Digital Signal Processing
Lab 3: Discrete Fourier Transform

Discrete Time Fourier Transform (DTFT)

The discrete-time Fourier transform (DTFT) of a sequence x[n] is given by

$$X(e^{i\omega}) = \sum_{n=-\infty}^{\infty} x[n]\, e^{-j\omega n} \qquad (3.1)$$

which is a continuous function of ω, with period 2π.

The inverse discrete-time Fourier transform (IDTFT) of $X(e^{j\omega})$ is given by

$$x[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(e^{i\omega}) e^{j\omega n} d\omega \qquad (3.2)$$

Important observation. Matlab cannot be used to perform directly a DTFT, as $X(e^{j\omega})$ is a continuous function of the variable ω. However, if x[n] is of finite duration, eq. (3.1) can be applied to evaluate numerically $X(e^{j\omega})$ at any desired frequency $\omega = \omega_0$.

Given a real-valued sequence x[n], considering the periodicity and the symmetries of its discrete-time Fourier transform, to represent graphically $X(e^{j\omega})$ we need to consider only a half period of $X(e^{j\omega})$. Generally, in practice ω is chosen to be in the interval [0, π].

If x[n] is of infinite duration, then eq. (3.1) cannot be used to evaluate $X(e^{j\omega})$ from x[n] with Matlab. However, we can derive the analytic expression of $X(e^{j\omega})$ and then plot its magnitude and angle (or real and imaginary parts) over a set of frequencies in [0, π].

Example 1: Determine the frequency response $H(e^{j\omega})$ of a system characterized by the impulse response $h[n] = (0.9)^n u[n]$. Plot the magnitude and the phase responses.

$$H(e^{j\omega}) = \sum_{-\infty}^{\infty} h(n)e^{-j\omega n} = \sum_{0}^{\infty} (0.9)^n e^{-j\omega n} = \frac{1}{1 - 0.9e^{-j\omega}}$$

We evaluate $H(e^{j\omega})$ on 501 points uniformly distributed in the range [0, π]

```
w = [0:1:500]*pi/500; % 501 points on the [0, pi] axis
H = 1 ./ (1 - 0.9*exp(-j*w));
magH = abs(H); angH = angle(H);

subplot(2,1,1); plot(w/pi,magH); grid;
xlabel('\omega/\pi'); ylabel('|H|');
title('Magnitude Response');
```

```
subplot(2,1,2); plot(w/pi,angH/pi); grid
xlabel('\omega/\pi'); ylabel('Phase in radians/\pi');
title('Phase Response');
```

Discrete Fourier Transform (DFT)

The Discrete Fourier Transform of an N -point sequence x[n] is given by

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn} \qquad 0 \le k \le N-1 \qquad (3.3)$$

where $W_N = e^{-j\frac{2\pi}{N}}$

The Inverse Discrete Fourier Transform of an N -point DFT X[k] is given by

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] W_N^{-kn} \qquad 0 \le n \le N-1 \qquad (3.4)$$

If x[n] and X[k] are arranged as column vectors **x** and **X** we can write the DFT and IDFT as the matrix products $\mathbf{X}=\mathbf{D}_N\mathbf{x}$ and $\mathbf{x}=\mathbf{D}^*_N\mathbf{X}$ where we have used the square $\mathbf{D}_N$ matrix defined as

$$\mathbf{D}_N \triangleq \left[W_N^{kn}{}_{0\le k,n\le N-1}\right] = k \downarrow \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & W_N^1 & \cdots & W_N^{(N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{(N-1)} & \cdots & W_N^{(N-1)^2} \end{bmatrix} \quad n \longrightarrow$$

Note that this matrix can be generated directly in Matlab by the function `dftmtx(N)`.

