Lab 8 – DFT and FFT

8.1 Direct computation of the DFT

The DFT of a length-N column vector can be directly computed by multiplying it with an N x N DFT matrix. What are the entries of an N x N DFT matrix?

For this problem, write a single input/output function `directdft` that computes the DFT of an input (column) vector x and returns an output vector X.

Your function should implement the DFT equations directly, without using any shortcuts, or using the MATLAB `fft` command. Take advantage of MATLAB's vector and matrix-based syntax, instead of using nested `for` loops. The result of your algorithm should agree with MATLAB's `fft` command. Use the template below.

```
function X = directdft(x)
% x should be a length-N column vector
N = length(x);
% create N x N DFT matrix
F = ...
% compute w using a matrix-vector product
X = F*x;
end
```

Test your function for various signals including the discrete-time rectangular periodic pulse with L (< N) ones and and (N-L) zeros, i.e. `x = [ones(L,1),zeros(N-L,1)]`. In the same figure (using subplot) plot x and abs(X) as N is increased for fixed L.

8.2 The radix-2 FFT

Write a single input/output function `radix2fft` accepts a column vector x and returns an output vector X. To compute X, implement the decimation-in-time radix 2 FFT algorithm for an input vector whose length is a power of 2. Your function must have the recursive structure given in the solution template. That is, `radix2fft` is called within itself, until it is called using a signal of length N = 2. What is the DFT when N = 2?

Verify that the output of `radix2fft` matches with that of `directdft` above and MATLAB's `fft` command. You can use the template given below.

```
function X = radix2fft(x);
% x should be a length-N column vector
N = length(x);
if ~isequal(unique(factor(N)),2)
    error('N is not a power of 2!');
end
if N == 2
    % Compute length-2 DFT directly (it's super simple)
    X = ...
else
    % Split input vector into even and odd parts
    x_even = ...
    x_odd = ...

    % Take radix 2 FFT of each part
    X_even = ...
    X_odd = ...

    % Recombine length N/2 results into length N result
    X = ...
end
end
```

8.3 tic toc tic toc

In this experiment we will compare the time taken (empirically) to compute the DFT using the three implementations – `directdft`, `radix2fft`, and inbuilt `fft` command. This we will do by using the `tic` and `toc` MATLAB routines. As the length of the signal (i.e. N) is varied (in powers of 2 from 2^1 to 2^{10}) generate a random signal and compute its DFT (repeat this M=10 times for each N) using each of the three methods and record the time taken for each of them using the `tic` and `toc` routines. Plot the time taken as function of N for each of the methods (in the same plot). Note that these routines provide only an approximate estimate of computational time and may not be very precise.

Repeat the timing analysis above for the implementations `directdft` and `fft` when N is not restricted to powers of 2.

8.4 The DFT and convolution

As mentioned in the class, we can use the DFT to compute time-domain convolutions. Though we must be careful to distinguish between linear convolution (which is usually what we want) and circular convolution (which is involved in DFT properties).

Write a function `dftconv` that takes the inputs

- h, the first signal (e.g., a filter impulse response)
- x, the second signal
- N, an integer where N is greater than or equal to max(length(h), length(x))

The function should return

- conv_circ, the circular convolution of h and x using N points (using DFTs of length N)
- conv_lin, the linear convolution of h and x (using appropriately-sized DFTs)

The outputs of your function should match those of the commands `cconv` and `conv`, respectively (read up MATLAB documentation of these commands). You can compare the outputs by plotting results of `dftconv` and `cconv/conv` side by side in the same figure.

```
function [conv_circ, conv_lin] = dftconv(h,x,N);
lx = length(x);
lh = length(h);
if N<max([lx lh])
    error('N must be at least the length of the longer signal.')
end
% Compute order-N circular convolution based on length-N FFTs
conv_circ = ifft(...);
% Compute linear convolution based on appropriate-length
FFTs/zero padding
M = ...;
conv_lin = ifft(...);
end
```

(a) What is the computational complexity of evaluating the convolution as implemented above using FFT based methods?

(b) How does it compare with the computational complexity of directly evaluating the convolution in time domain?