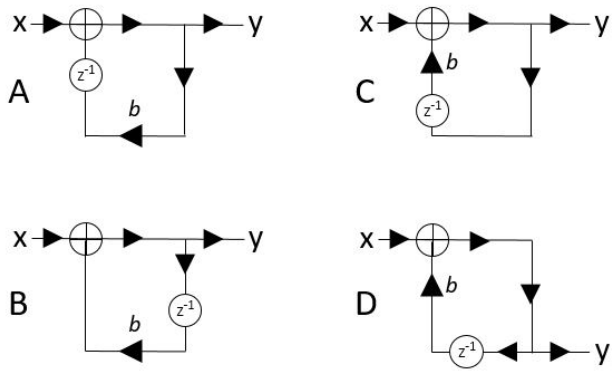
Third set of exercises (graphs and FFT)

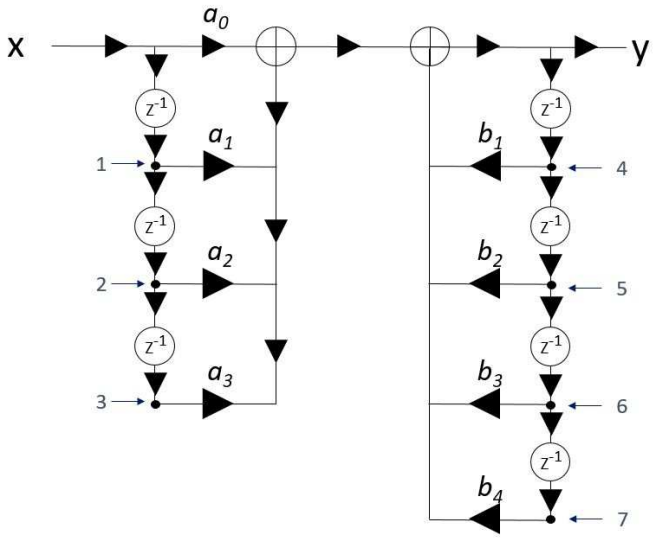
1. Draw the basic AR filter $y_n = x_n + by_{n-1}$ in 4 different ways, analogously to the four basic MA blocks. Which transformations are required to convert the first block into the others?

   ANSWERS

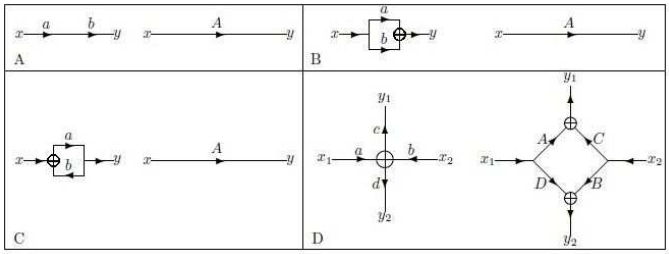


2. Draw an ARMA filter with MA order 3 and AR order 4 in the naïve way. Mark all points requiring memory. How many memory locations are required?

   ANSWER



3. The following examples demonstrate simplification of diagrams with gains. In all cases identify the gain(s) that appear in the right diagram in terms of those that appear in the left diagram.



ANSWERS

A: $\mathbf{A = a * b}$

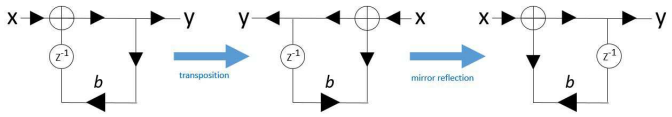
B: $\mathbf{A = a + b}$

C: $\mathbf{y = a(x + by) = ax + aby}$ **so** $\mathbf{(1 - ab)y = ax}$ **and** $\mathbf{y = \frac{a}{1-ab}x}$ **so** $\mathbf{A = \frac{a}{1-ab}}$

D: **on the left** $\mathbf{y_1 = acx_1 + bcx_2}$ **and** $\mathbf{y_2 = adx_1 + bdx_2}$ **on the right** $\mathbf{y_1 = Ax_1 + Cx_2}$ **and** $\mathbf{y_2 = Dx_1 + Bx_2}$, **so** $\mathbf{A = ac}$, $\mathbf{B = bd}$, $\mathbf{C = bc}$, **and** $\mathbf{D = ad}$.

4. The *transposition theorem* states that reversing all the arc directions, changing adders to tee connections and vice-versa, and interchanging the input and output, does not alter the system's transfer function. Prove this theorem for the simple case of $y_n = x_n + by_{n-1}$.

   PROOF



5. In a straightforward implementation a single complex multiplication is performed as 4 real multiplications and 2 real additions $(a + ib)(c + id) = (ac - bd) + i(bc + ad)$. Show how to perform it using 3 real multiplications. **ANSWER** $\mathbf{(a + ib)(c + id) = a(c + d) - d(a + b) + i\left(a(c + d) + c(b - a)\right)}$ How many real additions are now required? **5**

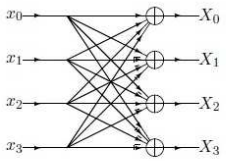
6. In a previous exercise you found explicit equations for DFT with $N = 4$. Draw a naïve graph for this case. How many complex multiplications are required? How many real multiplications are *truly* required? Define temporary variables that are used more than once in the above equations. How much can you save? How much memory do you need to set aside? (Hint: Compare the equations for X_0 and X_2.)