A direct way to implement the computation of the matrix $\mathbf{D}_N$ is the following:

```
n = [0:1:N-1];
k = [0:1:N-1];

DN = exp(-j*2*pi/N *n'*k);          % DN is a N by N DFT matrix
```

The following functions are a direct implementation of DFT and inverse DFT according to equations (3.3) and (3.4). They are inefficient for large values of N.

```
function [Xk] = dft(xn,N)
% Computes Discrete Fourier Transform
% ----------------------------------
% [Xk] = dft(xn,N)
% Xk = DFT coeff. array over 0 <= k <= N-1
% xn = N-point finite-duration sequence
% N = Length of DFT
%
n = [0:1:N-1];
k = [0:1:N-1];
xn=xn(:);        % make sure xn is a column vector

WN = exp(-j*2*pi/N *n'*k); % creates a N by N DFT matrix
Xk = WN * xn;

Xk=Xk.';         % make it a row vector of DFT coefficients
```

```
function [xn] = idft(Xk,N)
% Computes Inverse Discrete Fourier Transform
% ----------------------------------
% [xn] = idft(Xk,N)
% xn = N-point sequence over 0 <= n <= N-1
% Xk = DFT coeff. array over 0 <= k <= N-1
% N = length of DFT
%
n = [0:1:N-1];
k = [0:1:N-1];
Xk=Xk(:);        % make sure Xk is a column vector

WN = exp(j*2*pi/N *n'*k); % creates a N by N IDFT matrix
xn = (WN * Xk)/N;

xn = xn.';      % make it a row vector
```

Warning: if `c` is a complex vector, the vector `c'` is its transpose-conjugate. If you just want to transform from row to column (or vice versa) use the operation `c.'`

Example 2. Given the sequence x[n] equal to 1 for $0 \le n \le N$ and equal to 0 elsewhere, compute its DFT on N points and compare the magnitude of its DFT coefficients with the magnitude of its DTFT.
The DTFT is given by

$$X(e^{j\omega}) = \sum_{0}^{N-1} x[n]e^{-j\omega n} = \frac{1-e^{-j\omega N}}{1-e^{-j\omega}} = \frac{sin(\omega N/2)}{sin(\omega/2)} e^{j(N-1)\omega/2}$$

and its magnitude is $\left|X(e^{j\omega})\right| = \left|\frac{sin(\omega N/2)}{sin(\omega/2)}\right|$

```
N=11;
w=0:1/1000:2*pi;
magX=abs(sin(N*w/2)./sin(w/2));
plot(w/pi,magX); grid

x=ones(1,N);
X=dft(x,N);
hold on;
stem(2*[0:N-1]/N,abs(X),'r');  % N equispaced samples in [0:pi]
```

What happens if the sequences x[n] is padded with N zeros?

```
x2=[ones(1,N) zeros(1,N)];
X2=dft(x2,2*N);
stem(2*[0:2*N-1]/(2*N),abs(X2),'g');  % 2*N equispaced samples
```

And what happens when the sequences x[n] is padded with a multiple of N zeros?

```
x6=[ones(1,N) zeros(1,5*N)];
X6=dft(x6,6*N);
stem(2*[0:6*N-1]/(6*N),abs(X6),'m');  % 6*N equispaced samples
hold off
```

Verification of some properties of the DFT

The following examples verify empirically some important properties of the DFT.

Linearity

```
x1 = rand(1,11); x2 = rand(1,11);
alpha = 2; beta = 3;

X1=fft(x1);
X2=fft(x2);
```

```
x = alpha*x1 + beta*x2;         % Linear combination of x1 & x2
X = fft(x);      % DFT of x

% verification
X_check = alpha*X1 + beta*X2; % Linear Combination of X1 & X2
error = max(abs(X-X_check))   % check entity of max error
% within the limited numerical precision of Matlab?
```

Time-shift property

```
x = rand(1,11);
X=fft(x);

% signal shifted by two samples
y=cirshftt(x,2,length(x));
Y=fft(y);

k = 0:length(x)-1;
w = (2*pi/length(x))*k;  % frequency values

% verification
Y_check = exp(-j*2*w).*X;
error = max(abs(Y-Y_check))   % check entity of max error
```

