Carnegie Mellon DSP Electrical & Computer ENGINEERING

Digital Signal Processing (18-491/18-691)
Spring Semester, 2024

Problem Set 6

Issued: 3/14/24

Due: 3/21/24 at midnight via Gradescope

Reading: In the past week we introduced the basic decimation-in-time FFT algorithm, following OSYP 9.2, along with alternative structures, and non-radix-2 FFTs. The relevant sections of OSYP are Secs. 8.7, 9.0, and 9.2 to 9.5. We have also distributed some additional notes from the original Oppenheim and Schafer published in 1975 which describe the computation of non-radix-2 FFTs, as well of the Powerpoint presentations that were used for the discussions concerning the alternate FFT structures.

Next week we will discuss the implementation procedures for discrete-time systems, following OSYP Secs. 6.0 through 6.5.

Reminder: Quiz 2 will be on April 3

Problem 6.1: Problem 9.6 in OSYP.

Problem 6.2: Problem 9.7 in OSYP.

Problem 6.3: Problem 9.42 in OSYP.

Problem 6.4: Problem 9.58 in OSYP.

Problem 6.5: We discussed in class the efficient implementations of the DFT using decimation-in-time principles if the DFT sizes that are not exact powers of 2 but that is highly composite (meaning that it has a large number of factors). Let the DFT size be $N = p_1 p_2 p_3 \ldots p_\nu$ and let $q_1 = N/p_1, q_2 = N/p_1 p_2$, etc.

As usual,

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk} \tag{1}$$

This equation can be rewritten as

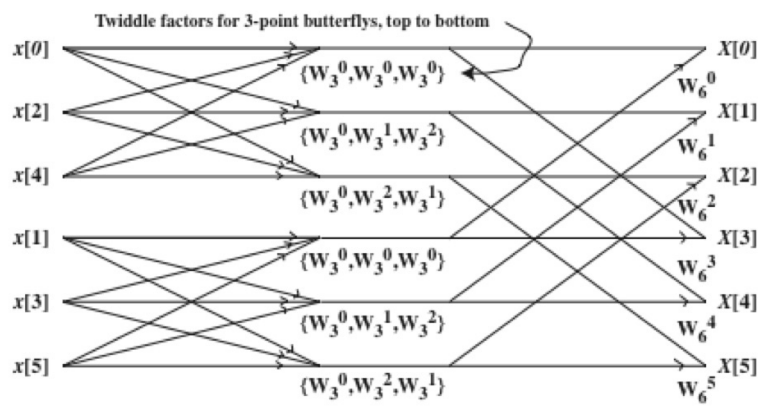
$$X[k] = \sum_{r=0}^{q_1-1} x[p_1 r] W_N^{p_1 rk} + \sum_{r=0}^{q_1-1} x[p_1 r+1] W_N^{(p_1 r+1)k} + \ldots \sum_{r=0}^{q_1-1} x[p_1 r+(q_1-1)] W_N^{(p_1 r+(p_1-1))k} \quad (2)$$

or, more generally,

$$X[k] = \sum_{l=0}^{p_1-1} W_N^{lk} \sum_{r=0}^{q_1-1} x[p_1 r + l] W_{q_1}^{rk} \quad (3)$$

As an example, consider the 6-point decimation-in-time FFT structure with $p_1 = 2$ and $p_2 = 3$. In this case we obtain

$$X[k] = \sum_{l=0}^{1} W_6^{lk} \sum_{r=0}^{2} x[2r + l] W_3^{rk}, \text{ which hase the signal flowgraph} \quad (4)$$



This techniques is discussed in much greater detail in the supplementary material from Oppenheim and Schafer (1975) that is posted on the class website.

* * * * * * * * *

In this problem we consider the radix-4 FFT, which is a popular alternative to the conventional radix-2 FFT when N is an integer power of 4.

(a) First, let us consider a the signal flowgraph for a conventional 16-point FFT.

1. Draw a signal flow graph for a 16-point decimation-in-time FFT algorithm using conventional techniques as discussed in class. Factor the twiddle factors (*i.e.* the W_N^k terms) outside the butterflys, so that the computation is reduced by a factor of 2, also as discussed in class.

2. How many complex multiplys are there if we count all coefficients W_N^k as one complex multiply?

3. How many complex multiplys are there if we do not count those twiddle factors that are actually equal to ± 1 or $\pm j$?

(b) Now draw the flow graph for a simple 4-point FFT realized without factoring (*i.e.* in a single column). Note that all of the twiddle factors of the 4-point DFT are equal to ± 1 or $\pm j$.

(c) Now we will consider the radix-4 16-point FFT. First, let us consider a the signal flowgraph for a conventional 16-point FFT.

1. Draw a signal flow graph for the 16-point decimation-in-time FFT algorithm using two columns of 4-point butterflys, as you drew in part (b). In other words, this would be a non-radix-2 FFT with $p1 = p2 = 4$. Again, factor the twiddle factors outside the butterflys, so that the computation is reduced by a factor of 2, also as discussed in class.

2. How many complex multiplys in the radix-4 implementation are there if we count all coefficients W_N^k as one complex multiply?

3. How many complex multiplys in the radix-4 implementation are there if we do not count those twiddle factors that are actually equal to ± 1 or $\pm j$? Compare these counts to the corresponding number of multiplys for the radix-2 implementation of the FFT.

(d) Draw the signal flow-graph of the decimation-in-frequency version of the simple 4-point FFT, which was your answer to part (b). As you know, this is most easily accomplished by considering the decimation-in-frequency FFT as the transpose of the corresponding decimation-in-time FFT.

MATLAB Problems

For the MATLAB questions this week and in the future, please submit the following components of your answer to the **written** component of your homework submission on Gradescope. An easy way to handle the code part of this is to use the `publish` feature in MATLAB and submit the output .pdf to the Written assignment on gradescope.

- Answers to the written portions of the problems
- Your plots
- A pdf copy of your code

The MATLAB component of your submission should contain only your `.m` files. We appreciate your help in complying with these formatting requests as it makes your work much easier to grade. As before, we will deduct points for noncompliant submissions.

Problem C6.1:

In this problem you will implement in MATLAB the 16-point FFT algorithm that you developed in Problem 6.5.

(a) Please complete the subroutine `fft16.m` that implements this algorithm. It should run without error in the main file `main_7_1.m` that we provide. Note that we provide a specific input for the program that we will use to test the results.

(b) Using your favorite MATLAB profiling utility, compare the run time of the following. If the time is too brief to be measured accurately, call the subroutine 100 times in succession. We are more concerned with the ratios of the various run times than with their individual values (which will depend on the processor speed and other hardware and operating system issues in all cases).

- The execution time for your program `fft16.m`
- The execution time for the built-in MATLAB function `fft`

What to turn in in written form to Gradescope:

- A signal flowgraph for your 16-point FFT algorithm.
- A .pdf file with the source code for your completed function `fft16.m`
- Your answers to part (b) in ratio form relative to the execution time of the standard MATLAB function `fft`.
- A printout of the output of the main program `main_7_1.m` that compares the numerical output of your routine to that of the standard MATLAB function `fft`.

What to turn in in code form to Gradescope:

- An executable version of your subroutine `fft16.m`