   ANSWER (from DSPCSP page 549)

The four-point DFT

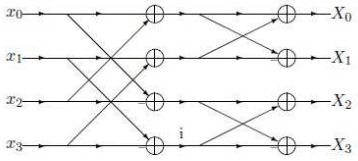
$$\begin{aligned} X_0 &= x_0 + x_1 + x_2 + x_3 \\ X_1 &= x_0 - ix_1 - x_2 + ix_3 \\ X_2 &= x_0 - x_1 + x_2 - x_3 \\ X_3 &= x_0 + ix_1 - x_2 - ix_3 \end{aligned}$$

graphically is



where we have employed two ad-hoc short-hand notations, namely that a line above an arrow means multiplication by -1, while a line before an arrow means multiplication by i. We see that the four-point DFT requires 12 complex additions but no true multiplications. In fact only the radix-2 and radix-4 butterflies are completely multiplication free, and hence the popularity of these radixes.

four-point DFT via radix-2 DIT



We are surprised to see that only eight complex additions and no multiplications are needed. Thus it is more efficient to compute a four-point DFT using radix-2 butterflies than radix-4!

7. How many complex multiplications are needed for the naïve DFT with $N = 16$. How many are *truly* needed (i.e., are not identically 1)? **One would expect $\mathbf{16^2 = 256}$ since we need to find 16 frequency domain vectors of length 16, each by multiplying a 16*16 matrix times the length-16 time domain vector. However, the first row and the first column are identically 1, so the first matrix row requires no true multiplications, and the other 15 matrix rows only require 15 multiplications each, thus we are left with $\mathbf{15 * 15 = 225}$ nontrivial complex multiplications.** How many complex additions are needed? **Each matrix row times the time domain vector requires 15 complex additions, so 16 frequency components require $\mathbf{16 * 15 = 240}$ additions.** How many complex multiplications are needed for the DIT FFT with $N = 16$? How many are *truly* needed (i.e., are not identically 1). **There are 4 layers of butterflies, each requiring 16/2=8 multiplications, so we expect 4*8=32 complex multiplications. However, none of the layers really requires 8 multiplications. The leftmost layer of butterflies really needs 7 multiplications (since $\mathbf{W_{16}^0 = 1}$); the layer of 2 N=8 butterflies only really requires 2*3=6 multiplications, and layer of 4 N=4 butterflies only really requires 4*1=4 multiplications, and the leftmost layer are N=2 butterflies which require no multiplications at all. So, the FFT really only needs 7+6+4=17 complex multiplications.** How many real multiplications are needed? **Assuming 4 real multiplications for each complex one, 17 complex multiplications translates to 68. Using 3 real multiplications this is reduced to 51. Blindly performing the DFT would have involved $\mathbf{4 * 16^2 = 1024}$ real multiplications. So we have reduced this by a factor of $\mathbf{1024/51 \approx 20}$!**

8. A simple summation MA filter $y_n = x_{n-L+1} + x_{n-L+2} + \ldots + x_{n-1} + x_n$ can be updated with only 2 additions, $y_{n+1} = y_n - x_{n-L+1} + x_{n+1}$. Show that we can similarly update a more general MA filter, as long as the coefficients grow geometrically $a_l = q^l a_0$. What operations are required? $\mathbf{y_{n+1} = x_{n+1} + q(y_n - q^{L-1}x_{n-(L-1)})}$. **For example, if $\mathbf{y_n = x_n + 2x_{n-1} + 4x_{n-2} + 8x_{n-3}}$ (i.e., q=2 and L=4) then $\mathbf{y_{n+1} = x_{n+1} + 2x_n + 4x_{n-1} + 8x_{n-2} = x_{n+1} + 2(y_n - 8x_{n-3})}$.** What algorithm does this imply for the DFT? **Note that each row of the DFT matrix contains powers of the element in the second column; e.g., the second row is $\mathbf{(W_N^0, W_N, W_N^2, W_N^3, \ldots W_N^{N-1})}$ which has $\mathbf{q = W_N}$; the next row is $\mathbf{(W_N^0, W_N^2, W_N^4, W_N^6, \ldots W_N^{2(N-1)})}$ which has $\mathbf{q = W_N^2}$; etc. So, we can update a DFT by one point at a time using what we can call the *FIFO-FFT* (see section 14.9 in DSPCSP). We need to somehow initialize the algorithm (e.g., by using the FFT or even the straightforward DFT) to find the DFT of $\mathbf{(x_0, x_1, \ldots x_{N-1})}$ and then use the FIFO-FFT to update this to become the DFT of $\mathbf{(x_1, x_2, \ldots x_N)}$.** Is this algorithm ever better than the FFT? **It turns out that if we need to calculate many DFTs, and there is major overlap between consecutive ones (say 75 percent or more) the FIFO-FFT is cheaper than calculating the FFT each time!**