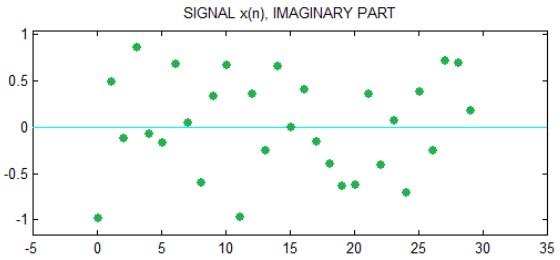
Fig. 5

Here we do not have an integer number of cycles but rather "2.4414 cycles" and the energy "leaks" into all the FFT "bins".









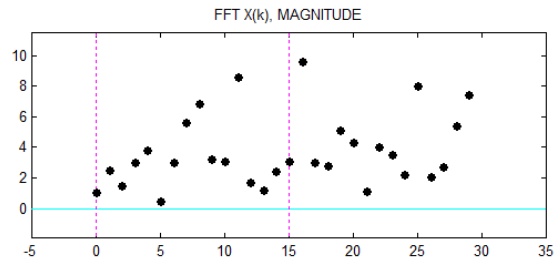
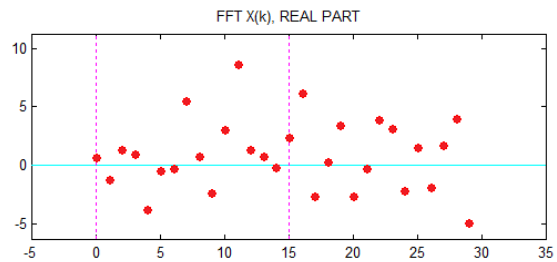
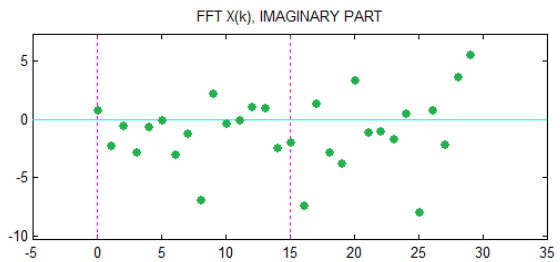


Fig. 6

Here we have a very general case of a fully complex signal with random real and imaginary parts. The magnitude is not symmetric.





and imaginary input components were separately transformed. Indeed, we can take the results apart using symmetry operations (as we mentioned for doing two FFT at once). One importance of the random signal is that it has energy at the middle, as in the "leakage" case of Fig. 5. Further, in our interpolation studies, we naturally wonder if we look between the given samples if we find more random samples, or if it is structured (it's structured of course). Finally, in as much as we wish to interpolate FFTs (sequences of samples in frequency) as well as time signals, and FFTs are generally complex, it will help us greatly if we understand the familiar time-domain case of a complex input signal.

Fig. 7 is also an example for comparison or reference. Here we have reduced the number of time points from 30 to just 29, but kept the cosine symmetry and a purely real signal. If we glance back to Fig. 3a it looks pretty much the same, unless we count the samples, or easier here – note the position of the magenta line in the FFT plots. The center of the FFT is now at k=14.5 instead of k=15. There is no energy there because there is no integer there. So the question of bandlimiting, or not, in Fig. 5 and Fig. 6, does not come up. So this potentially allows us to dodge the issue. In our interpolation procedure [1,2] we insert zeros in the middle of the FFT. If the middle is occupied by a non-zero value, what do we do? Well, we split it! This we are obliged to justify. If the middle were unoccupied, meaning properly bandlimited to strictly less than half the sampling frequency, then there is nothing to split. There is also no energy at the center frequency if it is not an integer, as in Fig. 7. Soon we will see what is "really" there if we interpolate in the time domain with the FFT!

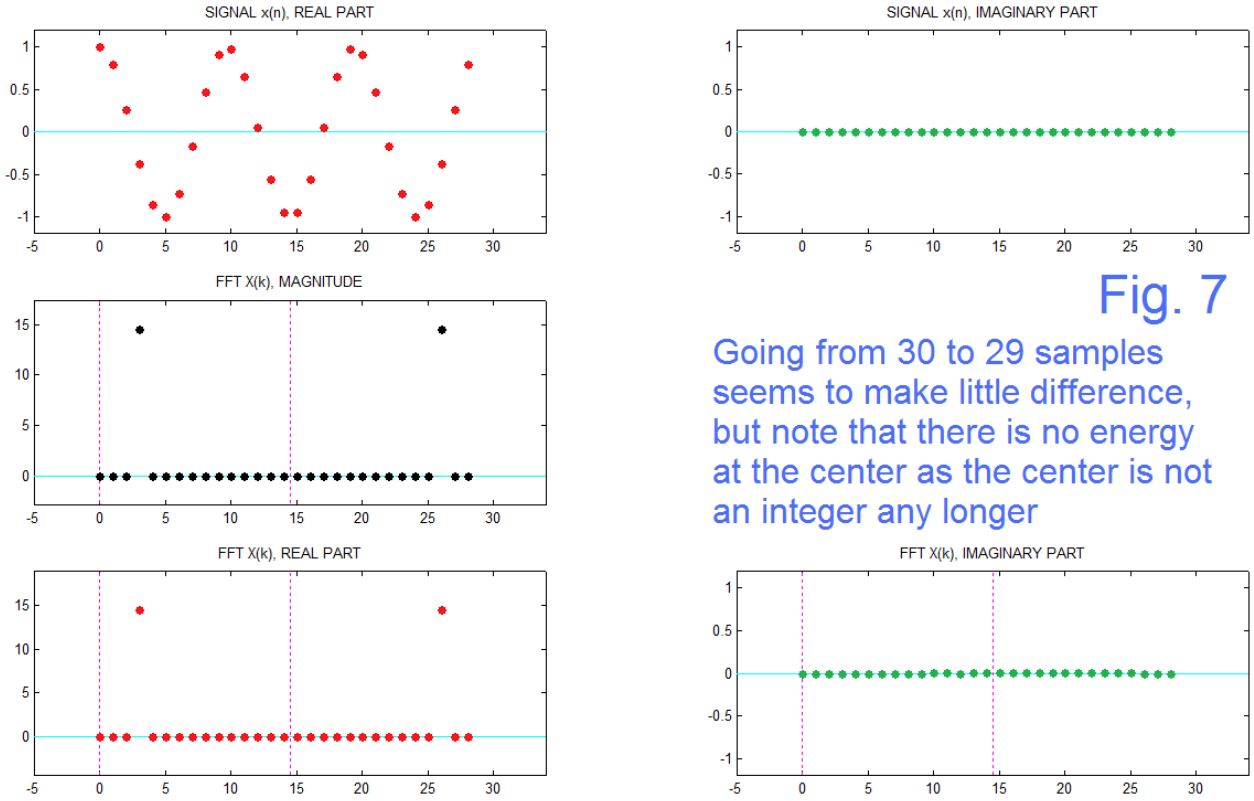
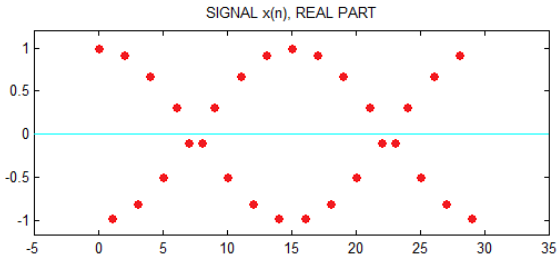


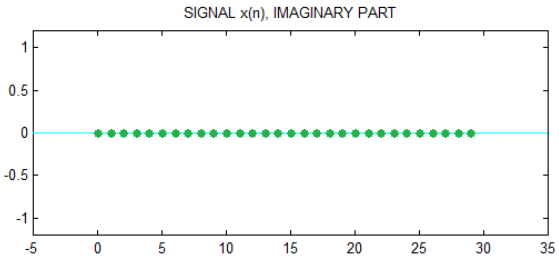
Fig. 7

Going from 30 to 29 samples seems to make little difference, but note that there is no energy at the center as the center is not an integer any longer

We next add to our collection some examples of what happens when the signal is chosen so that the energy is very close to or at the actual center of the FFT. Fig. 8a shows a modification of the familiar cosine case except instead of a frequency of k=2 and 3 we use a higher value of k=14, the center being at k=15. This is not really different from Fig. 3a in any other respect, but it sure looks different. This is a frequency that is very close to half the sampling frequency and it looks a lot like "beating" in the time domain [6,7]. It looks like there should be a component at k=1. But the frequency content is nothing more than the FFT tells us. There is a cosine at k=14, and another at k=16 (that's at k=-14). The only question would perhaps be related to whether or not we could separate the component at 14 from the one at 16 in a recovery effort with an appropriate low-pass, which would have to come down pretty rapidly just above k=14 and be very far down by k=16. But nothing special otherwise. See later Fig. 12a.

Fig. 8b shows something that is special. The frequency is k=15 and it is a cosine that is sampled. Accordingly we have samples at the extreme values, alternating +1 and -1, and it is a full-size signal. In one sense, this is not allowed of course. The sampling theorem does not allow a frequency to actually reach half the sampling frequency. But we have gone ahead with what actually happens, and the frequency domain corresponds to what is happening in the time domain. Note that the FFT shows that the component coming down at (N-k) has overlapped exactly the one going up (k) and we get the spectral lines added. Nothing wrong with the math – apparently. See later Fig. 12b.





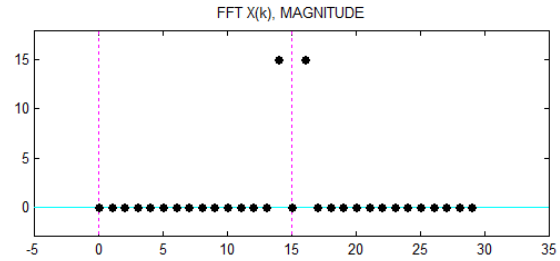
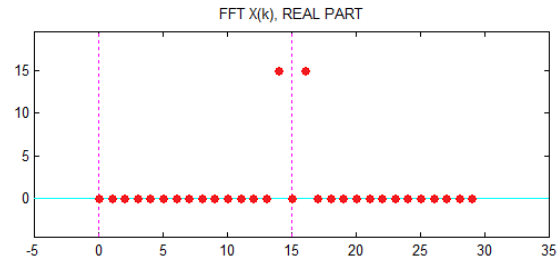
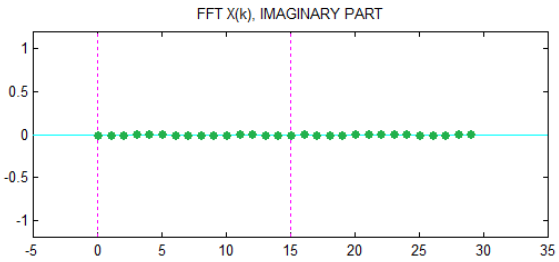
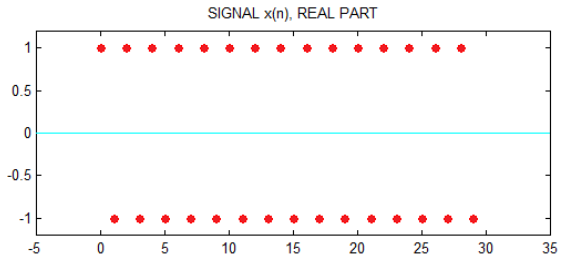


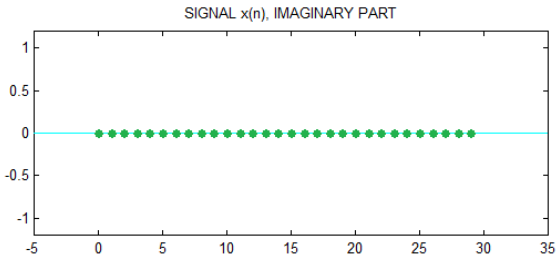
Fig. 8a

Cosine of frequency 14/30 is sampled slightly faster than twice per cycle. Compare upper left here to upper left of Fig. 8b.









