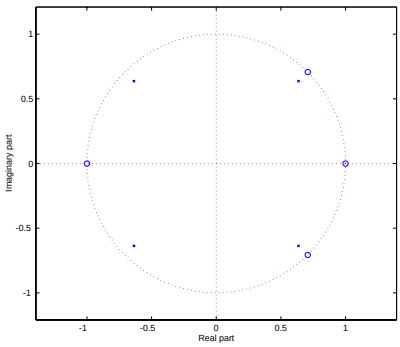
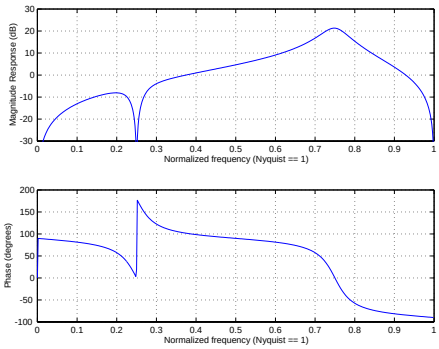
1. (a) In this direct-form II second-order-section filter, the first stage has a transfer function $H_1(z) = \frac{1 - 1.41z^{-1} + z^{-2}}{1 + 1.27z^{-1} + 0.81z^{-2}}$ with zeros at $z = e^{\pm j\pi/4}$ and poles at $z = 0.9e^{\pm j3\pi/4}$. The second stage has a transfer function $H_2(z) = \frac{1 - z^{-2}}{1 - 1.27z^{-1} + 0.81z^{-2}}$ (note sign change in denominator) with zeros at $z = \pm 1$ and poles at $z = 0.9e^{\pm j\pi/4}$. Thus, the pole-zero diagram of the product of these two stages looks like the union of the two sets of roots, i.e.:





(b) The actual frequency response, courtesy of Matlab, is shown to the right of the p-z diagram. I was principally looking for the three zeros (at $\omega = 0$, $\pi/4$ and π) and the π phase jump as the system goes through the zero at $\pi/4$. Peaking around the pole at $3\pi/4$ was nice too, along with some kind of reaction in the phase function.

(c) Here, what I was getting at was the idea of distributing the poles and zeros in second-order-sections so that the poles and zeros in each stage most nearly cancel out (limiting the risk of large intermediate values that might overflow computation). We would want to swap the feedback coefficients (denominators) in the illustrated system to achieve this.

Comments: Many people made slips of varying scales on factorizing the system functions; I graded this generously as long as the frequency response was consistent with the pole-zero diagram (although having poles outside the unit circle means the DTFT doesn't exist, so there would be nothing to sketch).

2. (a) This is the pole-zero diagram of a Chebyshev-I continuous-time low-pass filter, with the characteristic elliptic pole locus.

(b) Chebyshev-I filters are defined by their **order** ($N = 8$ in this case, from counting the poles), their **cutoff frequency** (around $\Omega = 10$ from the frequency of the last pole), and the **depth of the ripples** in the passband, which we don't know; however, we can estimate it as the ratio of

the difference in the length of the vectors from the $j\Omega$ axis to the two nearest poles at $\Omega = 0$ (a minimum in the ripples) compared to a value of Ω closest to one of those poles (ripple maximum, around $\Omega = 2$). By eye, the two poles closest the origin are at about [-1.8 ±j2], so this ratio would be $(1.8^2 + 2^2)/(1.8 \cdot \text{sqrt}(1.8^2 + 4^2)) = 0.92$ or about -0.75 dB. In fact, the filter was designed with 1 dB passband ripple: `>> [b,a]=cheby1(8, 1, 10, 's');`

Note that standard design constraints such as stopband edge and minimum stopband attenuation don't enter into the specification of a Chebyshev I filter, except via the filter order.

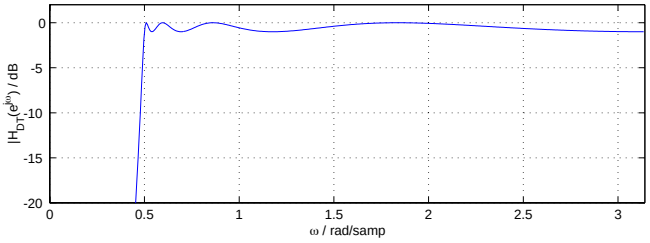
(c) There are a couple of paths by which we could convert this filter to a DT high pass. The simplest is:

