1 Properties of The Discrete Fourier Transform

We saw last class that the N-point Discrete Fourier Transform (DFT) $X[k]$ can be obtained by sampling the Discrete-time Fourier Transform (DTFT) $X(e^{j\omega})$ every $2\pi/N$ in ω, i.e.,

$$X[k] = X(e^{j\omega})|_{\omega=(2\pi/N)k} \qquad \text{for } k = 0, 1, \ldots N-1.$$

We also saw that sampling in frequency produces a signal which is periodic in time by adding copies of $x[n]$ every N samples. Therefore, whenever we work with the DFT, we have to bear in mind that we are implicitly working with a periodic time signal. This implied periodicity does add some new twists to the usual Fourier properties. These notes describe how the modulation/shifting properties and the multiplication/convolution properties change a bit for DFTs. The final section presents a common application of the DFT which illustrates the importance of understanding the DFT properties clearly. Overlap-add block convolution can be used to compute the output of an FIR filter $h[n]$ when the input signal $x[n]$ is of unknown, and possibly indeterminate, length. Using the DFT to implement the convolutions for each block can be very fast, but requires a clear understanding of how to use the circular convolution implemented by the DFT to get the same convolution result that we would get with the standard linear convolution we saw earlier in the semester.

2 Circular Shifts and the DFT

This example examines what happens when the DFT $X[k]$ is multiplied by a complex exponential. Our Fourier transform intuition suggests that multiplying by a complex exponential in frequency is the same as shifting in time. We will see that this is basically right, but there is an additional twist caused by the lurking periodicity of the time-domain signals when using the DFT. Consider the 8-point finite length signal shown in Figure 1. We can compute the 8-point DFT as shown below, then multiply it by a complex exponential to obtain a new DFT $Y[k]$.

$$X[k] = \sum_{n=0}^{7} x[n]e^{-j(2\pi/8)kn} \tag{1}$$

$$Y[k] = e^{-j3(2\pi/8)k}X[k] \tag{2}$$

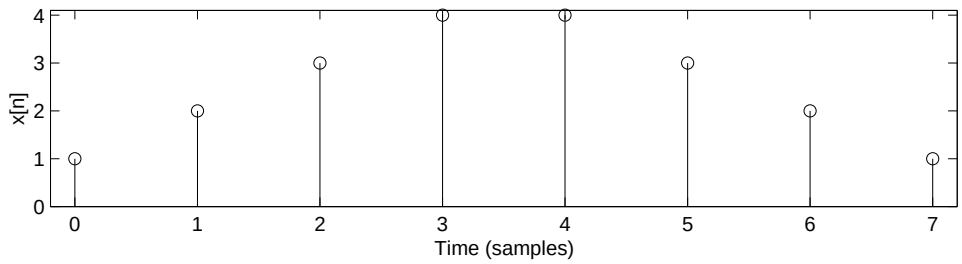

Figure 1: 8-point signal $x[n]$ for Example 3

Substituting $Y[k]$ into the inverse DFT equation gives a new finite-length time signal $y[n]$, as shown in Figure 2. This signal signal does appear shifted, but perhaps not quite as expected. Instead of moving all the values three samples to the right, it appears as though the values shifted past $n = 7$ reappear on the left edge of the signal. You can visualize this by imagining the signal on a piece of paper wrapped around a can, then shifted by three samples. After shifting, the paper is cut off the can at the new $n = 0$ point, yielding the signal shown in Figure 2.