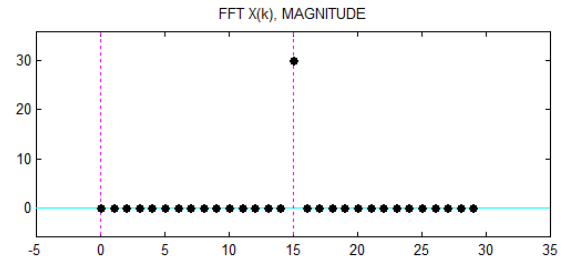
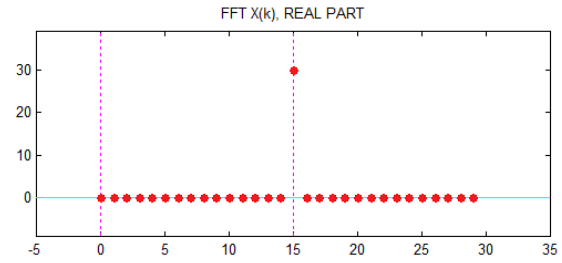
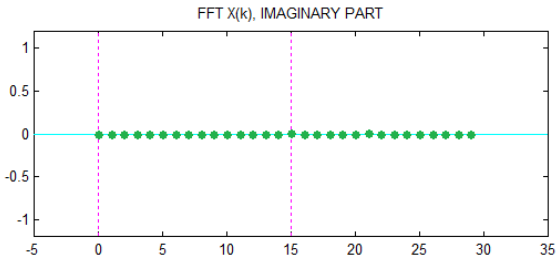
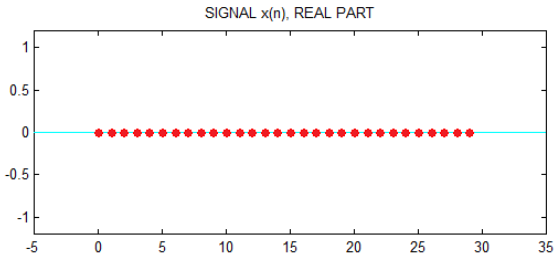


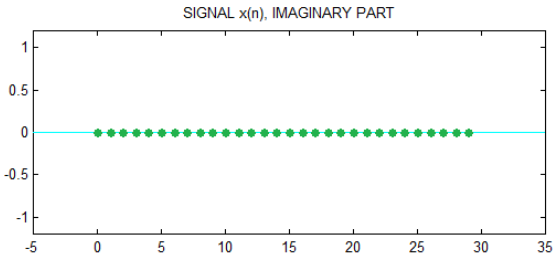
Fig. 8b

Cosine of frequency 15/30 is sampled exactly twice per cycle. Components of the FFT from k=15 and k=30-15 overlap and add.









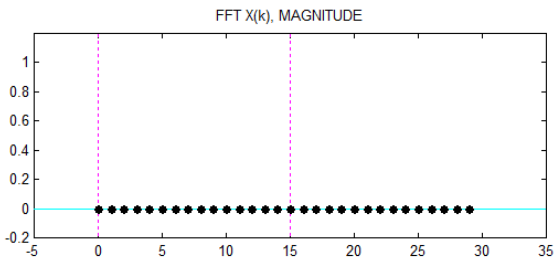
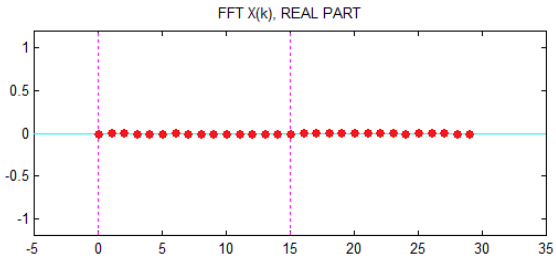
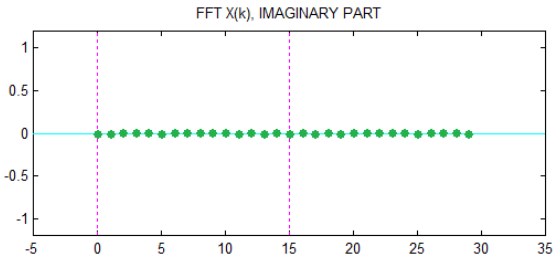


Fig. 8c

Sine at frequency 15/30 is sampled twice each cycle. Everything is zero. Signal is lost.





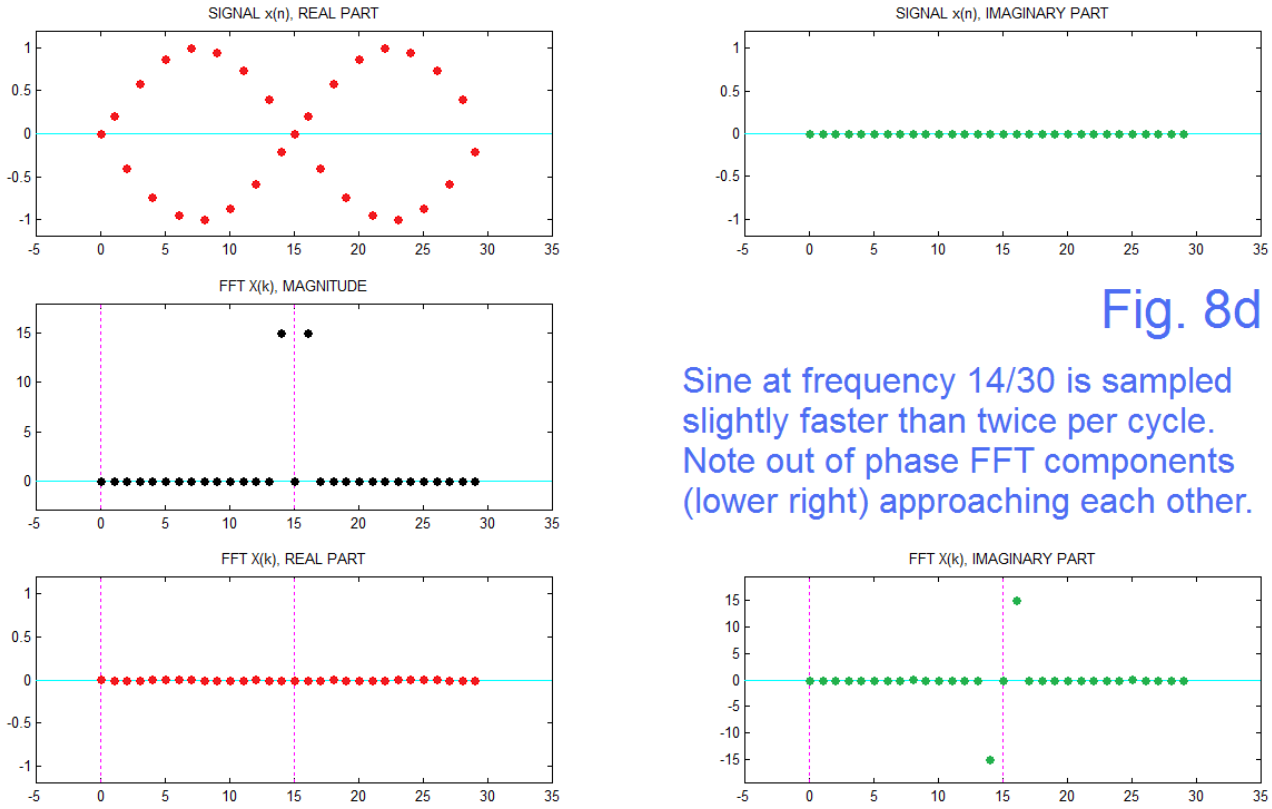


Fig. 8d

Sine at frequency 14/30 is sampled slightly faster than twice per cycle. Note out of phase FFT components (lower right) approaching each other.

Next we make the simple change of a sine instead of a cosine. We perhaps do not really need this graph (Fig. 8c) because we can guess what it looks like. NOTHING! But these graphs in a digital paper don't really cost us any actual paper. Part of the effort in all these graphs is to actually verify that what we supposed had to happen actually did happen. In Fig. 8c, we sampled a sine twice a cycle starting at zero. Every sample was zero, and the FFTs that follow have to be zero. To get a better idea how this happened, consider Fig. 8d where the frequency of the sine was backed off to k=14 instead of k=15. As we might have expected, the input signal in Fig. 8d is a phase-shifted version of the one in Fig. 8a, and we agreed that Fig. 8a was perfectly normal. Likewise Fig. 8d is perfectly normal, so why don't the two lines of the FFT add as they did in Fig. 8b when we go to Fig. 8c?

From Fig. 8d, k=14, we saw that the FFT lines for the sine case were of opposite sign. When k=15 for the sine case, the FFT lines are equal in magnitude, opposite in phase, and both located at k=15. Thus they cancel to zero. Another way to demand this result is of course to say that there was no energy in the time waveform, therefore the frequency description had to be zero as well.

It does seem strange that the phase matters. There is nothing about phase in the sampling theorem. Sine and cosine or any other phase should work the same. Indeed they are in Fig. 3a, Fig. 3b, and Fig. 3c as well as Fig. 8a and Fig. 8d, and in most other case we might try. The FFTs are supposed to tell us the level of a particular frequency in a

signal, and for k=15, the FFTs seem to be failing – telling us in one case (Fig. 8b) there is a full signal and in the other (Fig. 8c) there is none at all. It doesn't know! We can if we wish try various combinations, and it finds the cosine phase, but not the sine phase. Viewing this as sampling twice per cycle, we can understand how the set of samples we get will walk up and down as the chosen phase changes. Well, that's why the sampling theorem says less than half the sampling frequency.

How should we handle this case of energy at half the sampling frequency? Possibly in two ways depending on the application. If we are using the FFT as a spectrum analyzer being applied to some signal that represents samples of some continuous-time signal, then probably we should just discard the middle bin as "taken in error". This is only an issue with even length, and likely in most cases, the energy there will be minor. On the other hand in the case of interpolation, perhaps we should keep the center bin. As we have argued, this bin is to be divided into two pieces. Below we will see that the signal at half the sampling rate emerges from the interpolation process in this case.

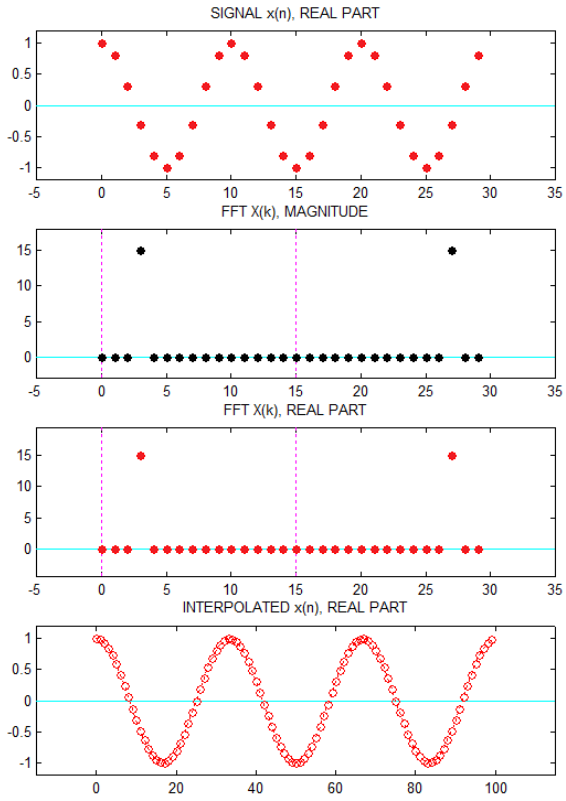
Interpolation in Time

We have thought of this note as a review of interpolation in time with an extension to interpolation in frequency. Eventually we notice that all the interpolation with the FFT is really about is truncating an ordinary Fourier Series (FS) using the DFT harmonics. Well – not quite an ordinary FS. The difference here is that the cosine and sine components have frequencies that are multiples of f_s/N. So we have no prior fitting of this FS fundamental to a known fundamental of a periodic function we propose to represent by the FS. With proper setup. we can use the DFT to estimate FS coefficients [8], but in general the FFT just gives us a FS based on its own terms. Once we accept this FFT-based FS, we have found the entire underlying continuous time function that is the correct bandlimited solution corresponding to the samples given, in the sense of the DFT.

