

FIR filter design using the DFT

Topics: FIR filter design, DFT

The DFT can be used for quickly designing filters: simply specify the filter you want in the frequency domain, and take the inverse DFT to get the coefficients. Its simplicity hides the main flaw, that the filter designed in this way will only meet the desired response *at the DFT sampled frequencies*. At other frequencies, the response can and will oscillate pretty far from the ideal response.

First, here's how to design a filter using the DFT. Suppose we want a FIR filter with M coefficients. For illustrative purposes, let us require it to be a lowpass filter, which ideally passes frequencies less than F_c cycles/sample, and blocks all others. Then the DFT of the M coefficients should look as in Figure 1, where $F_c=0.25$ has been chosen as an example:

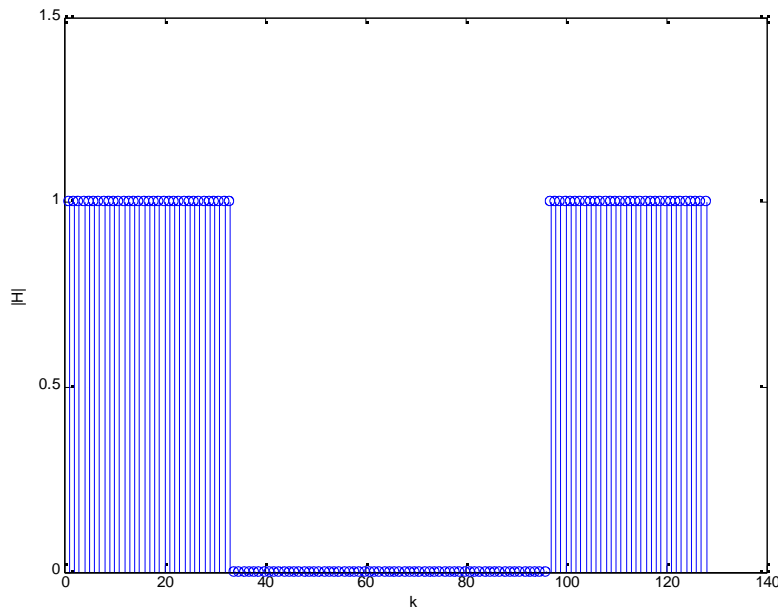


Figure 1: The DFT of a low pass filter, with $F_c=0.25$, and having 128 coefficients. Note that $H[k]=1$ for $k \leq 31$ and $k \geq 97$, and is zero otherwise.

The frequency response has to have mirror symmetry around Nyquist (index 64) to correspond to a real-valued filter. We can immediately find the filter coefficients by computing the inverse DFT (and throwing away the tiny imaginary part that creeps in due to numerical error.) The MATLAB command is “`h=real(iff(H))`”. The result is plotted in Figure 2.

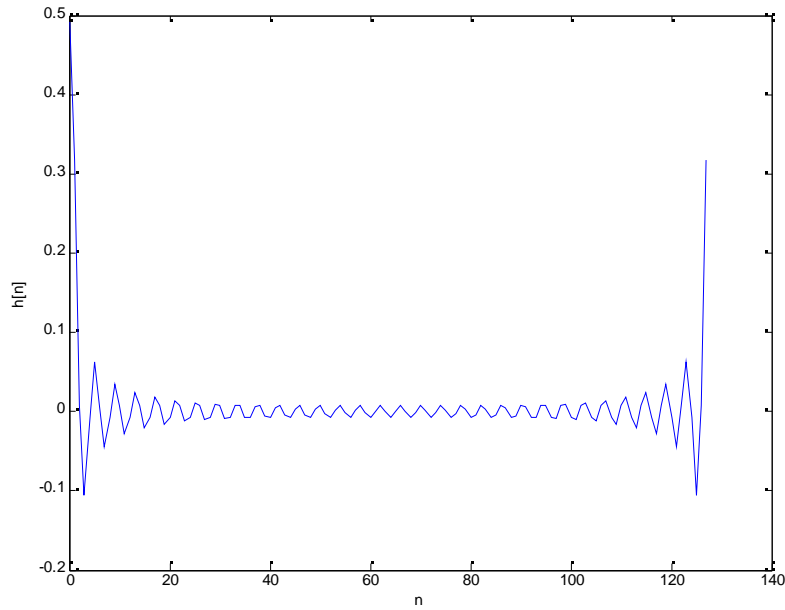


Figure 2: Filter coefficients (impulse response) obtained by taking the inverse DFT of the filter in Figure 1.

Notice that the response is mirror-symmetric around the midpoint. This is due to the periodicity inherent in the DFT. To obtain the filter properly centered, we can simply swap the right and left halves: “`h=fftshift(h)`” in MATLAB. See Figure 3.