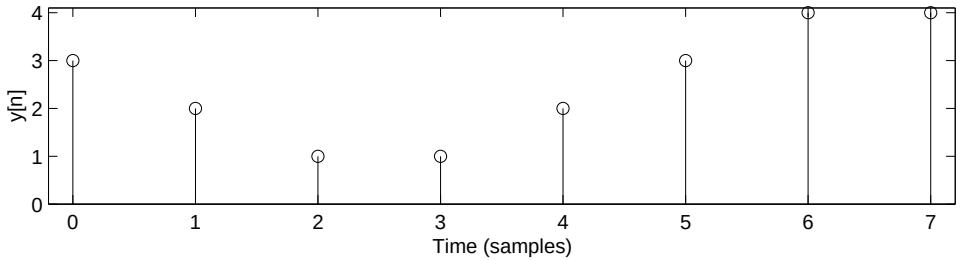

Figure 2: 8-point signal $y[n]$ for Example 3

Figure 3 uses the three step procedure described earlier to demonstrate what happens in this example. The top panel shows the periodic signal $\tilde{x}[n]$ generated by copying $x[n]$ every $N = 8$ samples as described in Step 1. The dashed lines indicate where the "primary" period $0 \leq n \leq 7$ falls. For Step 2, multiplying by a complex exponential in frequency causes a shift in time, which produces the signal $\tilde{y}[n]$ shown in the bottom panel of Figure 3. Finally, for Step 3, only the period $0 \leq n \leq 7$ between the dashed lines are kept for the new finite length signal $y[n]$. The signal between the dashed lines exactly matches the one from Figure 2. These pictures make clear why the circular shift occurs, and how the values "leaving" the right edge of the interval $0 \leq n \leq 7$ reappear on the left edge of the interval.

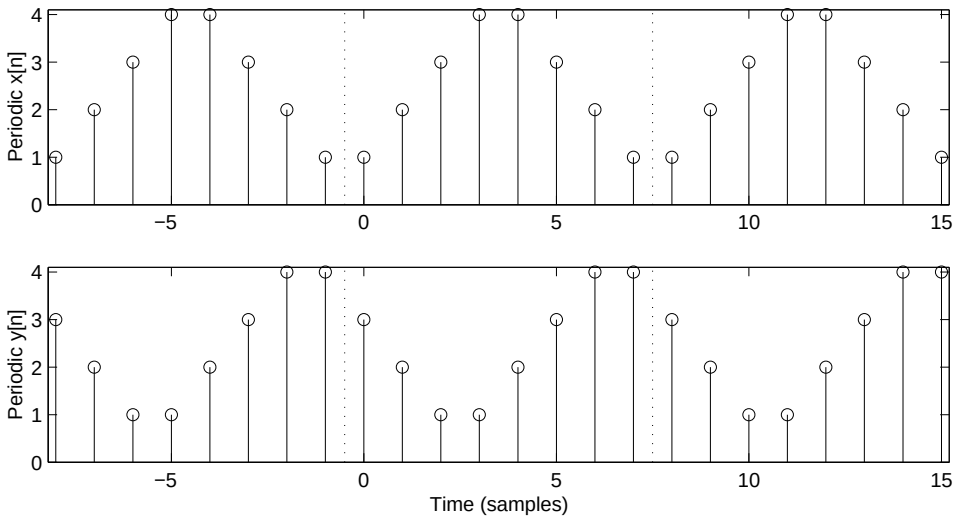

Figure 3: Periodic signals $\tilde{x}[n]$ and $\tilde{y}[n]$ for Example 3

If the DFT size is large enough, the resulting shifted signal will be exactly what we expect from the DTFT. Repeating the process in Eqs. (1) and (2) above, except with $N = 16$ instead of $N = 8$, produces the signal $y[n]$ shown in Figure 4. This signal does not show any evidence of a circular shift. Considering the start of the signal as $n = 0$, the shifted signal only has a length of 11, so the DFT size $N = 16$ is large enough to prevent any periodic effects from occurring. Another perspective on this is that the DFT size is large enough that the main period of the signal perfectly matches the finite-length signal through all the operations. This would not work if the exponential in Eq. (2) were $e^{j(2\pi/N)3k}$ instead of $e^{-j(2\pi/N)3k}$. In that case, the shift would be to the left, immediately causing signal values to fall of the left edge and reappear at the right edge, regardless of the DFT size.

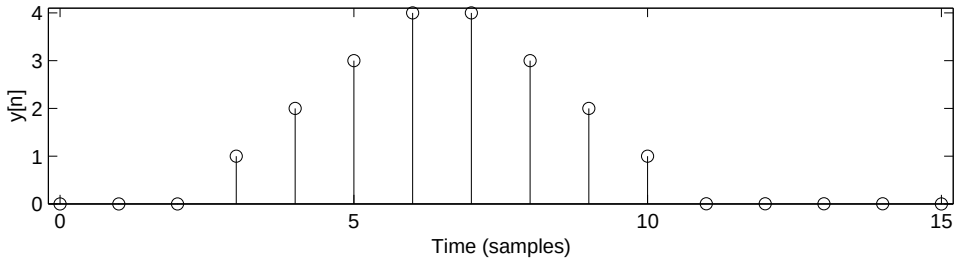

Figure 4: Shifted signal $y[n]$ for increased DFT size $N = 16$ in Example 3.