- Pre-warp the DT band edge $\omega_c = 0.5$ rad/samp to the CT domain via the inverse of our standard bilinear transform warp, so $\Omega_c = \tan(\omega_c/2) \approx 0.2553$.
- Transform our prototype low-pass CT filter polynomial $H_{LP}(s)$ to a high-pass filter with the new passband edge through the s-plane transformation: $H_{HP}(s) = H(\hat{s})\Big|_{\hat{s} = \frac{\hat{\Omega}_c \Omega_c}{s}}$

  where Ω_c is the desired cutoff 0.2553 and $\hat{\Omega}_c$ is the cutoff of the prototype filter, 10 rad/sec in this case. This substitution results in a polynomial with both poles and zeros. In Matlab, `[bh,ah]=lp2hp(b,a,0.2553*10);`
- Convert the new CT highpass polynomial into a DT filter via the bilinear transform, $H_{DT}(z) = H_{CT}(s)\Big|_{s = \frac{z-1}{z+1}}$. This gives a polynomial in z of the same order (still 8), with

  both poles and zeros, and warps the cutoff frequency back to our original design value of 0.5 rad/samp. In Matlab, `[bd,ad]=bilinear(bh,ah,0.5);` (where we always use 0.5 as the transform sampling rate).

The result of this transformation is a filter with the same magnitude and phase values as the original CT LP prototype, (i.e. $H_{DT}(e^{j\omega}) = H_{LP}(j\Omega)$ for some ω and Ω) but with an extensively remapped frequency axis. However, the ripples in the pass band, and the monotonicity of the stop band, are preserved. In particular, there will still be 4 ripples in the pass band, arising from the 4 poles in each half of the z-plane, as shown in the plot below:



Comments: This question was pretty straightforward, so it was graded more strictly. It was not sufficient to simply write down the variable substitutions for the two transformations, but I wanted to see evidence that you understood how they were substituted to create new polynomials defining the new filters.

3. (a) This question is simply asking how to delay a signal by 1.25 milliseconds. The problem is that, with the 10 kHz sampling rate specified, this corresponds to 12 $^1/_2$ samples i.e. not an integer number of samples, so not something that can be created with a simple chain of unit delay elements.

   We know the DTFT of the system we want to produce: it has a magnitude of 1 for all frequencies, and a linear phase corresponding to the 12.5 sample delay i.e. $H(e^{j\omega}) = e^{-j12.5\omega}$. Thus, it seems like we should be able to find a filter that has this response. One approach would be to sample the DTFT to give a DFT $H[k]$ of some length, then calculate its IDFT to get an impulse response. This runs the risk of unbounded magnitude error in-between the DTFT samples (i.e. the Gibbs' phenomenon kind of effect), but it's not a bad start. One problem lies with the sample at $\omega = \pi$; this should be $e^{-j12.5\pi} = -j$, but that would break the conjugate symmetry of the response; for a real filter, this single magnitude point must be set to zero, with the implication that at least some high frequency information will be lost.

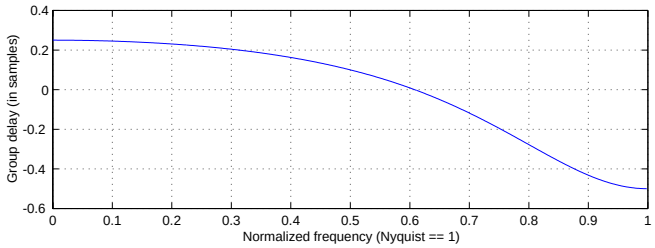
   What I was hoping for as an answer was that the half-sample delay would bring to mind the type II (and type IV) linear phase FIR filter types, which have a point of symmetry half way between two samples, and thus an effective delay of $N/2+^1/_2$ (where N is the filter order). Thus, a 12.5 sample delay can be implemented very effectively with a 24 point (23rd order) symmetric FIR filter. Consistent with the previous approach, we found that type-II linear phase filters are obliged to have a zero at $\omega = \pi$, so have an implicit low-pass nature. We could design the filter with Parks-McClellan.

   Another approach to part (a) is to note that if only the sampling rate were twice as high (20 kHz), then the filter would be trivial: just a chain of 25 simple delay elements. Thus, we could implement the delay by upsampling the signal to twice the sampling rate, applying the delay, and downsampling again. The only thing that makes this complicated is that we are obliged to low-pass filter the delayed result prior to downsampling, else the aliasing will completely cancel our result. The net effect of the upsample-delay-anti-alias-downsample is an FIR filter consisting of every odd-numbered sample of the half-band low-pass anti-alias filter from the upsampled domain; one fairly crummy choice is an ideal sinc lowpass filter with some kind of tapered window to truncate it; the odd-numbered samples of this will form a symmetric sinc-like profile, which will end up looking very similar to the previous two approaches.

