ELEC6410 Project 4

Answers to Digital Signal Processing Project #4

Michael J. Carroll

October 8, 2010

Question 1

Write a MATLAB function that implements a DFT using the DFT summation formula.

```
function [ X ] = dftmjc( x )
%dftmjc Computes the DFT summation of an input sequence.
%   Michael's (probably bad) implementation of the DFT.

% Verify that we are getting a proper 1-D sequence in.
validateattributes(x,{'numeric'},{'vector'});
% Preallocate the array for speed.
N = length(x);
X = zeros(N);
% Summation is O(n^2) (bad news)
for k=1:N,
   for n=1:N,
       X(k,n)=x(k)*exp(((-2*pi*1i)/N) * (k-1) * (n-1));
   end
end
% Let MATLAB do the actual sum, it probably vectorizes this.
X = sum(X);
end
```

Question 2

Compare the time required for a DFT matrix multiply and the function written in question 1. To accomplish this, I ran each implementation 10 times to get an average value for the tic/toc output. The code I used is included below. The results are discussed in Question 3.

```
F = dftmtx(2^11);
mjctimes = zeros([1,10]);mattimes = zeros([1,10]);
```

```
x = rand([1,2^11]);
for n = 1:10,
    tic;Xf = dftmjc(x);mjctimes(n) = toc;
    tic;Xf = F * x';mattimes(n) = toc;
end
mean(mjctimes)
mean(mattimes)
```

Question 3

I repeated the same exercise with MATLAB's FFT function.

```
x = rand([1,2^11]);
ffttimes = zeros([1,10000]);
for n=1:10000,
    tic;Xf = fft(x);ffttimes(n) = toc;
end
mean(ffttimes)
```

DFT Summation	DFT Matrix Multiply	MATLAB FFT
2.778 seconds	0.768 seconds	35.64 microseconds

Table 1: Average execution time of three DFT implementations

As can be seen in Table 1, MATLAB's FFT is significantly faster than the DFT matrix multiplication and the DFT summation function that I wrote. This stands to reason, as the FFT algorithm itself is much faster than either of the others. The summation and the matrix multiply should be close in time complexity (on order of $O(n^2)$), while the FFT should have a much lower time complexity (on the order of $O(n\ log(n))$).

In addition to the algorithmic speedup, the matrix multiplication and the FFT should also have increased speedup from MATLAB's implementation. MATLAB relies heavily on the FFTW library, which uses various additional instruction sets to gain additional speed from hardware. Matrix multiplication in MATLAB probably also utilizes these low-level speedups.

Question 4

I repeated the same exercise as Question 3, except this time the sequence was length $2^{11} - 1$. Including the code is not necessary, as it is identical to Question 3's implementation. The major factor playing into execution time for this sequence is that it is prime-length. This means that the typical FFT algorithm cannot be used, and a different (Rader) algorithm, with a higher time complexity, is used. The average execution time for this sequence was 357.9 microseconds, or roughly an order of magnitude slower.

Question 5

My implementation of a MATLAB function that implements linear convolution of two input sequences using FFTs is included below. The results are in Table 2.

```
function [ y ] = linearConv( x1,x2 )
%linearConv Performs linear convolution using power-of-2 FFT's
N1 = length(x1);
N2 = length(x2);
% Zero pad
X1 = fft(x1,N1 + N2 - 1);
X2 = fft(x2,N1 + N2 - 1);
Y = X1 .* X2;
y = ifft(Y);
end
```

MATLAB's Convolution	FFT Convolution
0.0155 seconds	0.0045 seconds

Table 2: Average execution time of two convolution implementations

The FFT convolution is about three times faster than the bulit-in MATLAB conv function. This is probably because MATLAB's implementation is a more general case, and cannot take advantage of the FFT implementation's speedup.

Question 6

I implemented a MATLAB function that implements a non-power-of-two FFT using power-of-two FFT's. My function is included below.

The results of the speed comparison are included in Table 3. MATLAB's FFT is still faster than my convolution function, but is still considerably slower than the power-of-two FFT. I would assume that the two are probably on the same order of time complexity, but my implementation is probably worse than the FFTW implementation that MATLAB is using. Once again, MATLAB also has a speed advantage by using additional hardware.

My implementation uses the Bluestein algorithm, also known as the Chirp-z algorithm. I opted to use FFTs to perform the convolution.

```
function [ X ] = nonpowfft( x )
% References:
% http://www.engineeringproductivitytools.com/stuff/T0001/PT11.HTM#Head617
% http://www.katjaas.nl/chirpZ/chirpZ.html
% http://en.wikipedia.org/wiki/Bluestein%27s_FFT_algorithm
% http://healpix.jpl.nasa.gov/html/libfftpack/bluestein_8c-source.html
% I'm using the notation for Bluestein's algorithm from wikipedia.
validateattributes(x,{'numeric'},{'vector'});
N=length(x);

% We want the convolution index to be > 2N-1 (from reference 1).
Nf = 2^nextpow2(2*N-1);

% Coefficient a_{n} and b_{n} from Bluestein's algorithm.
n=1:N;
a(n)=x(n).*exp(-1i*pi*(n-1).^2/N);
b(n)=exp(1i*pi*(n-1).^2/N);
b(Nf-(n-2))=b(n);

% The loop generates a b_{n} 1 longer than it's supposed to.
% I started late, and I'm too tired to make this clever.
b=b(1:Nf);

% Convolve a_{n} and b_{n} together.
% Using Nf as an argument for the fft guarantees a power-of-2 input.
% We could just as easily zero-pad and convolve.
y = ifft(fft(a,Nf).*fft(b,Nf),Nf);

% Multiply by coefficient b_{k} from Bluestein's algorithm.
X(n) = y(n).*exp(-1i*pi*(n-1).^2/N);
end
```

MATLAB's FFT	FFT by Convolution
0.617 seconds	2.149 seconds

Table 3: Average execution time of two FFT implementations

The complete source to this project, including TeX, is available on my Github account at http://github.com/mjcarroll/elec6410