The last example demonstrates an important consideration in using the DFT to process finite length signals. The DFT size N must not only be longer than the input signal $x[n]$, but must be longer than any of the intermediate signal or the output signal as well. This issue often arises when using the DFT to implement convolution in the time domain. The computational speed of the FFT makes the DFT an attractive technique for implementing the convolution of long signals. When both $x[n]$ and $h[n]$ are long, it is often faster to compute the output $y[n]$ using the DFT than directly computing the convolution sum

$$y[n] = \sum_{\ell=0}^{N-1} x[\ell]h[n-\ell]$$

in the time domain. Our experience with the Fourier transform makes us expect that to convolve signals in time, we should multiply the DFTs of the signal. This is exactly what we do. We first compute $X[k]$ and $H[k]$ with the FFT, multiply them to form $Y[k] = H[k]X[k]$, then use the inverse FFT to find $y[n]$. The implied periodicity of the DFT is again an issue here. If we want to get the same output $y[n]$ obtained from linear convolution, we must be careful to make sure the DFT size N is not only long enough to avoid aliasing for both $x[n]$ and $h[n]$, but also to avoid aliasing the longer output $y[n]$.

3 Convolving signals using the DFT

This example compares two approaches to computing the convolution. The first approach is the standard use of the convolution sum from Chapter 2. For an M-point input $x[n]$ and P-point impulse response $h[n]$, the output $y_{lin}[n] = x[n] * h[n]$ can be computed using the convolution sum as

$$y_{lin}[n] = \sum_{\ell=0}^{N-1} x[\ell]h[n-\ell].$$

From our knowledge of convolution from Chapter 2, we know that the output $y[n]$ is at most $M + P - 1$ points long. The other approach to computing the convolution is to compute the N-point DFT $X[k]$, define $Y[k] = H[k]X[k]$, then compute the inverse DFT $y[n]$. The essential question is how large N must be such that the DFT-based procedure gives the same result as $y_{lin}[n]$. Multiplying two DFTs does implement convolution in the time domain, but a form of convolution called circular convolution, which is convolving the periodic versions of the signals created by sampling in frequency to obtain the DFT. Because the convolution $y[n] = x[n] * h[n]$ makes an output which can be longer than either input, if we make a poor choice of N, it is possible to alias the output $y[n]$ even if neither $x[n]$ or $h[n]$ is aliased. To get the circular convolution result to match the standard convolution result, the DFT size must be larger than the largest possible length for $y[n]$, so $N \geq M + P - 1$.

To illustrate these issues, we will look at the convolution of a rectangular pulse $x[n]$ (shown in Figure 5) with itself, so $y_{lin}[n] = x[n] * x[n]$. Because $x[n]$ is a rectangular pulse 5 points long, the convolution sum $y_{lin}[n] = x[n] * x[n]$ will be a triangle which is $5+5-1 = 9$ points long, and with a peak height of 5. Figure 6 shows the signal obtained using the Matlab `ylin = conv(x,x)` command to implement the convolution sum.

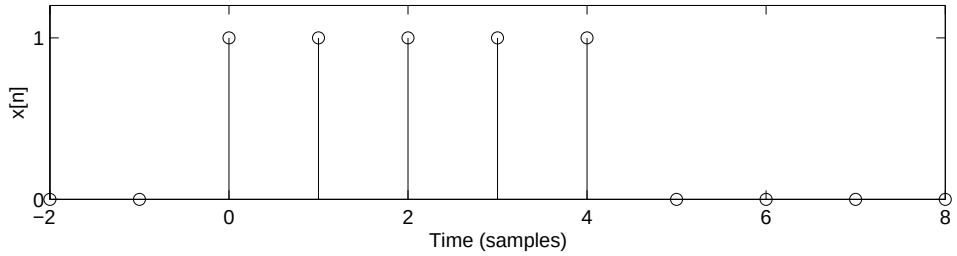

Figure 5: 5-point rectangular pulse $x[n]$ for Example 2

Now let's look at computing the time-domain convolution by multiplying DFTs. If we only take the length of $x[n]$ into account when choosing the DFT size N, and do not consider the length of the output $y[n]$, we would choose $N = 5$. Figure 7 shows the aliased output that results. In fact, the triangles in $y_{lin}[n]$ have overlapped in the periodic copies so that they add up to a constant for all n. Clearly, the output $y[n]$ in Figure 7 and the output $y_{lin}[n]$ in Figure 6 are not the same.