   (b) By halving the sampling rate, we end up looking for a 6.25 sample delay. The point of this part of the question was to make people who had arrived at the type-II FIR filter solution for part (a) think a little deeper. Perhaps the easiest way to think about designing this filter is to upsample by 4, delay by 25 samples, low-pass at $\pi/4$, then downsample by 4.

Comments: I think only one person came up with the type-II LP FIR approach I had been intending. Many people thought fractional delays were impossible, and that we should make do with 12 samples; some people suggested a weighted average of delays of 12 and 13 samples (which is of course a first-order type-II LP FIR filter, but with a very poor magnitude response compared to the longer filters designed by the above methods). Several people went on to suggest a 6.25 sample delay as $.75z^{-6} + .25z^{-7}$. This doesn't work so well; in Matlab, `grpdelay([.75 .25])` will plot for you the effective delay this introduces as a function

of frequency, which falls from the intended 0.25 samples near d.c. to -0.5 samples at the Nyquist frequency, as shown below:



4. (a) The first part of this question is just warming you up to the idea of the fast Fourier transform algorithm for a radix other than two. Given the DFT definition, $X[k] = \sum_{n=0}^{N-1} x[n]W_N^{nk}$ where $W_N = e^{-j2\pi/N}$, and given $N = 3M$ where M is an integer, we can break the DFT summation into three terms,

$$X[k] = \sum_{n=0}^{M-1} \left(x[3n]W_N^{3nk} + x[3n+1]W_N^{(3n+1)k} + x[3n+2]W_N^{(3n+2)k} \right)$$

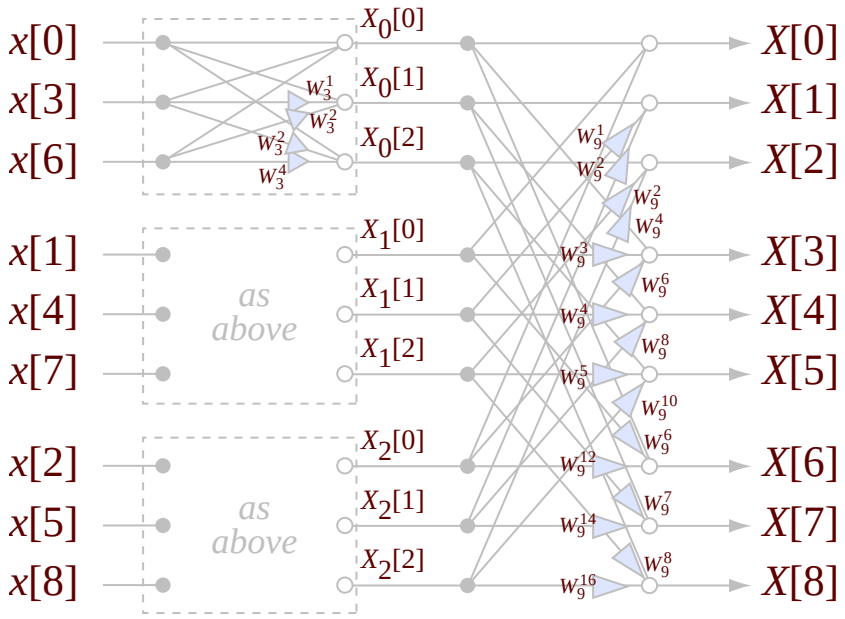
which we can rearrange as

$$X[k] = \sum_{n=0}^{M-1} x_0[n]W_M^{nk} + W_N^k \sum_{n=0}^{M-1} x_1[n]W_M^{nk} + W_N^{2k} \sum_{n=0}^{M-1} x_2[n]W_M^{nk}$$