With the suggestion that, in order to interpolate the time-domain data, we are to insert zeros in the "middle" of the FFT and then take the inverse FFT of the longer sequence, we need to keep in mind that middle is the high-frequency region. The portion below the "split" that does not move, is the usual low-frequency portion, and the portion above the padded zeros is the "negative side" of this same low-frequency stuff, once we consider the periodicity. The usual Euler equations and symmetry properties can be used to combine these terms into cosines and sines, forming the FS, as in equation (1c) in exponential form.

Fig. 9a is the first of our examples showing interpolation in time. The program here is the same as the one for the previous graphs, except it adds two panels at the bottom. All that we have done for interpolation is to take the FFT and insert 69 zeros in the middle, splitting X(15) between the new X(15) and X(85). This is multiplied by (100/30) to make up

the lost gain, and the inverse FFT is then taken for 100 samples. To keep the plots of the interpolated results relatively uncluttered, open circles were plotted. In addition, we also plotted a dotted version of a continuous line. This will prove to be instructive shortly.



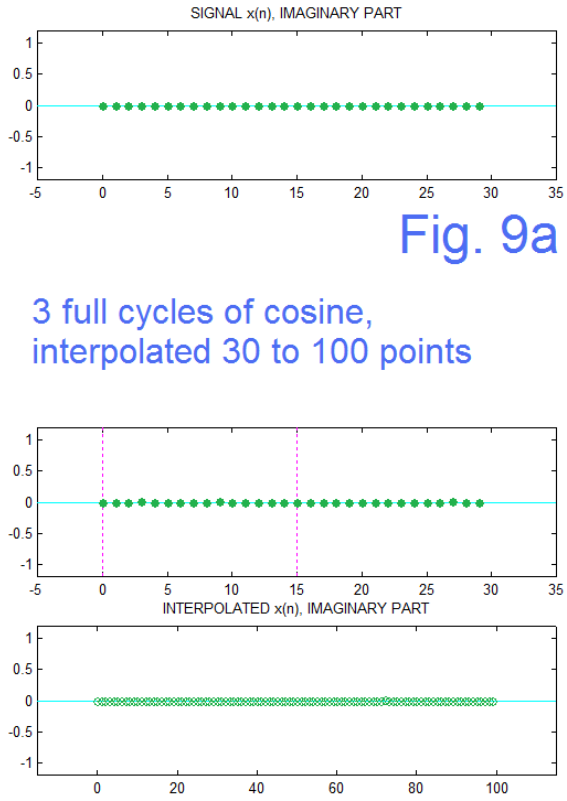
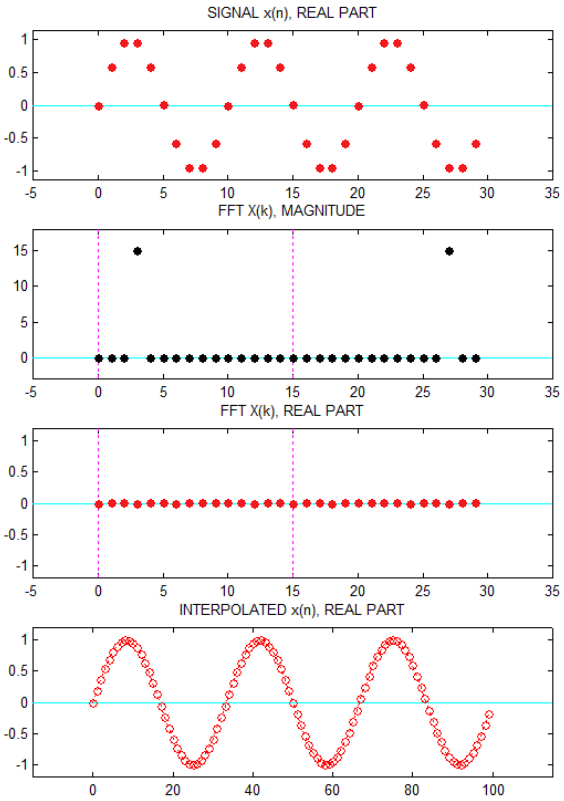


Fig. 9a

3 full cycles of cosine,
interpolated 30 to 100 points



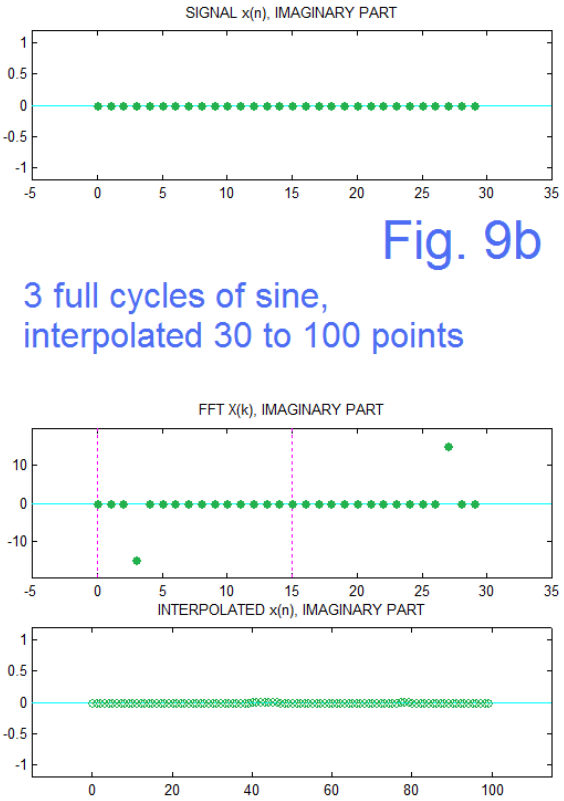


Fig. 9b

3 full cycles of sine,
interpolated 30 to 100 points

Fig. 9b is the sine case corresponding to the cosine of Fig. 9a and is included here mainly because it has the amusing zig-zag as the real signal (upper left of Fig. 9b) has a purely imaginary FFT (third row right of Fig. 9b) and then jumps back to the real interpolation (lower left of Fig. 9b)

Another instructive case will be that when we have a non-integer number of cycles for input samples, and this is shown in Fig. 10. This example actually has 2.4414 cycles of the cosine. We see that the FFT here has non-zero magnitude for all k=0,1,2,…29, so all of the possible DFT harmonics are involved. That is, we have a truncated Fourier Series for k=0 to 15. Here interpolation can offer us two things. It of course gives us the interpolated equally spaced samples, but also in giving us more samples it provides an idea of what continuous periodic waveform the FFT "had in mind". Note here that there was energy at the very middle, and it was necessary to split the middle term (k=15) of the FFT.

If we concentrate on the <u>center portion</u> of the reconstruction (lower left of Fig. 10) we see a not unreasonable interpolation of a cosine with frequency 2.4414 (as far as we could tell). The ends have that discontinuity, and are not very useful for interpolation. However, they do provide an answer to the "had in mind" question. Bear in mind that any signal represented by a DFT pairing is by definition always periodic. The actual periodic waveform corresponding to the Fig. 10 upper left is not a cosine of frequency 2.4414 but just the repetition of the 30 samples over and over. The interpolation (lower left) shows the end distortions (relative to an expected cosine). Because of the 0.4414 cycle (almost half a cycle) there is a <u>severe discontinuity</u> and corresponding jog at the ends.

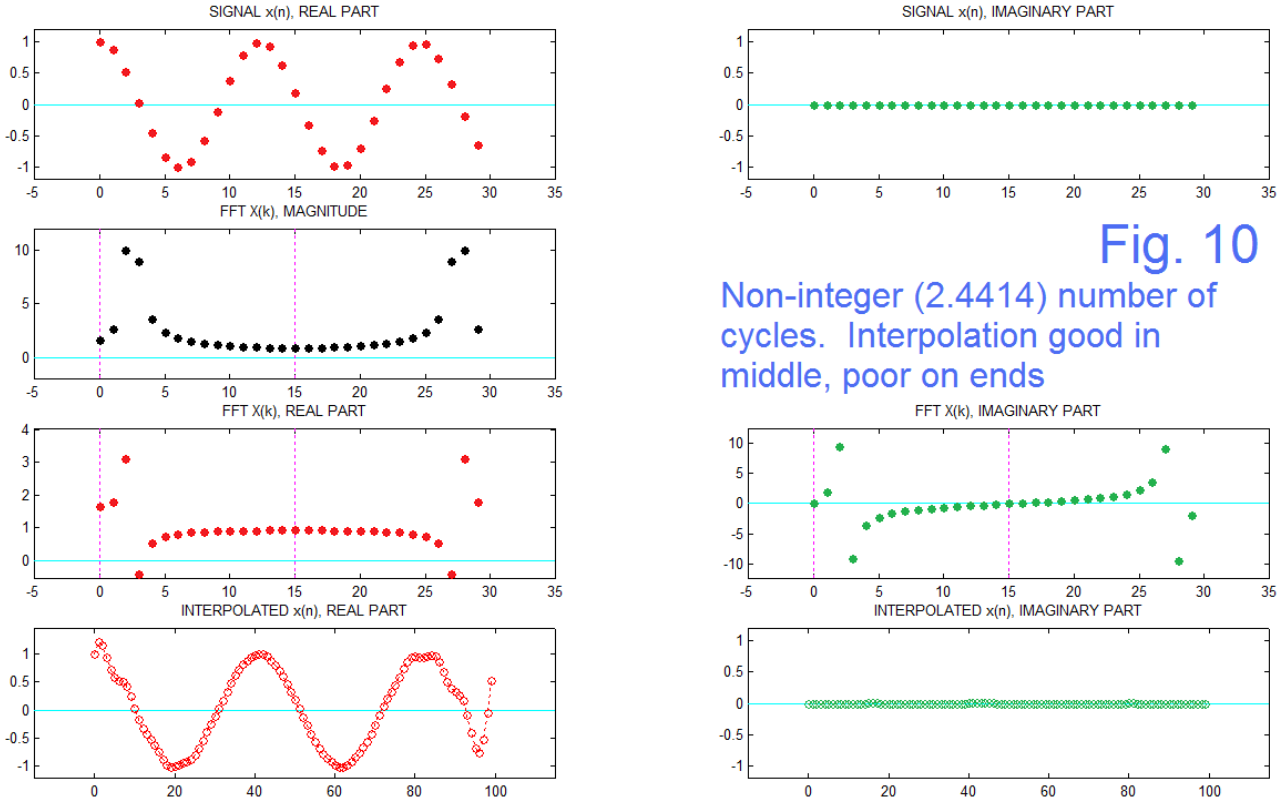


Fig. 10

Non-integer (2.4414) number of cycles. Interpolation good in middle, poor on ends

Fig. 11a shows another instructive case – that of a signal that is fully complex having both a non-zero real component and a non-zero imaginary component. Such we conveniently and convincingly obtain by choosing a length-30 random sequence for the real part and a second length-30 random sequence for the imaginary part (as in Fig. 6). It seems fair to say that the general perception of apparent randomness continues through the FFT representations. (Nothing apparent is going on that does not look random.) The interpolations on the other hand (bottom panels of Fig. 11a) give a different impression. In these we see what is apparently a bandlimited reconstruction. The interpolated sequences look to be somewhat smoothed, rather than random. What we have is in effect the interpolation of the two components (by linearity). Both the real and imaginary parts of the input (which add together) are thus combined to a single FFT which is then used for interpolating, giving the interpolation of both. Note that if we had only the real sequence, it still would have had a complex FFT, which would have had the usual symmetries for a real time signal. What we have in Fig. 11a shows FFT parts which have lost symmetry because there are actually 60 numbers now.