If we increase the DFT size to $N = 8$, the aliasing is reduced because $y_{lin}[n]$ is 9 points long, there is still one point of aliasing, where the last point of the triangle aliases

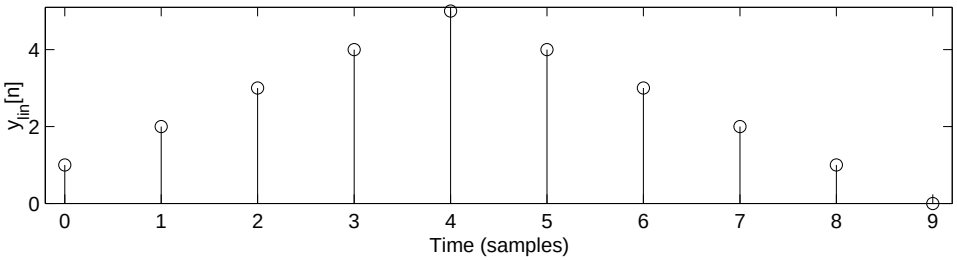

Figure 6: Triangle output signal $y_{lin}[n]$ for Example 4.

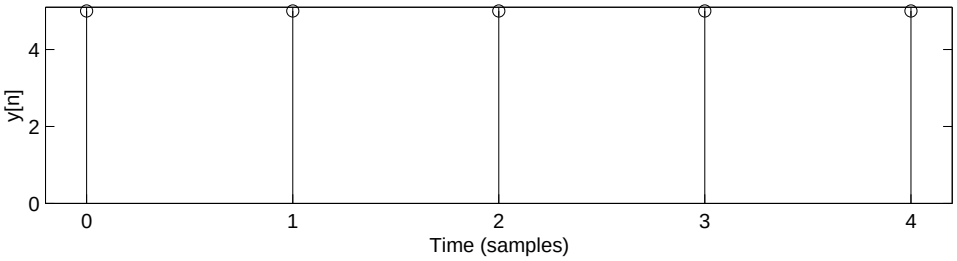

Figure 7: Aliased output signal $y[n]$ caused by DFT size $N = 5$ in Example 4.

on top of the first point. Figure 8 illustrates the output $y[n]$ resulting for this DFT size. The signal is closer to $y_{lin}[n]$, but is missing the point at $n = 8$ and has the wrong value at $n = 0$ due to aliasing.

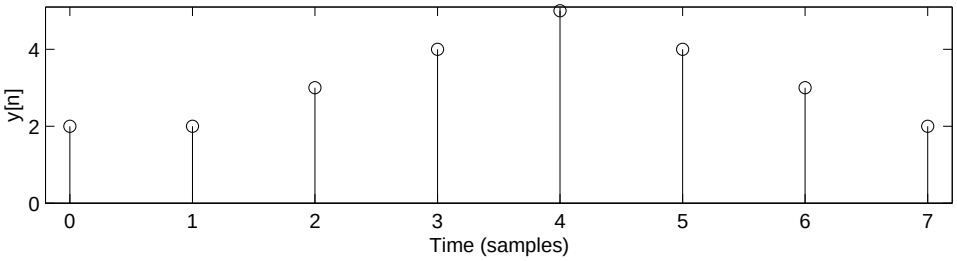

Figure 8: Aliased output signal $y[n]$ caused by DFT size $N = 8$ in Example 4.

Finally, if we increase the DFT size to $N = 10$, the result of computing the convolution with the DFT is the same as with the convolution sum. The DFT size is larger than the length of $y_{lin}[n]$, so the output is no longer aliased. Figure 9 plots the signal that results for this case. This matches $y_{lin}[n]$ in Figure 6 exactly.

This example illustrated the most important point when using the DFT to implement time-domain convolution. The DFT size must be at least as big as the longest possible output signal $y[n]$.