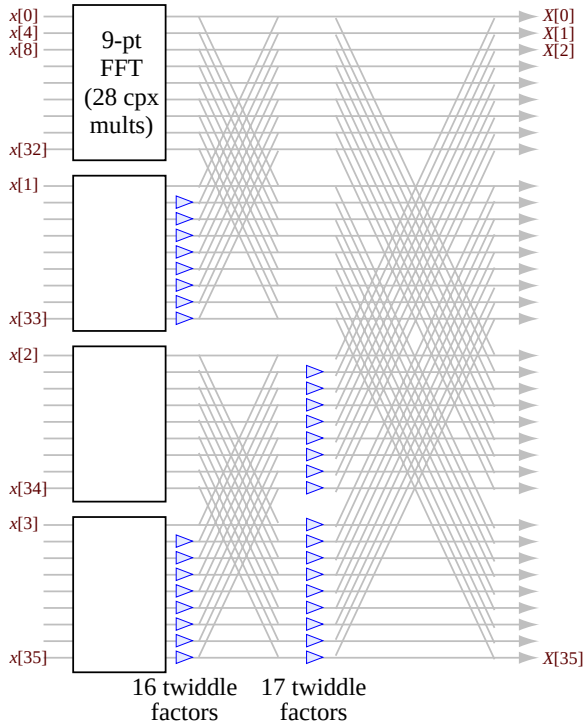
where $W_M = W_N^3 = W_{N/3}$ and $x_i[n] = x[3n+i]$ for $i = 0, 1, 2$.

Thus, $X[k] = X_0[\langle k \rangle_M] + W_N^k X_1[\langle k \rangle_M] + W_N^{2k} X_2[\langle k \rangle_M]$ for $k = 0$ .. N-1, where $X_i[k]$ is the M-point DFT of $x_i[n]$, for $i = 0,1,2$.

(b) We can calculate a 9-point DFT by applying the factor-of-3 decimation from the part (a) on a first stage of three 3-point DFTs. Thus, according to the above equation, each final output value $X[k]$ is the sum of one value from each of the first-stage 3-point DFTs $X_0[k]$, $X_1[k]$ and $X_2[k]$, with appropriate twiddle factors (that depend on the particular k we are looking at) applied to the values from $X_1[k]$ and $X_2[k]$. To draw the compete flowgraph, we also need to include all the operations required for the 3-point DFTs, which we can write down directly from the DFT equation. Hence, the flowgraph below:



It's certainly a bit of a mess to draw, but in fact the twiddle factors are quite simple to work out: Each $X[k]$ is the sum of three components, one from X_0, one from X_1 and one from X_2. The component from X_1 is, according to the equation from part (a), scaled by W_9^k, and the component from X_2 is scaled by W_9^{2k}.



(c) A 36 point DFT is most efficiently decomposed into two radix-3 stages (i.e. four 9 point DFTs in parallel) followed by either two radix-2 stages or a single radix-4 stage. An optimized radix-4 solution would actually involve the fewest multiplications, but let's go with the radix-2 solution for familiarity: By analogy with slide 17 of the FFT lecture (topic 10), we can combine each pair of 9-point DFT outputs into 18-point DFTs with just 8 twiddle factors applied to the second half of each pair (16 complex multiplies total), and the final combination of the two 18-point DFTs will require 17 twiddle factors, again applied once to the second bank. From the figure above, we can see that a 9 point DFT requires 12+16=28 complex multiplies, so the total number of multiplies = 4*28 + 16 + 17 = 145. (The 36-point radix-4 solution would require only 24 multiplies for a total of 136).

Comments: It was common to be overwhelmed the complexity of drawing the flow diagram, and particularly inserting all the correct twiddle factors, although several people managed it. No-one did a complete analysis of the number of multiplies required, although several people used equation 8.45 given in Mitra, which predicts 6 multiplies per point, or 216 multiplies for a radix-2 solution (it doesn't account for skipping the W_N^0 terms, or for the additional savings from factoring out the twiddles in the radix-2/radix-4 stages).

5. This was an open-ended question, graded according to how much breadth of thinking you showed and the likelihood of success of the solutions proposed. I had in mind a solution based on the short-time Fourier transform (i.e. a spectrogram display), although the details of how to pick out a 'salient' energy concentration from such a representation are beyond the scope of this course. One issue I was hoping to bring out concerned the tradeoff between time and frequency resolution: In order to discern the short duration of the snapping turtle sounds, we want to use an STFT window length of the order of 10 ms, but this will limit our frequency resolution to about 100 Hz, which is unlikely to allow us to distinguish between the slightly-differing frequencies generated by the seals. One good suggestion is to use two STFTs in parallel, one with a short window to pick out turtle events, and one with a longer window to allow a finer frequency measurement for the seals and dolphins. With luck, this would also resolve the problems arising from the spectral overlap between dolphins and turtles, since 'smearing' the short-duration turtle events over a 100 or 200 ms window looking for dolphin sounds will likely reduce their energy considerably.