Frequency-shift property

Let us consider $x[n] = 0.9^n$, $0 \le n \le 100$ and $y[n] = e^{j\pi n/4} x[n]$

```
n = 0:100; x=0.9.^n;
X=fft(x);

y = exp(j*pi*n/4).*x;      % signal multiplied by exp(j*pi*n/4)
Y = fft(y);
k = 0:length(x)-1;
w = (2*pi/length(x))*k;  % frequency values between 0 and 2*pi

% Graphical verification
subplot(2,2,1); plot(w/pi,abs(X)); grid; axis([0,2,0,12])
xlabel('\omega/\pi'); ylabel('|X|');
title('Magnitude of X');
subplot(2,2,2); plot(w/pi,angle(X)/pi); grid; axis([0,2,-0.5,0.5])
xlabel('\omega/\pi'); ylabel('radians/\pi');
title('Phase of X');
subplot(2,2,3); plot(w/pi,abs(Y)); grid; axis([0,2,0,12])
xlabel('\omega/\pi'); ylabel('|Y|');
```

```
title('Magnitude of Y');
subplot(2,2,4); plot(w/pi,angle(Y)/pi); grid; axis([0,2,-0.5,0.5])
xlabel('\omega/\pi'); ylabel('radians/\pi');
title('Phase of Y');
```

Circular shift of a sequence

The following function implements the circular shift of m samples of a sequence x[n] with respect to a buffer of length N. The length of the sequence x[n] must be ≤N.
Note that the modulo-N operation is being used on the argument (n−m) to implement this operation.

```
function y = cirshftt(x,m,N)
% Circular shift of m samples wrt size N
% in sequence x
% ------------------------------------------------------------
% [y] = cirshftt(x,m,N)
% y = output sequence containing the circular shift
% x = input sequence of length <= N
% m = shift in samples
% N = size of circular buffer

if length(x) > N
 error('N must be >= length of x')
end
x = [x zeros(1,N-length(x))];
nx = [0:N-1];
ny = mod(nx-m,N);  % apply shift to sample indexes
y = x(ny+1);       % indexes of Matlab vectors must start from 1
```

Circular convolution

When we multiply two N-point DFTs in the frequency domain, we get correspondingly the **circular convolution** (and not the usual linear convolution) of the original sequences in the time domain.

$$x_3[n] = \sum_{m=0}^{N-1} x_1[m]\, x_2[\langle n-m \rangle_N] \quad \leftrightarrow \quad X_3[k] = X_1[k] \cdot X_2[k]$$

The following function implements the circular convolution of two sequences directly in the time-domain.

```
function y = circonvt(x1,x2,N)
% N-point circular convolution between x1 and x2: (time-domain)
% ------------------------------------------------------------
% [y] = circonvt(x1,x2,N)
```

```
% y = output sequence containing the circular convolution
% x1 = input sequence of length N1 <= N
% x2 = input sequence of length N2 <= N
% N = size of circular buffer
% Method: y(n) = sum (x1(m)*x2((n-m) mod N))

if length(x1) > N
error('N must be >= the length of x1')
end
if length(x2) > N
error('N must be >= the length of x2')
end

x1=[x1 zeros(1,N-length(x1))];
x2=[x2 zeros(1,N-length(x2))];
m = [0:1:N-1];
x2n = x2(mod(-m,N)+1);   % obtain the sequence x2n[n]=x2[-n]

for n = 0:N-1
  y(n+1)=x1 * cirshftt(x2n,n,N).';
end
```

Important observation: Given the sequences $x_1[n]$ of length N_1 and $x_2[n]$ of length N_2, if we make both $x_1[n]$ and $x_2[n]$ sequences of length $N=N_1+N_2-1$ by padding an appropriate number of zeros, then the circular convolution is identical to the linear convolution.