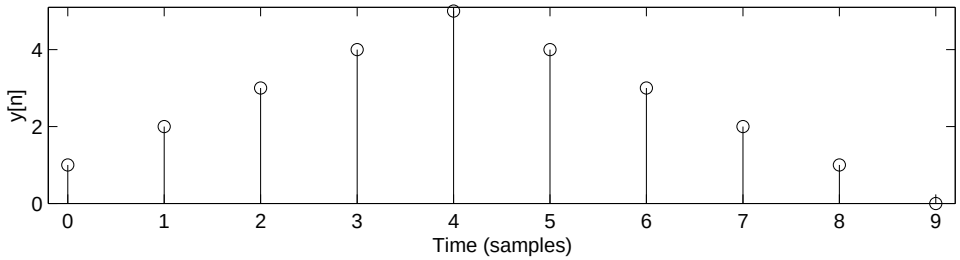

Figure 9: Unaliased output signal $y[n]$ produced by DFT size $N = 10$ in Example 4.

4 Overlap-add Block Convolution with the DFT

A common application in practical signal processing is filtering a very long signal $x[n]$ with an FIR filter $h[n]$. One way to do this would be to compute very large DFTs for $X[k]$ and $H[k]$, then multiply these together to get $Y[k] = H[k]X[k]$, then compute the IDFT to get $y[n]$. If the DFT size is larger than the length of $y[n]$, this should get the right answer. There are several drawbacks to this approach, though. First, we might not know how long $x[n]$ is in advance, and it may effectively be infinite if we are processing something like an audio stream in real time. Second, the DFT approach requires that we wait until we have received all of the input $x[n]$ before we can find any of the output $y[n]$, since we need all of $x[n]$ to compute $X[k]$. In many situations, we cannot wait this long. Also, if $x[n]$ is very long, it may require too much memory to process all at once. One solution to these problems is to use the block convolution approach.[1] In block convolution, we break the input signal into blocks of length L, compute the convolution of each block with $h[n]$ separately, then add the output blocks together to get $y[n]$.

Let's describe the algorithm with more mathematical detail. First, we break the input $x[n]$ into blocks of length L, and reindex each block to count from $n = 0, \ldots, L-1$. Define the rth block of the input $x[n]$ to be

$$x_r[n] = \begin{cases} x[n + rL], & 0 \leq n \leq L-1 \\ 0, & \text{otherwise,} \end{cases}$$

for blocks $r = 0, 1, 2, \ldots$. Using this definition, we can write the overall signal $x[n]$ as

$$x[n] = \sum_{r=0}^{\infty} x_r[n - rL]$$

Figure 10 illustrates breaking the signal into blocks of length L. The top panel shows the entire signal $x[n]$, and the three panels below illustrate how the first three blocks are extracted, and how they have been re-indexed to start from $n = 0$ in each block. It is clear that adding the three bottom panels as they are shown with the correct delays would produce the signal in the top panel. Although the signals are discrete-time signals, we've plotted them here with a continuous line since they are long enough that they would look awkward using the `stem` style of plotting.

[1] Remember we saw this in Matlab I earlier this semester.