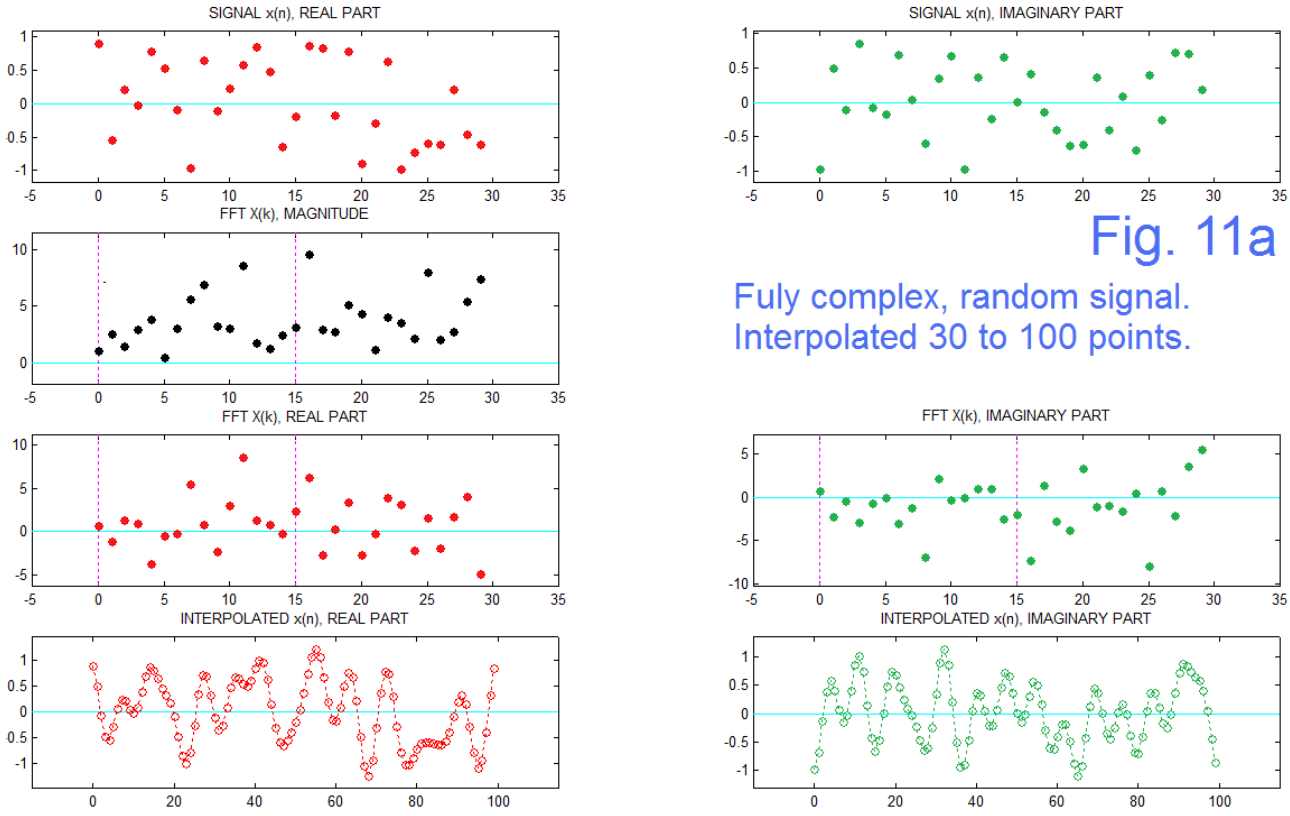


Fig. 11a

Fuly complex, random signal.
Interpolated 30 to 100 points.

So, we like to suppose that the length-100 interpolation, say the bottom left of Fig. 11a, is pretty much just a filled-in version of the original length-30 top left. Careful study perhaps indicates some potential doubts. For one thing, the end of the interpolation (last three samples) shoots upward, unsupported by the previous samples that were low. This is actually just another case (like Fig. 10 lower left) where the FFT feels obliged to correspond to periodic sequences – it is going back up to the beginning end which happened to be high. So, that in itself is not suspicious or unexplained.

The other problem is that we have interpolated samples that are larger than the surrounding samples of the original. In Fig. 11a, lower left, we see this at samples 55 (exceeds +1) and at sample 68 (less than -1). This too is perfectly explainable. The input samples are from a supposed underlying continuous waveform, and there is no reason (in general) to suppose that samples will be taken at or close to the peaks. So we are interested in just where the interpolated samples fall relative to the originals. This will be easier to do if we run the same sequence with a different interpolation factor.

Since we derive the interpolation of 100 points from an input of 30 points, the interpolation factor is 100/30 = 3⅓. This is not an integer ratio. If we had interpolated to 120 points the interpolation ratio would have been 120/30=4, an integer. Original samples would appear with three interpolated samples in-between. This we show in Fig. 11b. This looks much the same as Fig. 11a. Here, in Fig. 11b, lower left, we have marked every 4th sample, the originals, in solid blue. Thus we can better see how the peaks are supported by a combination of large surrounding samples.

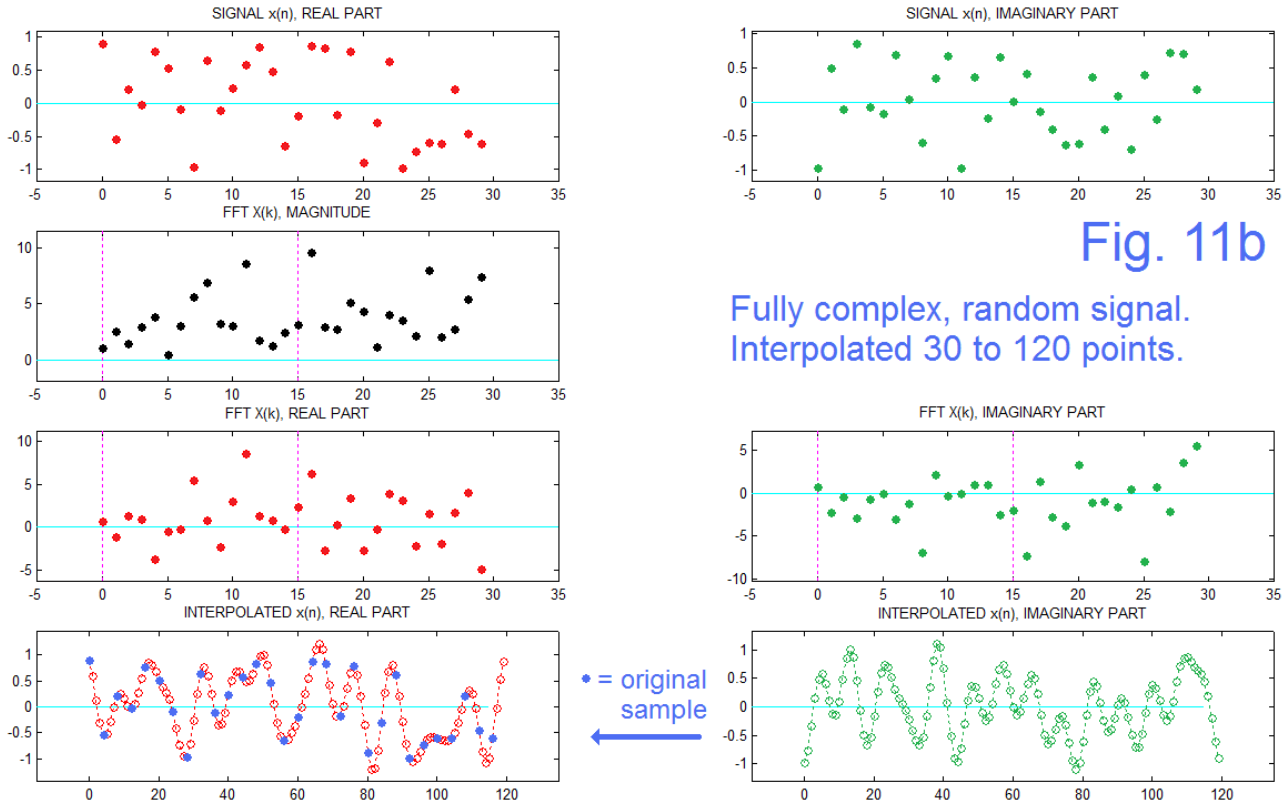


Fig. 11b

Fully complex, random signal.
Interpolated 30 to 120 points.

Why not do this with the interpolation to length 100? Well, we can – we just don't learn as much because in that case, only every 10th sample of the reconstruction is one of the originals. That is, 3⅓ x 3 = 10 to get to an integer index of the originals. This curious situation is shown in Fig. 11c, where the solid blue dots are those that are common to both the original set and to the interpolation.

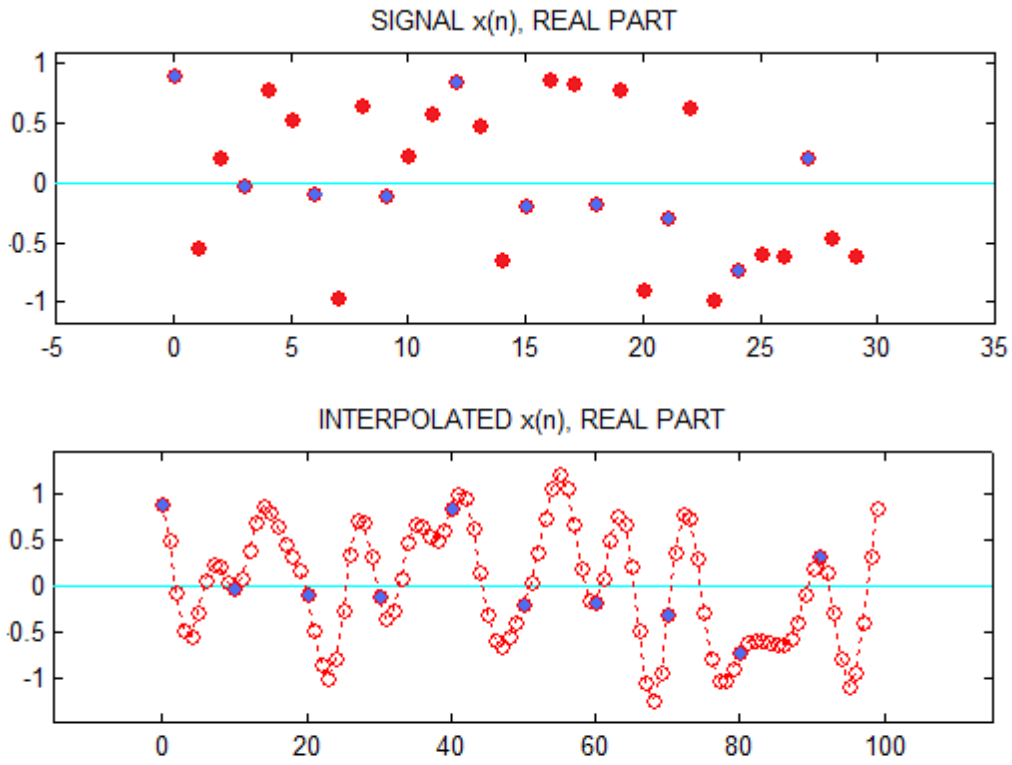
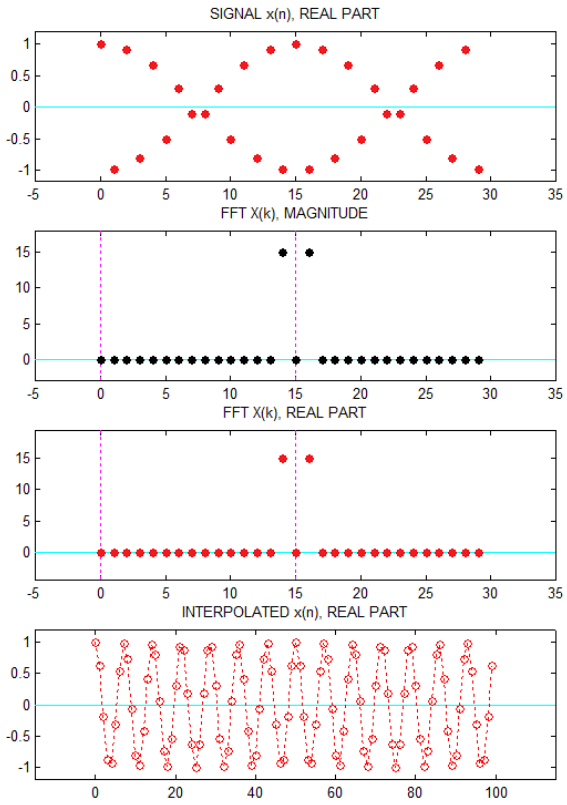