Example 3:

```
>> conv([1 2 3], [-1 1 1 -1])
ans =    -1    -1     0     4     1    -3
>> circonvt([1 2 3], [-1 1 1 -1],6)
ans =    -1    -1     0     4     1    -3
```

Exercise. Verify that in the previous example N=6 is the minimum length of the circular convolution in order to obtain the same result as the linear convolution. What happens when we choose N>6?

Note that Matlab provides a specific functions **`circshift`** to shift array circularly and **`cconv`** to perform modulo-N circular convolution. Moreover, the implementations of DTFT and DFT presented above have only didactical purpose. Matlab offers much more efficient functions.

Example 4:

```
a = [1 2 -1 1];
b = [1 1 2 1 2 2 1 1];

c = cconv(a,b);            % Circular convolution
cref = conv(a,b);          % Linear convolution
```

Fast Fourier Transform (FFT)

Matlab provides an efficient `fft` function to compute the DFT of a vector `x`. It is invoked by

`X = fft(x,N)`

which computes the N-point DFT. If the length of `x` is less than N, then `x` is padded with zeros. If the argument N is omitted, then the length of the DFT is the length of `x`. If `x` is a matrix, then `fft(x,N)` computes the N-point DFT of each column of `x`.

The implementation is very efficient as it is written in machine language (i.e., it is not available as a .m file) and therefore it executes very fast.

If N is a power of two, then a high-speed radix-2 FFT algorithm is employed. If N is not a power of two, then N is decomposed into prime factors and a slower mixed-radix FFT algorithm is used. Finally, if N is a prime number, then the `fft` function is reduced to the raw DFT algorithm.

The inverse DFT is computed using the `ifft` function, which has the same characteristics as `fft`.

Note that `fft(x)` assumes that the first sample of `x` is at time instant n=0.

Exercise. Compare the results of Example 2, where the function `dft` was used, with those produced when using the function `fft.`

Example 5. Plot magnitude, phase, real part and imaginary part of the FFT of the sequence $x[n] = (0.9)^n$, $-10 \le n \le 10$.

As the sequence is real, it is enough to limit the plot in the frequency interval $[0, \pi]$.

```
nx=[-10:10];
x=(-0.9).^nx;

M=512;  %length of the fft to have a good frequency resolution

% fft assumes that the first sample of x is at time instant 0
% therefore we need a suitable circular shift
x=cirshftt(x,-10,M);

X=fft(x,M);
```

```
X=X(1:M/2);     % take only the interval between 0 and pi
w=[0:(M/2-1)]/(M/2);

magX = abs(X); angX = angle(X);
realX = real(X); imagX = imag(X);

subplot(2,2,1); plot(w,magX); grid
xlabel('\omega/\pi'); title('Magnitude of X');
subplot(2,2,2); plot(w,angX/pi); grid
xlabel('\omega/\pi'); ylabel('Phase of in radians/\pi');
title('Phase of X')
subplot(2,2,3); plot(w,realX); grid
xlabel('\omega/\pi'); title('Real Part of X');
subplot(2,2,4); plot(w,imagX); grid
xlabel('\omega/\pi'); title('Imaginary Part of X');
```

Example 6. Compute the DFT of the sequence x[n] = {1, 2, 3, 4, 5} on 512 points and plot magnitude, phase, real part and imaginary part.

```
n = -1:3; x = 1:5;
M=512;

x=cirshftt(x,-1,M);

X=fft(x,M);

k = 0:M-1;
w = (2*pi/M)*k;  % frequency values between 0 and 2*pi

magX = abs(X); angX = angle(X);
realX = real(X); imagX = imag(X);

subplot(2,2,1); plot(w/pi,magX); grid
xlabel('\omega/\pi'); title('Magnitude');
subplot(2,2,2); plot(w/pi,angX/pi); grid
xlabel('\omega/\pi'); ylabel('Phase in radians/\pi'); title('Phase')
subplot(2,2,3); plot(w/pi,realX); grid
xlabel('\omega/\pi'); title('Real Part');
subplot(2,2,4); plot(w/pi,imagX); grid
xlabel('\omega/\pi'); title('Imaginary Part');
```