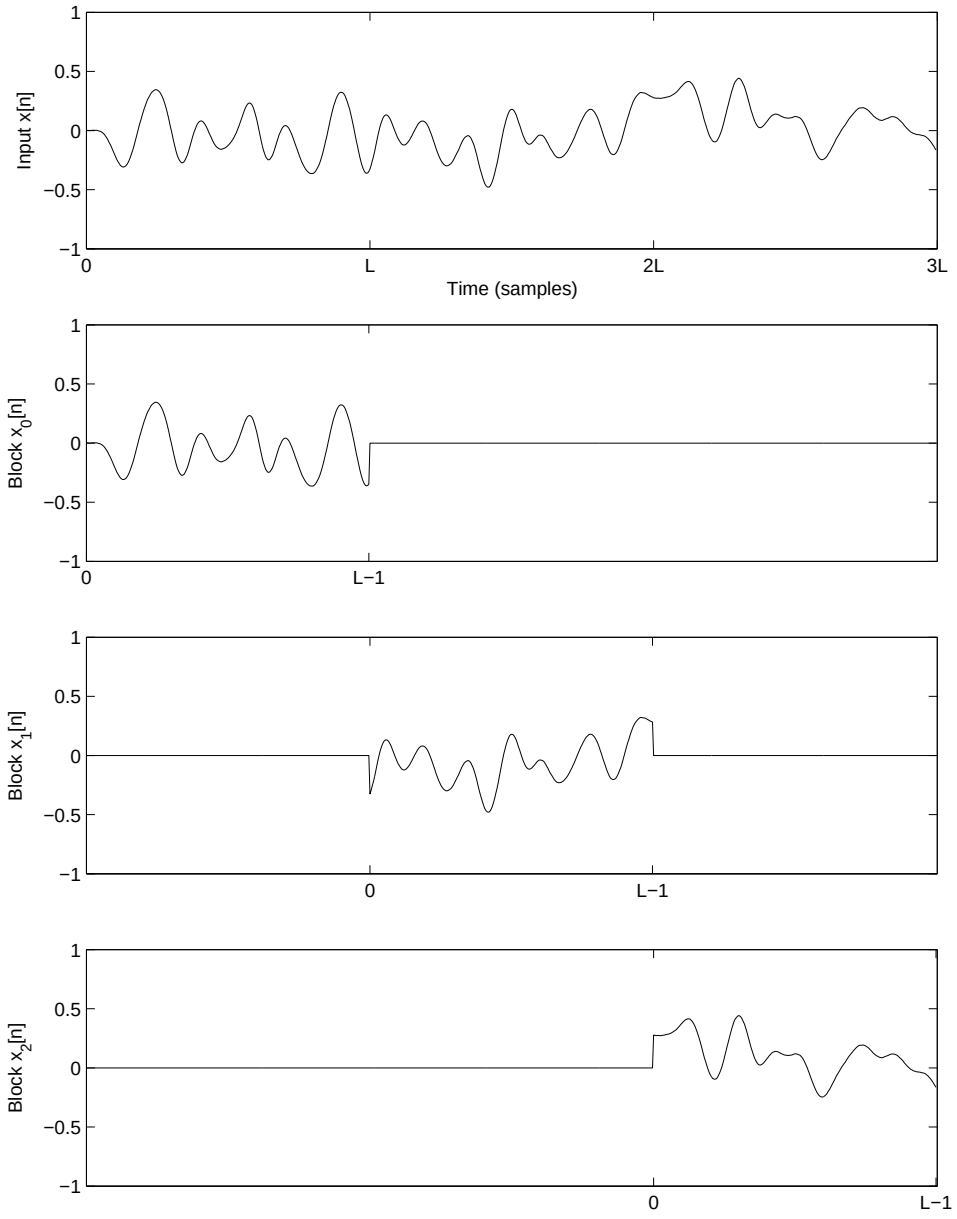

Figure 10: Illustration of breaking signal $x[n]$ into non-overlapping blocks $x_0[n], x_1[n], x_2[n], \ldots$

The distributive property of convolution illustrates how we can compute the convolution for each block then reassemble them to get the overall convolution output.

$$\begin{aligned} y[n] &= h[n] * x[n] && (3)\\ &= h[n] * \left(\sum_{r=0}^{\infty} x_r[n - rL] \right) && (4)\\ &= \sum_{r=0}^{\infty} \left(h[n] * x_r[n - rL] \right) && (5)\\ &= \sum_{r=0}^{\infty} y_r[n - rL] && (6) \end{aligned}$$

where $y_r[n]$ is the rth output block obtained by convolving the rth input block $x_r[n]$ with the impulse response $h[n]$.

The DFT can be a very fast way to compute the convolution for each $y_r[n]$ block of the output. Specifically, the Fast Fourier Transform, or FFT, computes the DFT very efficiently, especially when the DFT size N is a power of 2. As we saw in the last section, though, it is important to have the DFT size be at least as big as the output signal expected. Assume the FIR filter $h[n]$ is P points long, so $h[n] = 0$ for $n < 0$ and $n > P - 1$. We know that the output $y_r[n]$ will have length $L + P - 1$ since convolving two finite length signals produces an output that is one less than the sum of the input lengths. This gives a constraint for the relationship between the filter size P, input block size L and the DFT size N such that the circular convolution gives the same result for each block as the linear convolution does:

$$N \geq L + P - 1.$$

In practice, the filter size P is generally fixed, and we choose a suitably large power of 2 for N, then solve backward to get L as

$$L = N - P + 1.$$

The output blocks will each be $L + P - 1$ samples long, but Eq. (6) shows that we only shift each output block $y_r[n]$ by L samples more than the previous block. Therefore, the start and end of each output block will overlap with the previous and next block. These overlapping samples need to be added to find $y[n]$ in the overlapping regions. This gives this block convolution technique the name overlap add. Figure 11 shows a sample of how the blocks overlap at the output before adding them together.