Fig. 11c Interpolate 100/30

Every 10th Sample of Output is an Original

Finally, we look at two examples (Fig. 12a and Fig. 12b), expanding on Figures 8a, 8b, 8c, and 8d, where we again interpolated cosines (as in Fig. 9a) except here the frequencies are much higher, at the upper limits. Both are real and have even symmetry, so their FFTs are real. One has a frequency of 14/30 and the other has a frequency of 15/30. We note first that the interpolation output, the lower left of these figures, look nearly the same. These results would be very confusing were it not for the dashed red lines that connect the samples, so we can see that Fig. 12a seems to represent 14 cosine cycles while Fig. 12b seems to represent 15 cosine cycles. This we likely expected. Because these are so similar, it is perhaps surprising how different the upper left panels of the same figures are. Indeed the upper left of Fig. 12b is just an alternating sequence of +1 and -1, 15 such alterations. This is exactly what we had to have from 30 samples of 15 cycles of a cosine.

The upper left of Fig. 11a is quite different. It kind of looks like single cycles out of phase. Of course what is happening is that the cosine of frequency 14/30 is being sampled just a bit earlier each cycle (unlike Fig. 12b) and thus the samples are not at the peak all the time, but migrate along the waveform, and alternate in sign. This is nothing more than sampling a frequency near the maximum frequency (1/2 the sampling rate) allowed by the sampling theorem. The FFT in this case is very tight. Recovering the signal from the samples would require a low-pass filter that passes the frequency 14/30 but



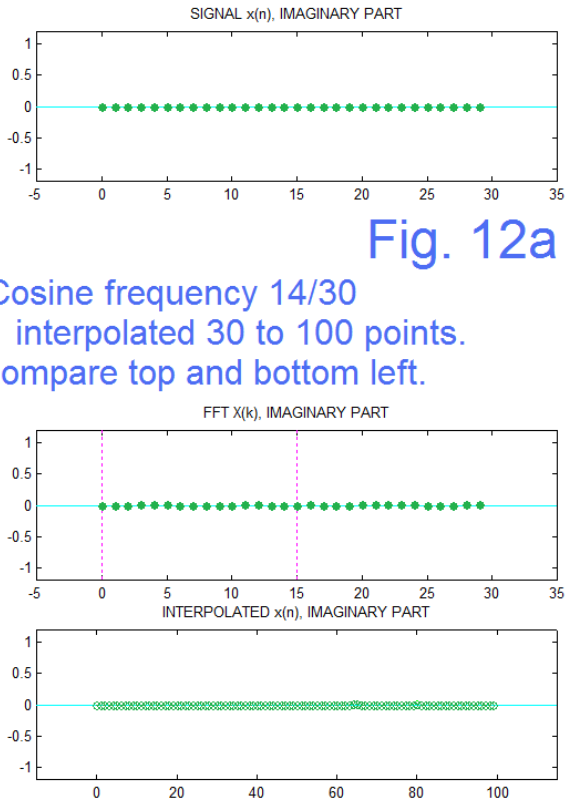
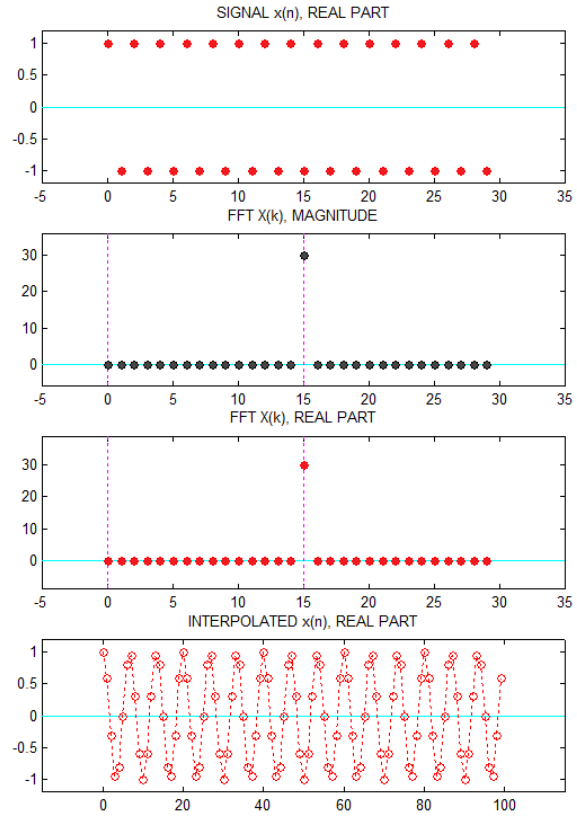


Fig. 12a

Cosine frequency 14/30
interpolated 30 to 100 points.
Compare top and bottom left.



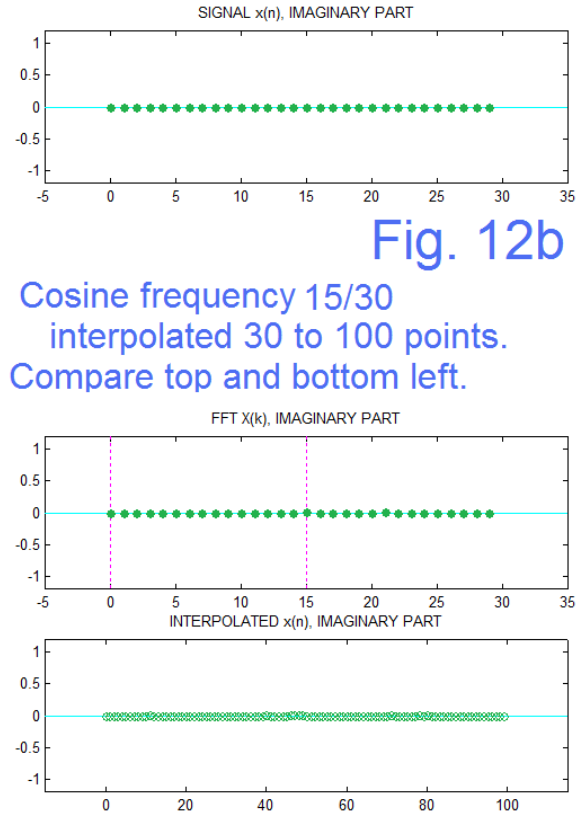


Fig. 12b

Cosine frequency 15/30
interpolated 30 to 100 points.
Compare top and bottom left.

rejects the frequency 16/15. That's quite an order. It may first of all be a surprise that a sequence of samples such as that in the upper left of Fig. 11a, with its huge amplitude variations, can result in a much more uniform amplitude [6,7]. Here we get a huge boost in the filtering problem by the upsampling (analogous to "oversampling" in audio [9]) and the FFT inverse performs the low-pass filtering. Thus Fig. 12a should be a limiting case.

Accordingly Fig. 12b is technically "the limit" or better, beyond the limit. As we mentioned above, when we sample a signal that has frequency half the sampling rate, the amplitude is arbitrary. Had we sampled a sine, all the samples would have been zero. Thus we get the alternating sequence at the correct amplitude only by the artificial choice of exactly cosine phase. A nice curiosity to keep in mind.

Interpolation in Frequency

For many years, when people wanted to have a better view of the spectrum than that obtained directly from the FFT, there was the concept of "zero-padding" of the time sequence and taking a larger sized FFT. For example, if we had 16 time points, we can take a length 16 FFT, of which the frequencies $kf_s/16$ are used for k=0,1,2,3,4,5,6, and 7 (here assuming a real sequence as discussed above). Nothing prevents us from trying a longer time sequence, say by adding 240 zeros to the 16 given points, making it length 256 total. Now the length 256 FFT has frequencies $kf_s/256$ for k=0,1,2,…127. This is not more information. We have no additional resolution, although things may be more agreeable to glance at.

Now, if we borrow what we learned from interpolation in time, we discover many curious things. First, traditional zero-padding puts the padded zeros at the end of the given sequence of samples. This had the virtue of leaving the original sequence in place and just adding a tail of zeros which obviously added no "energy" or information to the signal. But in time domain interpolation we split the spectrum (the FFT) in the middle. Different! One often hears that you must put the padded zeros on the end or else you get a very different spectrum due to the middle gap. Understandably, we become uneasy with the notion of special points in time, a matter which we will take up soon below.

But for now, let's argue that putting the zeros in the "middle" is logically as good, or better, than putting them on the end. Once we decide that time samples are inside the province of the DFT (we say that we are going to analyze it with an FFT), we have declared it to be periodic. The padded zeros could be inserted anywhere in that periodic sequence. If we put them in the middle, then the samples below the middle remain in place while the samples above the middle move up (usually way up) to the far end, and this looks strange. But periodically (because of the wrap-around), the time sequence is the same. If you wish, these high end samples are to be understood as those on the negative side. In the language of filters, putting the zeros on the end is a causal, linear phase case while putting

them in the middle is a non-causal zero phase case (often very convenient). The magnitude FFT does not care as long as the original sequence remains in one segment periodically.

[Here we discuss the notion of a zero of time versus a zero of frequency. We are advocates of the duality or symmetry of Fourier transform notions. But this is mathematics. Physical reality gives a different perspective. We know the zero of frequency well: f=0 is just DC. It's phase holding still for all time. Where is the corresponding zero of time? We feel that must be arbitrary because so many math proofs begin with "with no loss of generality we set the step transition to t=0", of something like that. So, is there a non-arbitrary time for which phase stops for all frequencies? If you know the answer, let me know. It must be how we perceive things. Perhaps it is the zero-phase positioning being discussed here. I have worried about this for years to no resolution.]

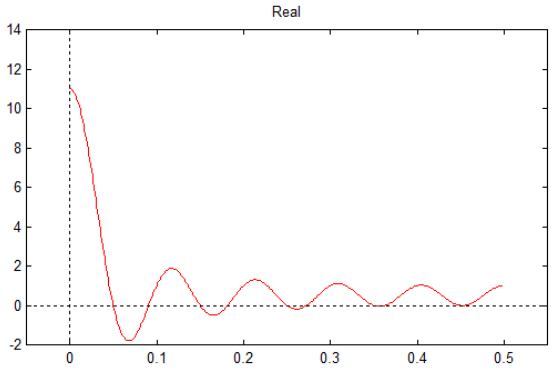
In the time domain, the FFT interpolation procedure could be viewed as a discovery of the finite bandwidth (truncated) Fourier Series (a continuous periodic function), from which we could obtain the original samples or a larger or smaller set of samples. Interpolation as it occurs in practice is generally to a substantially larger set of samples, all of the same underlying continuous-time periodic waveform. That's all it is.

So you may be supposing you have never interpolated in frequency, but very likely you have. If you have studies and/or used a digital filter, you know that its frequency response is a periodic function of continuous frequency. Further, we totally expect the filter to operate over a continuous range of frequencies. We put it on the bench, connect up an analog frequency generator, and "sweep" the response. All frequencies between DC and half the sampling frequency are supposed to be available. When we calculate the frequency response, it is a continuous function – determined by the DTFT (equation 1d) where the continuous frequency is usually written as ω. The DTFT is just a Fourier Series with the usual roles of time and frequency reversed. In fact, the DFT is just a sampled version of the DTFT. So if you have calculated a frequency response, you have interpolated in frequency. Note that the equivalent of bandlimiting in frequency is time-limiting in time. Thus, for example, an FIR (Finite Impulse Response) filter is the perfect excuse for truncating the FS (the DTFT).