Compare the obtained plots with the graphical representation in the complex plane produced by

```
figure; polar(angX,magX,'-o')
```

Exercise: Repeat the same analysis for the sequences $x_1[n] = \{\underline{1}, 1, 1, 1, 1\}$ and $x_2[n] = \{1, 1, \underline{1}, 1, 1\}$. Compare and comment on the results.
Hint: in the case of $x_2[n]$, something strange seems to happen: the non-infinite precision of computations has its relevance.

Based on the properties of the DFT, a convolution can be computed as inverse FFT of the product of the FFTs of two sequences. The following `fastconv` function represents an implementation of this procedure. For efficiency, the length of the FFTs is equal to the next power of 2 with respect to the foreseen length of the output sequence (this value can be also obtained with the Matlab function `nextpow2`). For long input sequences this approach provides a faster convolution.

```
function [y] = fastconv(x,h)
% Fast convolution using FFT
% -----------------------------------------------------------------
% [y] = fastconv(x,h)
% y = output sequence
% x = first input sequence
% h = second input sequence
%
L=length(x)+length(h)-1;            % output length
N = 2^(ceil(log10(L)/log10(2)));    % next power of 2
y = ifft(fft(h,N).*fft(x,N));       % zero padding up to length N
y=y(1:L);
```

Exercise. Compare the results of Example 3 with those produced when using the function `fastconv`.

Example 7: Spectral analysis of a whale call.

The file *bluewhale.au* contains audio data from a Pacific blue whale vocalization recorded by underwater microphones off the coast of California. The file is from the library of animal vocalizations maintained by the Cornell University Bioacoustics Research Program.

Because blue whale calls contain very low frequencies, they are barely audible to humans. Therefore the time scale in the data has been compressed by a factor of 10 to raise the pitch (i.e. frequencies become 10 times higher) and make the call more clearly audible.

```
[x,fs]=auread('bluewhale.au');   % read the audio data from file
plot(x)
sound(x,fs)   % listen the compressed-time signal
```

An A "trill" is followed by a series of three B "moans". The B call is simpler and easier to analyze. After selecting a subsequence corresponding to the first B call it is possible to analyze its frequency content by means of a Discrete Fourier Transform.

```
b= x(2.45e4:3.10e4);
B=fft(b);
f=fs/length(b)*[0:(length(b)-1)];
plot(f/10,abs(B))    % plot with the true frequency scale
```

The B call is composed of a fundamental frequency around 17 Hz and a sequence of harmonics, with the second harmonic emphasized.

A more complete spectral analysis of the full sequence is obtained with a graphical representation (spectrogram) of the so called Short Time Fourier Transform, which is obtained as a sequence of DFTs computed on adjacent short intervals of the signal.
Try for example:

```
winlen=512; step=256; Nfft=1024;
spectrogram(x,winlen,step,Nfft,fs/10,'yaxis')
```

Exercises

1. Plot magnitude, phase, real part and imaginary part of the DFT of the following sequences
   - $x1[n] = (0.7)^{|n|} \cdot (u[n+20]-u[n-21])$
   - $x2[n]=n \cdot (0.9)^n \cdot (u[n]-u[n-21])$

2. Given $x[n]=10 \cdot (0.7)^n$, $0 \leq n \leq 10$, determine $y[n]= x[\langle -n \rangle_{11}]$. Compute the DFTs of x[n] and y[n]. How are they related?

3. Using a fft-based spectral analysis determine graphically which tones are present in the sequence generated by the Matlab instruction `x=cos(pi*n/5)+sin(n);` when `n=[0:300]`, assuming a sampling rate of 1 kHz.