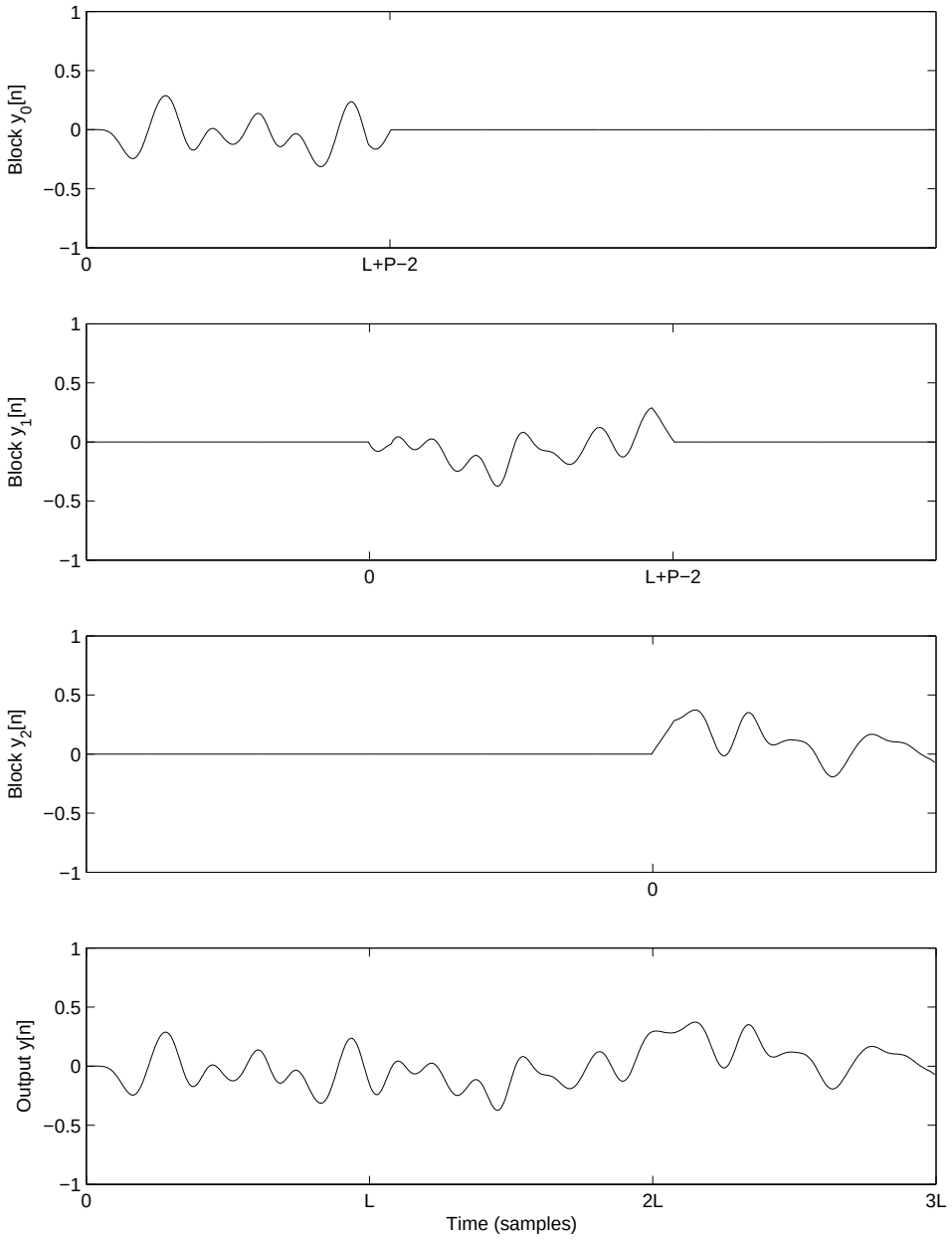

Figure 11: Illustration of output blocks $y_0[n], y_1[n], y_2[n], \ldots$ in top three panels which add to give $y[n]$ in the bottom panel.