While we could more-or-less redo all the steps above reversing the time and frequency domains, it seems easier and more productive to proceed on the very familiar territory of frequency response calculation for simple FIR filters. Suppose we use a length-11 rectangular impulse response consisting of 11 ones in a row. We know this well enough [10]. That's 11 ones in the time domain, all in a sequence. For one thing, we could take the FFT of this time signal (and will in a bit – Fig. 14) but it would only be 11 points. We kind of had in mind that we could have some notion of the frequency response for a continuous range of frequencies. We need more points – we need to interpolate!

Users of Matlab will almost immediately ask to use the function ***freqz***. Almost certainly if given an impulse response such as h = [1 1 1 1 1 1 1 1 1 1 1] we recognize this as the

numerator of a transfer function (the denominator being just a single1), and run ***freqz(h,1,500)*** to get 500 points (thus looking more-or-less continuous, not just 11 from the FFT) and thereafter we know what the correct answer looks like. This we see in Fig. 13a.



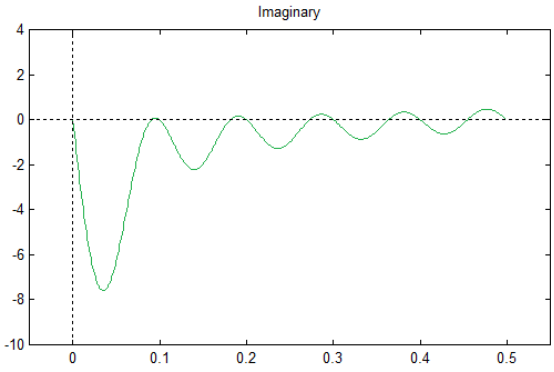
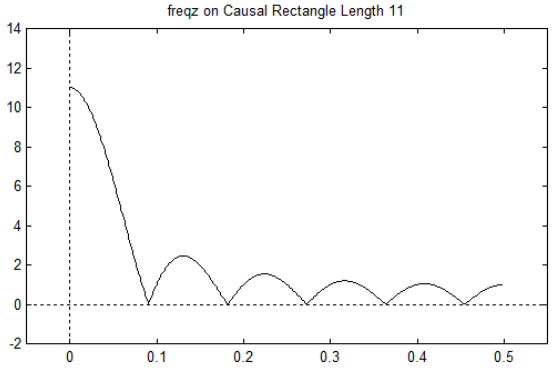


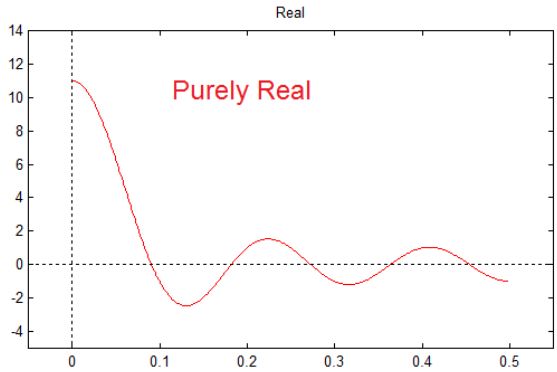
Fig. 13a



h = [1 1 1 1 1 1 1 1 1 1 1]

H = freqz(h,1,500)

Perfectly ordinary frequency-response (using freqz)



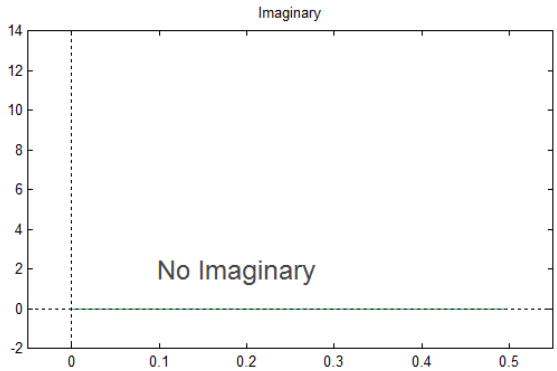
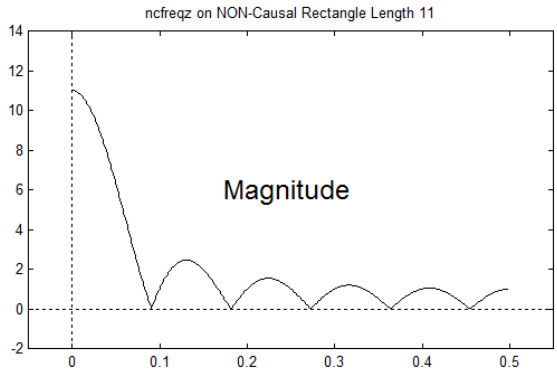


Fig. 13b



h = [1 1 1 1 1 1 1 1 1 1 1]

H = ncfreqz(h,1,500)

Making the frequency-response purely real by making the impulse response non-causal.

Okay. Nothing much is as familiar as that "periodic sync" seen in the magnitude response (Fig. 13a, lower left). What may be curious here is that when we look at the real and imaginary parts (which we don't often do) we see that they are unusual looking. This is all because of the linear phase causality assumed by the ***freqz*** function. We do not have to assume this. In fact, I found this so bothersome that years ago (20 years ago!) I wrote a few lines called ***ncfreqz*** (non-causal freqz) which simply undid the linear phase in favor of zero phase, and this was very useful for symmetric impulse responses as they could be made purely real. (It is an alternative, not a replacement.) Perhaps I have published this simple modification before, or perhaps not. Here it is:

```
function H=ncfreqz(b,a,N)
%   H=ncfreqz(b,a,N)
%    This function does a non-causal version of freqz.
%    It should be used principally for FIR filters.
%    In particular, even symmetric sequences will have
%       a purely real result while odd symmetric
%       sequences will have a purely imaginary result.
%   B. Hutchins          EE425                Fall 1994
L=length(b);
omega=0:pi/N:(N-1)*pi/N;
H=freqz(b,a,N);
H=H.*exp(j*((L-1)/2)*omega).' ;
```

Using ***ncfreqz*** instead of ***freqz*** leads to Fig. 13b. Note immediately that the magnitude (lower left) is the same in both Fig. 13a and Fig. 13b. Fig. 13b is nicer, at least in this case (symmetric impulse response), and there is no imaginary part. It is non-causal, zero-phase, but differs only by a shift. We are accustomed to think of the shift as to the right, making non-causal causal. (In fact, we often simplify a design by doing the non-causal calculations and then just shifting right.) Here a left shift looks like the reverse process, to negative times. But it will be seen here that when we don't have negative time indices (as with an FFT), because of the periodicity the left shift is exactly equivalent to inserting zeros in the MIDDLE of the time sequence.

In this and in other simple cases, we don't need ***freqz*** so much. We can do the calculation in closed form using equation (1d). Consider the impulse response h(n) = 1 for n = -5 to +5, zero otherwise. Thus using equation (1d) we have (setting T=1) [10] :

$$H(\omega) = e^{j5\omega} + e^{j4\omega} + e^{j3\omega} + e^{j2\omega} + e^{j\omega} + 1 + e^{-j\omega} + e^{-j2\omega} + e^{-j3\omega} + e^{-j4\omega} + e^{-j5\omega}$$

$$= 1 + 2\cos(\omega) + 2\cos(2\omega) + 2\cos(3\omega) + 2\cos(4\omega) + 2\cos(5\omega) \qquad (5)$$

which is just a truncated Fourier Series in frequency. In fact we can plot this from ω=0 to ω=π, and the result is shown in Fig. 13c, identical to the upper left of Fig. 13b).

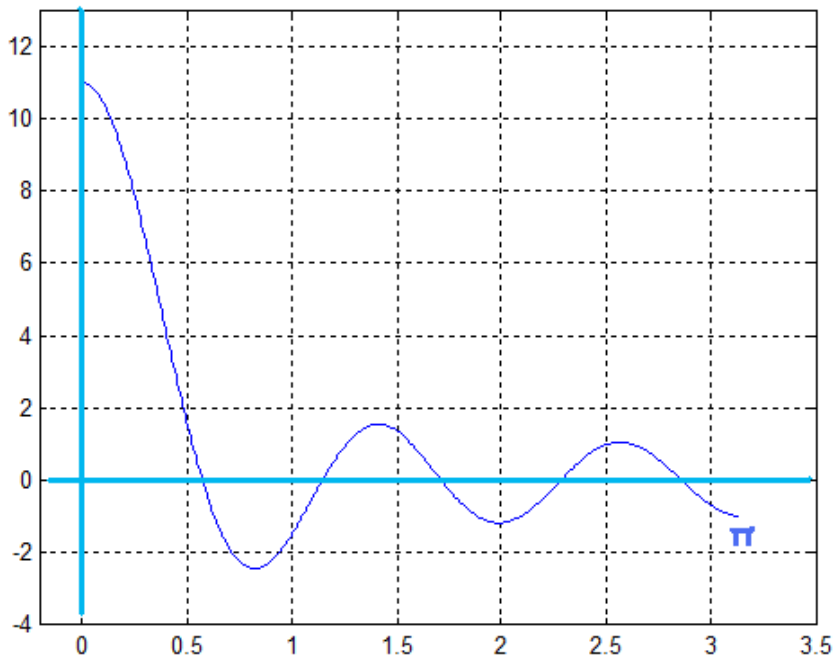
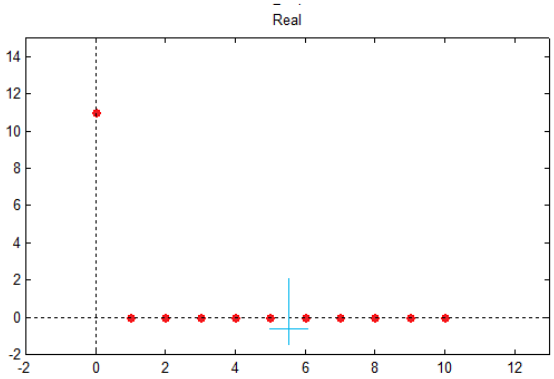
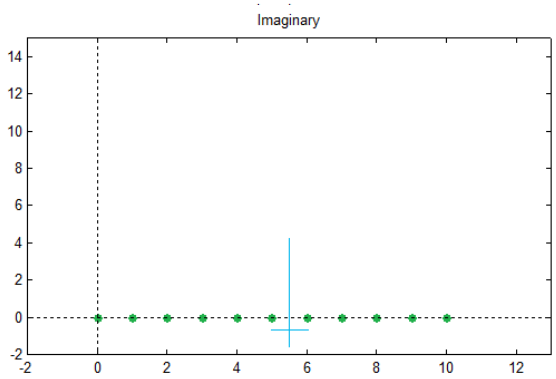


Fig. 13c

Analytic Calculation:

DTFT or Truncated Fourier Series in Frequency





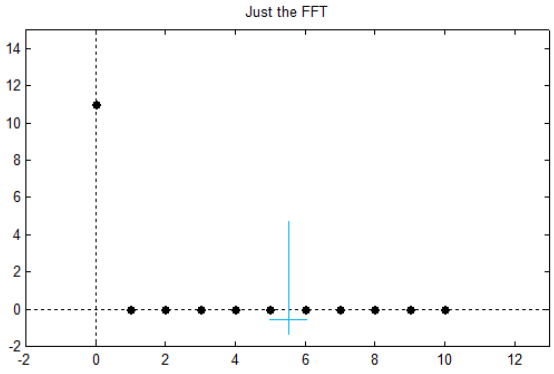


Fig. 14

Length-11 FFT of a length-11 time sequence (not much to see here).

```
h = [ 1 1 1 1 1  1  1 1 1 1 1 ]
H=fft(h)
```

Our final step here is to relate this ***freqz*** stuff, and the hand calculation, equation (5), to the FFT. Fig. 14 shows the FFT of the length-11 sequence of ones. Not much there – it's just a value of 11 at k=0 (DC) for the real part and for the magnitude. Indeed, if your time signal is length-11 and all 11 values are 1, and it is periodic because we are using the FFT, all we have is DC. So we can believe that FFT passes DC, but we of course do not suppose that it blocks all other frequencies. Note as well the light blue lines demarking the lower and upper (redundant) portions of the FFT.