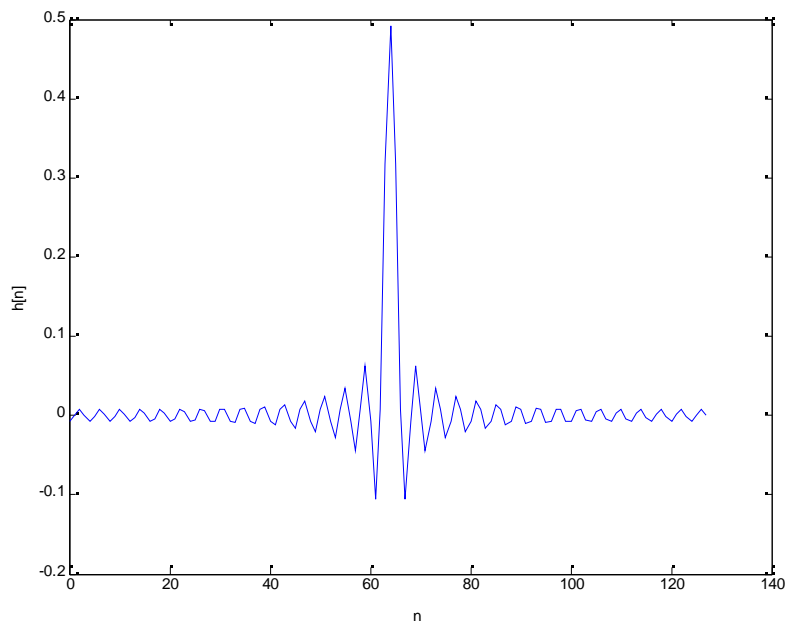


Figure 3: Impulse response of the filter after centering

This centering operation changes the phase. It is now $-2\pi k 64/128 = -\pi k$, for each $H[k]$. This means that the group delay is 64 samples at all frequencies: therefore, the filtered output lags the input by 64 samples.

Figure 3 shows that the filter we obtained is simply a sinc() function, without any tapering window. We should suspect that this will not actually achieve the ideal response at *all* frequencies. Consider Figure 4:

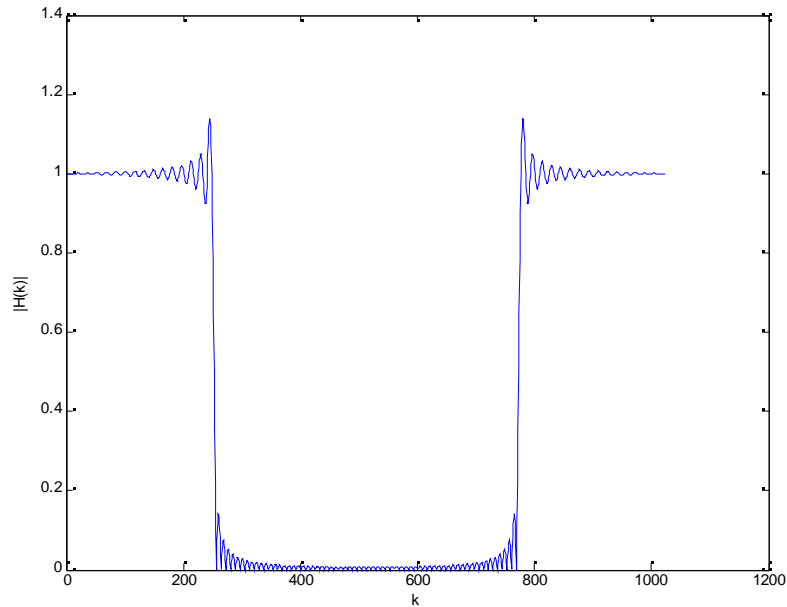


Figure 4: Frequency response zero-padded to 1024 samples.

Aside from the frequencies sampled by the original 128-point DFT, the frequency response is far from ideal. See Figure 5.

The DFT method has its advantages, however: it is quick to understand, and easy to generalize to any shape of filter (ie. “arbitrary” shape). As long as we keep in mind that the response is only guaranteed to be what we specified at the original sampled frequencies, this method of filter design is a useful DSP tool.

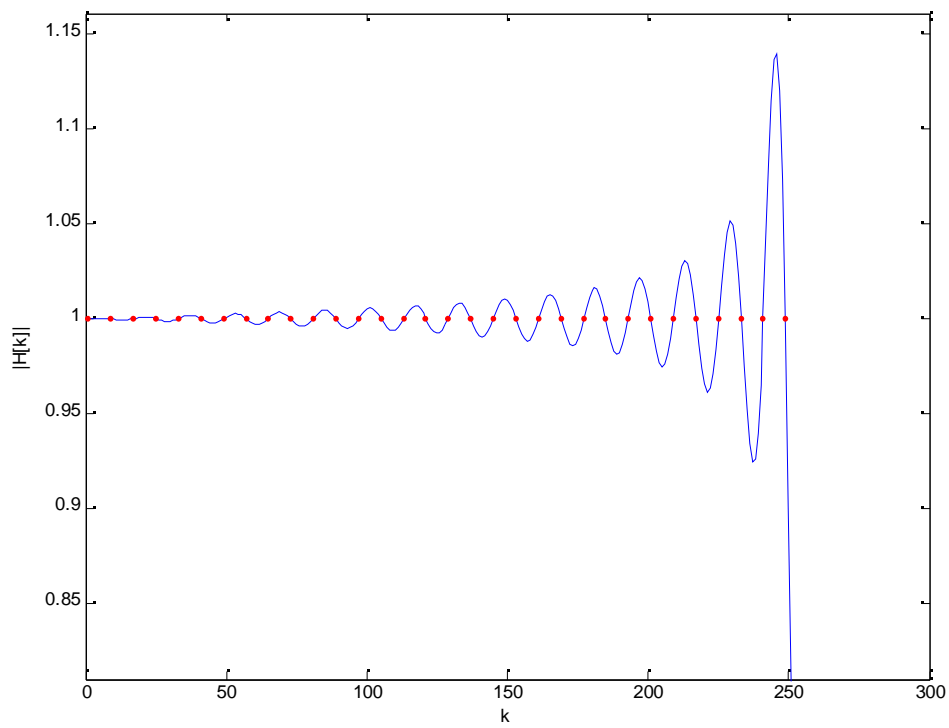


Figure 5: Close up of the response in Figure 4, where the frequencies sampled by the original 128 point DFT are shown by dots