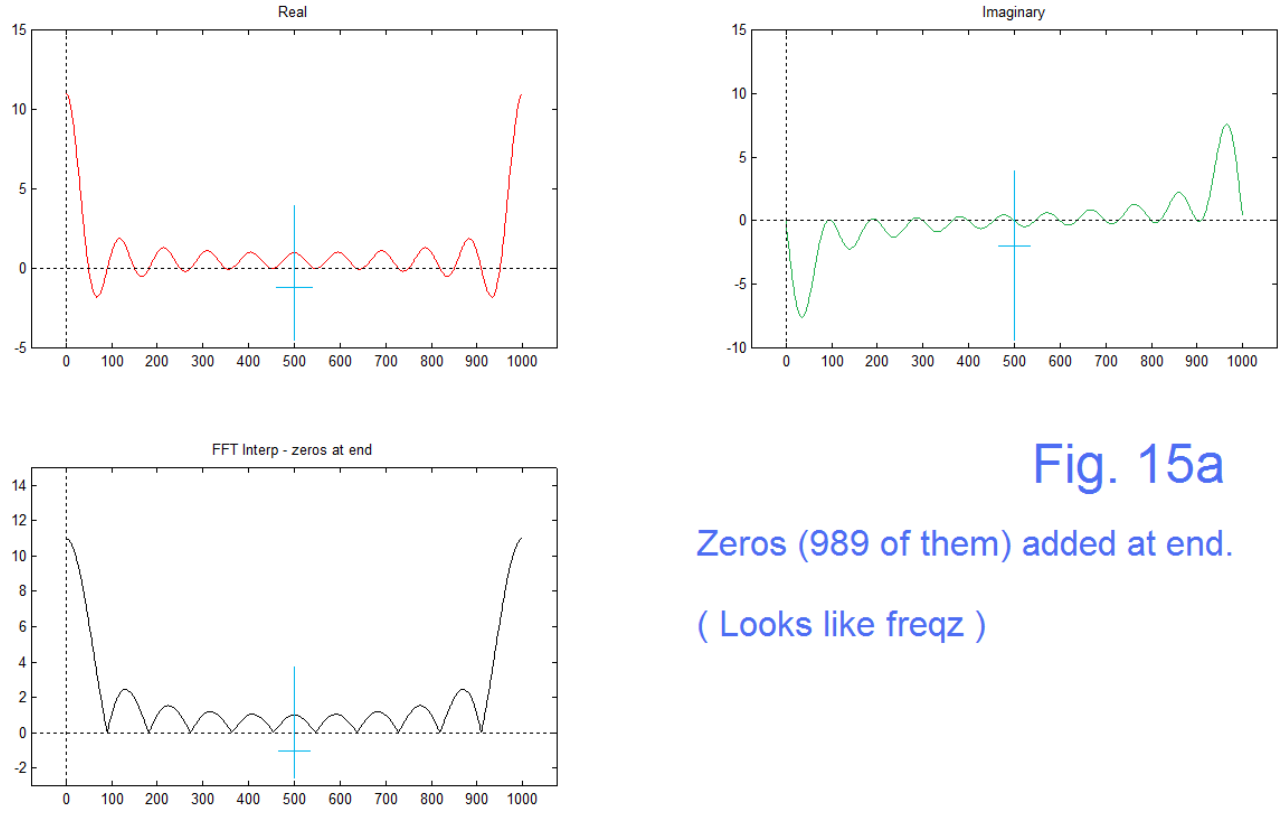


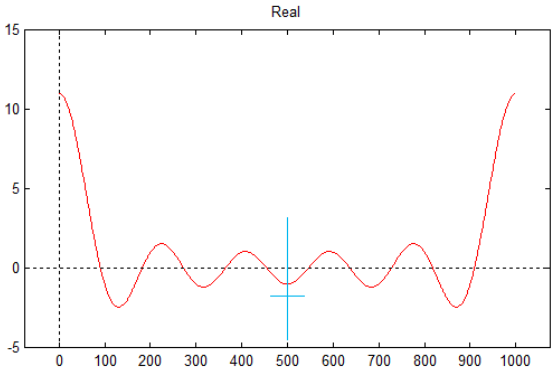
Fig. 15a

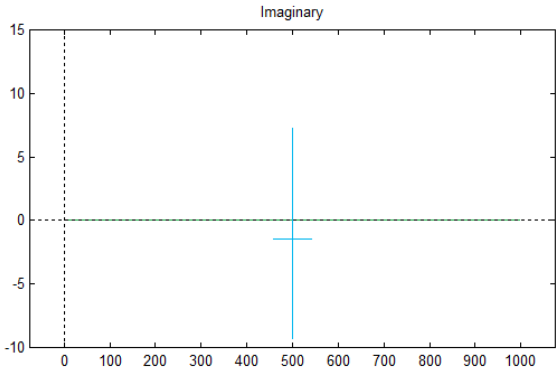
Zeros (989 of them) added at end.

(Looks like freqz)

So instead of sticking with the length 11 FFT, here we will zero-pad the time sequence in three different ways: at the end (Fig. 15a), in the middle (Fig. 15b), and at the beginning (Fig. 15c). We will add 989 zeros to the length 11 signal and take the resulting length 1000 FFT. Here the 1000 points are close enough that we plot the FFTs as continuous curves. Indeed, we see that there were lobes between the zeros of Fig. 14. Again, the light blue vertical line marks the middle of the FFT. Here are the important things to note:

The lower left panels of Fig. 15a, Fig. 15b, and Fig. 15c are all identical and give the same result as the use of ***freqz*** above. That is, the magnitude response is in total agreement regardless of the particular methods here. We note in addition that the results in Fig. 15a (padded zeros at end), looking carefully at the real and imaginary parts left of the light blue divider, are the same as for the original use of ***freqz***, as seen in Fig. 13a. The results in Fig. 15b (padded zeros in the middle), are correspondingly the same as ***ncfreqz*** (Fig. 13b) and the hand calculation (Fig. 13c). This leaves us with Fig. 1c which is the same as Fig. 13a except the imaginary part is reflected left/right.





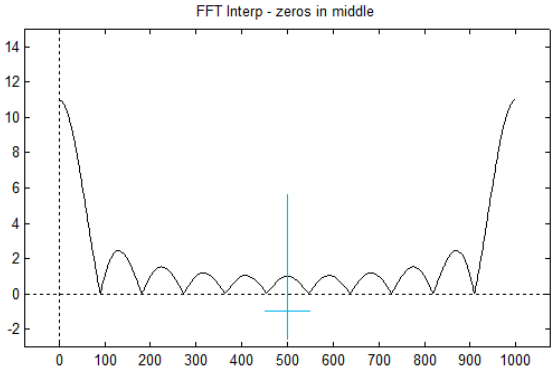
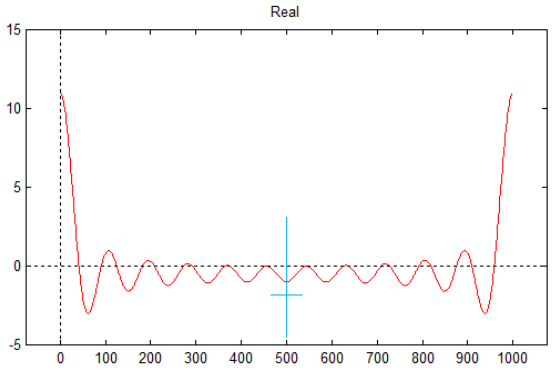
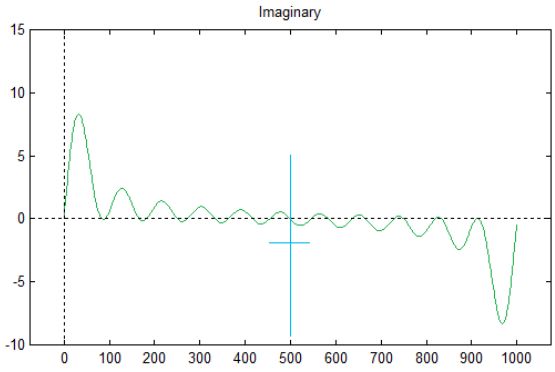


Fig. 15b

Zeros (989 of them) added to middle.

(Looks like ncfreqz)





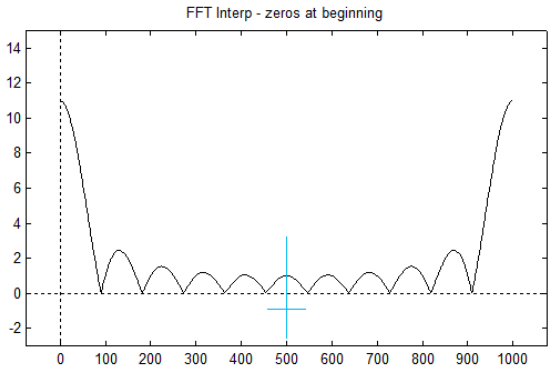


Fig. 15c

Zeros (989 of them) added to beginning.

(inverts imaginary part)

Summary:

Here we have put together a good deal of information and provided many examples. Further, we suggest that people use the Matlab code in the appendices (or similar) to make more examples. It is always useful to set up some input test, and before pressing enter, see if we can anticipate the output.

We have found here principally two things. First, we have a better understanding of why we sometimes need to "split the middle" of an FFT before interpolation. More fundamentally, we see why we are putting our padded zeros in the middle. Perhaps the newer point is that when we do this same thing in the time domain (interpolating the frequency domain), we are not putting a big gap in the time signal, but rather rotating the time signal. This is likely obvious when we actually see what we are accomplishing.

The second major finding is that while interpolation in the frequency domain has not been addressed here, in itself it is not new, as we have been using it all along to calculate frequency responses of digital filters.

Finally we find the interpolation studies useful in illustrating that we are NOT creating new information (resolution in time or frequency) but just making things more comfortable to look at.

REFERENCES: (all references by B. Hutchins from the Electronotes collection)

[1] "Interpolating Samples with the DFT/FFT," *Electronotes*, Vol. 16, No. 172, July 1988.

[2] "A Short Presentation of FFT (DFT) Interpolation," Electronotes Application Note 398 July 20, 2013
http://electronotes.netfirms.com/AN398.pdf

[3] "Fourier Map", Electronotes Application Note No. 410, May 6, 2014
http://electronotes.netfirms.com/AN410.pdf

[4] "An Intuitive Approach to FFT Algorithms", Electronotes Application Note No. 312, Dec. 1990
http://electronotes.netfirms.com/AN312.pdf

[5] Interpretation of DFT as One Frequency Sinusoidal Waveforms", Electronotes Application Note No. 373, February 2009
http://electronotes.netfirms.com/AN373.pdf

[6] "Guide of Digital Filter Network Design: Chapter 2," (Section 2-5 Signal Sampling and It's Consequences), *Electronotes* #A, Vol. 14, #144-146, Dec. 1982-Feb. 1983) pp 24-28

[7] "Revisiting: Beating," *Electronotes*, Volume 22, Number 213 January 2013, pp 30-53
http://electronotes.netfirms.com/EN213.pdf

[8] "Calculating the DFT and the Fourier Series: Each with the Other", *Electronotes*, Vol. 19, No. 188, February 1997

[9] "Basic Elements of Digital Signal Processing: Finite Wordlength Effects, Part 1," (Section 3e, Oversampling and Noise Shaping), *Electronotes*, Vol. 21, No. 204, August 2004, pp 20-33
http://electronotes.netfirms.com/EN204.pdf

[10] "Basic Elements of Digital Signal Processing: Filter Elements – Part 1" *Electronotes*, Vol. 20, No. 197, pp 3-6;
http://electronotes.netfirms.com/EN197.pdf
"Basic Elements of Digital Signal Processing: Filter Elements – Part 2" *Electronotes*, Vol. 20, No. 198 , pp 12-13
http://electronotes.netfirms.com/EN198.pdf

APPENDIX - PROGRAMS

As is our general practice, computer code in Matlab is provided so that readers can verify what was done and run additional examples. We always stress as well that computer code (unlike English and math notation) is unambiguous as computer have no tolerance for ambiguity. This is not to suggest that the code is particularly efficient or error-free.

Program 1 here is called EN222a.m and was used to produce the basic figures of the middle of this report. At the very top is a line that inputs the test signal. Additional test signals are also listed but commented out. One only needs to leave one of the test signals uncommented. Following the input generation lines are additional comments telling which figures of this report were produced. In some cases, a few additional tweaks are necessary to run the examples. All the figures were hand edited following the Matlab graphs. Note because Matlab does not allow an index of 0 (or negative), DFT indices of 0 to N-1 are Matlab indices of 1 to N.

Program 1

```
% EN222a
N=30
n=0:N-1;

  x= cos(3*2*pi*n/N);    % 3 cos cycles  (Fig. 3a, 9a)
% x= sin(3*2*pi*n/N);    % 3 sin cycles  (Fig. 3b, 9b)
% x= cos(30*2*pi/360)*cos(3*2*pi*n/N)+ sin(30*2*pi/360)*sin(3*2*pi*n/N); % 60 deg shift (Fig. 3c)
% x= 2*cos(2*2*pi*n/N)+  1*sin(3*2*pi*n/N);    % mixed 2 comp.(Fig. 4a)
% x= 2*j*cos(2*2*pi*n/N)+  1*j*sin(3*2*pi*n/N);  % mixed 2 comp. imag (Fig. 4b)
% x=cos(2.4414*2*pi*n/N); %  Real, Symmetric, 2.4144 cycles not BL (Fig. 5, 10)
% x=1*2*(rand(1,30)-0.5)+ j*1*2*(rand(1,30)-.5);  % random real and imag (Fig. 6, 11a)
%  N=29;n=0:28;x=cos(3*2*pi*n/29)  % Fig. 7  N=29, with other adjustments
%  N=30; n=0:N-1;  % revert to length 30
% x= cos(14*2*pi*n/N);  % cos of freq 14/30 Fig. 8a, 12a
% x= cos(15*2*pi*n/N);  % cos of freq 14/30 Fig. 8b, 12b
% x= sin(15*2*pi*n/N);  % Fig. 8c
% x= sin(14*2*pi*n/N);  % Fig. 8d
% Fig. 11b, changes interp length to 120
% Fig. 11c derived from 11a

xr=real(x);
xi=imag(x);
X=fft(x);
XR=real(X);
XI=imag(X);
XM=abs(X);

figure(1)

subplot(421)
plot([-5 N+5],[0 0],'c')
hold on
plot(n,xr,'or')
hold off
xmax=max(xr);
xmin=min(xr);
axis([-5 N+5 1.2*xmin-.000001 1.2*xmax+.000001])
if max(abs(xr))<0.00001; axis([-5 N+5 -1.2 1.2]); end
title('SIGNAL x(n), REAL PART')
```

```
subplot(422)
plot([-5 N+5],[0 0],'c')
hold on
plot(n,xi,'og')
hold off
xmax=max(xi);
xmin=min(xi);
axis([-5 N+5 1.2*xmin-.000001 1.2*xmax+.000001])
if max(abs(xi))<0.00001; axis([-5 N+5 -1.2 1.2]); end
title('SIGNAL x(n), IMAGINARY PART')


subplot(423)
plot([-5 N+5],[0 0],'c')
hold on
plot([N/2 N/2],[-1000 1000],'m:')
plot([0 0],[-1000 1000],'m:')
plot(n,XM,'ok')
hold off
xmax=max(XM);
xmin=min(XM);
axis([-5 N+5 -0.2*xmax 1.2*xmax])
if max(XM)<0.00001;axis([-5 N+5 -.2 1.2]);end
title('FFT X(k), MAGNITUDE')


subplot(425)
plot([-5 N+5],[0 0],'c')
hold on
plot([N/2 N/2],[-1000 1000],'m:')
plot([0 0],[-1000 1000],'m:')
plot(n,XR,'or')
hold off
xmax=max(XR);
xmin=min(XR);
axis([-5 N+5 1.3*xmin 1.3*xmax])
if abs(min(XR))<0.00001; axis([-5 N+5 -0.3*xmax 1.3*xmax]);end
if max(abs(XR))<0.00001;axis([-5 N+5 -1.2 1.2]); end
title('FFT X(k), REAL PART')


subplot(426)
plot([-5 N+5],[0 0],'c')
hold on
plot([N/2 N/2],[-1000 1000],'m:')
plot([0 0],[-1000 1000],'m:')
plot(n,XI,'og')
hold off
xmax=max(XI);
xmin=min(XI);
axis([-5 N+5 1.3*xmin 1.3*xmax])
if abs(min(XI))<0.00001; axis([-5 N+5 -0.3*xmax 1.3*xmax]);end
if max(abs(XI))<0.00001;axis([-5 N+5 -1.2 1.2]); end
title('FFT X(k), IMAGINARY PART')

% Now interpolate

pad=100-length(x)
XX=[X(1:15) X(16)/2 zeros(1,pad-1) X(16)/2  X(17:30)]
xx=(100/30)*ifft(XX);
%
```

```
subplot(427)
plot([-15 115],[0 0],'c')
hold on
xr=real(xx);
plot([0:99],xr,'or')
plot([0:99],xr,':r')
hold off
xmax=max(xr);
xmin=min(xr);
axis([-15 115 1.2*xmin-.000001 1.2*xmax+.000001])
if max(abs(xr))<0.00001; axis([-15 115 -1.2 1.2]); end
title('INTERPOLATED x(n), REAL PART')


subplot(428)
plot([-15 115],[0 0],'c')
hold on
xi=imag(xx);
plot([0:99],xi,'og')
plot([0:99],xi,':g')
hold off
xmax=max(xi);
xmin=min(xi);
axis([-15 115 1.2*xmin-.000001 1.2*xmax+.000001])
if max(abs(xi))<0.00001; axis([-15 115 -1.2 1.2]); end
title('INTERPOLATED x(n), IMAGINARY PART')


figure(1)
```

Program 2 – This program produces the final graphs as indicated

```
% EN222b   testing time--->freq interp and freqz
% Fig. 1,2,3,4,5,6 in Matlab are Fig 13a,13b,14,15a,15b,15c of EE#222

h=[1 1 1 1 1   1   1 1 1 1 1];
H=freqz(h,1,500);
figure(1)
subplot(221)
plot([0:.001:.499],real(H),'r')
hold on
plot([-1 2],[0 0],'k:')
plot([0 0],[-20 20],'k:')
axis([-.05 .55 -2 14])
hold off
title('Real')
subplot(222)
plot([0:.001:.499],imag(H),'g')
hold on
plot([-1 2],[0 0],'k:')
plot([0 0],[-20 20],'k:')
axis([-.05 .55 -10 4])
hold off
title('Imaginary')
%
```

```
subplot(223)
plot([0:.001:.499],abs(H),'k')
hold on
plot([-1 2],[0 0],'k:')
plot([0 0],[-20 20],'k:')
axis([-.05 .55 -2 14])
hold off
title('freqz on Causal Rectangle Length 11')
%
%
h=[1 1 1 1 1   1   1 1 1 1 1];
Hncfz=ncfreqz(h,1,500);
figure(2)
subplot(221)
plot([0:.001:.499],real(Hncfz),'r')
hold on
plot([-1 2],[0 0],'k:')
plot([0 0],[-20 20],'k:')
axis([-.05 .55 -5 14])
hold off
title('Real')
subplot(222)
plot([0:.001:.499],imag(Hncfz),'g')
hold on
plot([-1 2],[0 0],'k:')
plot([0 0],[-20 20],'k:')
axis([-.05 .55 -2 14])
hold off
title('Imaginary')
subplot(223)
plot([0:.001:.499],abs(Hncfz),'k')
hold on
plot([-1 2],[0 0],'k:')
plot([0 0],[-20 20],'k:')
axis([-.05 .55 -2 14])
hold off
title('ncfreqz on NON-Causal Rectangle Length 11')


HFFT=fft(h);
figure(3)
subplot(221)
plot([0:10],real(HFFT),'or')
hold on
plot([-5 15],[0 0],'k:')
plot([0 0],[-5 15],'k:')
axis([-2 13 -2 15])
hold off
title('Real')
subplot(222)
plot([0:10],imag(HFFT),'og')
hold on
plot([-5 15],[0 0],'k:')
plot([0 0],[-5 15],'k:')
axis([-2 13 -2 15])
hold off
title('Imaginary')
subplot(223)
plot([0:10],abs(HFFT),'ok')
hold on
plot([-5 15],[0 0],'k:')
plot([0 0],[-5 15],'k:')
axis([-2 13 -2 15])
hold off
title('Just the FFT')
figure(3)
```

```
% FFT Interp - zeros at end
HFFTend=fft([h,zeros(1,989)]);
figure(4)
subplot(221)
plot([0:999],real(HFFTend),'r')
hold on
plot([-100 1200],[0 0],'k:')
plot([0 0],[-5 15],'k:')
hold off
axis([-75 1075 -5 15])
title('Real')
subplot(222)
plot([0:999],imag(HFFTend),'g')
hold on
plot([-100 1200],[0 0],'k:')
plot([0 0],[-10 15],'k:')
hold off
axis([-75 1075 -10 15])
title('Imaginary')
subplot(223)
plot([0:999],abs(HFFTend),'k')
hold on
plot([-100 1200],[0 0],'k:')
plot([0 0],[-5 15],'k:')
hold off
axis([-75 1075 -3 15])
title('FFT Interp - zeros at end')


%  FFT ZEROS IN MIDDLE
figure(5)
HFFTmid=fft([h(1:6),zeros(1,989),h(7:11)]);
subplot(221)
plot([0:999],real(HFFTmid),'r')
hold on
plot([-100 1200],[0 0],'k:')
plot([0 0],[-5 15],'k:')
hold off
axis([-75 1075 -5 15])
title('Real')
subplot(222)
plot([0:999],imag(HFFTmid),'g')
hold on
plot([-100 1200],[0 0],'k:')
plot([0 0],[-10 15],'k:')
hold off
axis([-75 1075 -10 15])
title('Imaginary')
subplot(223)
plot([0:999],abs(HFFTmid),'k')
hold on
plot([-100 1200],[0 0],'k:')
plot([0 0],[-5 15],'k:')
hold off
axis([-75 1075 -3 15])
title('FFT Interp - zeros in middle')
```

```
% FFT INTERP  Zeros at Beginning
HFFTbegin=fft([zeros(1,989),h]);
figure(6)
subplot(221)
plot([0:999],real(HFFTbegin),'r')
hold on
plot([-100 1200],[0 0],'k:')
plot([0 0],[-5 15],'k:')
axis([-75 1075 -5 15])
hold off
title('Real')
subplot(222)
plot([0:999],imag(HFFTbegin),'g')
hold on
plot([-100 1200],[0 0],'k:')
plot([0 0],[-10 15],'k:')
hold off
axis([-75 1075 -10 15])
title('Imaginary')
subplot(223)
plot([0:999],abs(HFFTbegin),'k')
hold on
plot([-100 1200],[0 0],'k:')
plot([0 0],[-5 15],'k:')
hold off
axis([-75 1075 -3 15])
title('FFT Interp - zeros at beginning')


figure(1)

*****************************************************************
```