In previous laboratories, we have used the Discrete-Time Fourier Transform (DTFT) extensively for analyzing signals and linear time-invariant systems.

$$\text{(DTFT)} \quad X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n]\, e^{-j\omega n} \tag{4.1}$$

$$\text{(inverse DTFT)} \quad x[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(e^{j\omega})e^{j\omega n} d\omega\,. \tag{4.2}$$

While the DTFT is very useful **analytically**, it usually cannot be exactly evaluated on a computer because (4.1) requires an infinite sum and (4.2) requires the evaluation of an integral.

The discrete Fourier transform (DFT) is a sampled version of the DTFT, hence it is better suited for numerical evaluation on computers.

$$\text{(DFT)} \quad X_N[k] = \sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/N} \tag{4.3}$$

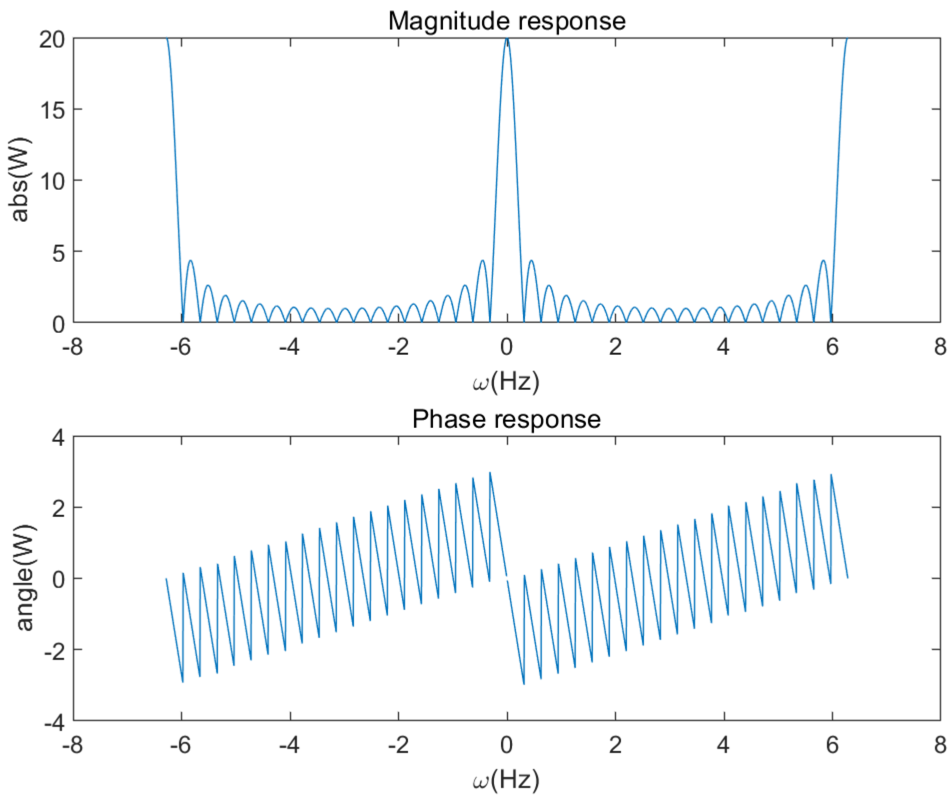
$$\text{(inverse DFT)} \quad x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X_N[k]e^{j2\pi kn/N} \tag{4.4}$$

4.2 Deriving the DFT from the DTFT

4.2.3 Windowing Effects

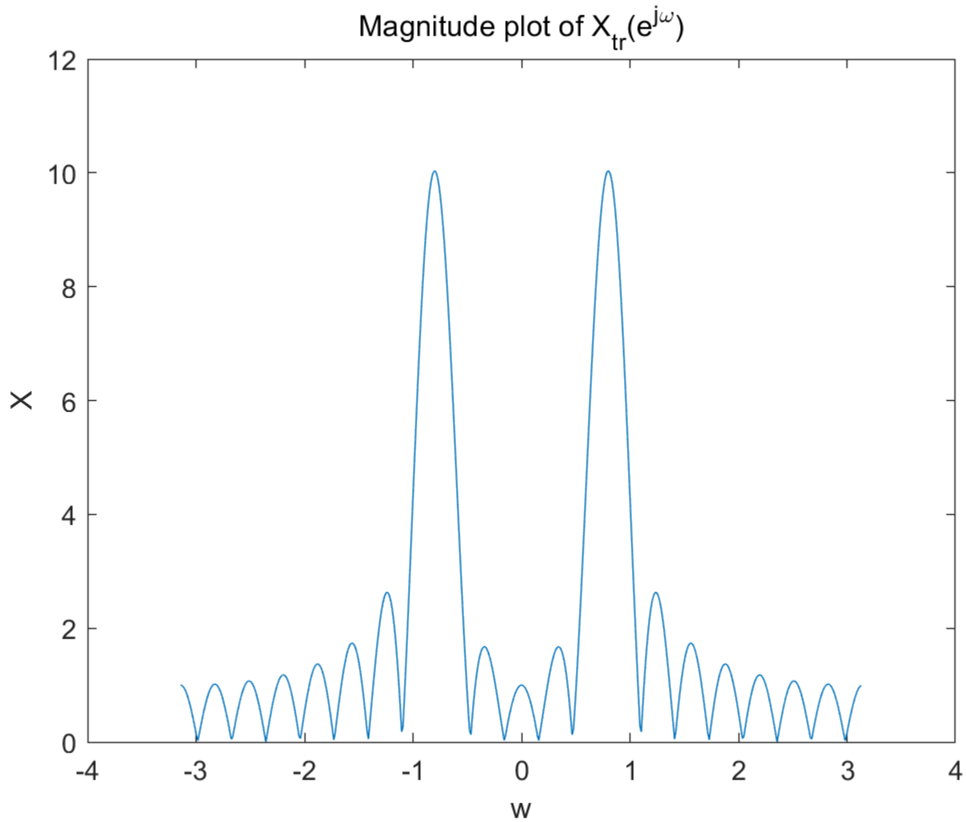
plot the phase and magnitude of W(e^jw)

```
clear;
N=20;
w=-2*pi:2*pi/1000:2*pi;
W = (w~=0).*exp(-j*w*(N-1)/2).*sin(w*N/2)./(sin(w/2))+(w == 0)*N;
figure
subplot(2,1,1),plot(w,abs(W)),xlabel('\omega(Hz)'),ylabel('abs(W)'),title('Magnitude
 response');
subplot(2,1,2),plot(w,unwrap(angle(W))),xlabel('\omega(Hz)'),ylabel('angle(W)'),titl
e('Phase response');
```



Truncate the signal X[n] using a window of size N=20 and then use DTFT.m to compute Xtr(e^jw).

```
clear;
% Truncate the signal
n=0:19;
x=cos(pi/4*n);
[X,w]=DTFT(x,512);
figure
plot(w,abs(X)),xlabel('w'),ylabel('X'),title('Magnitude plot of X_{tr}
(e^{j\omega})');
```



two time-shiftings of a sinc waves

4.3 The Discrete Fourier Transform

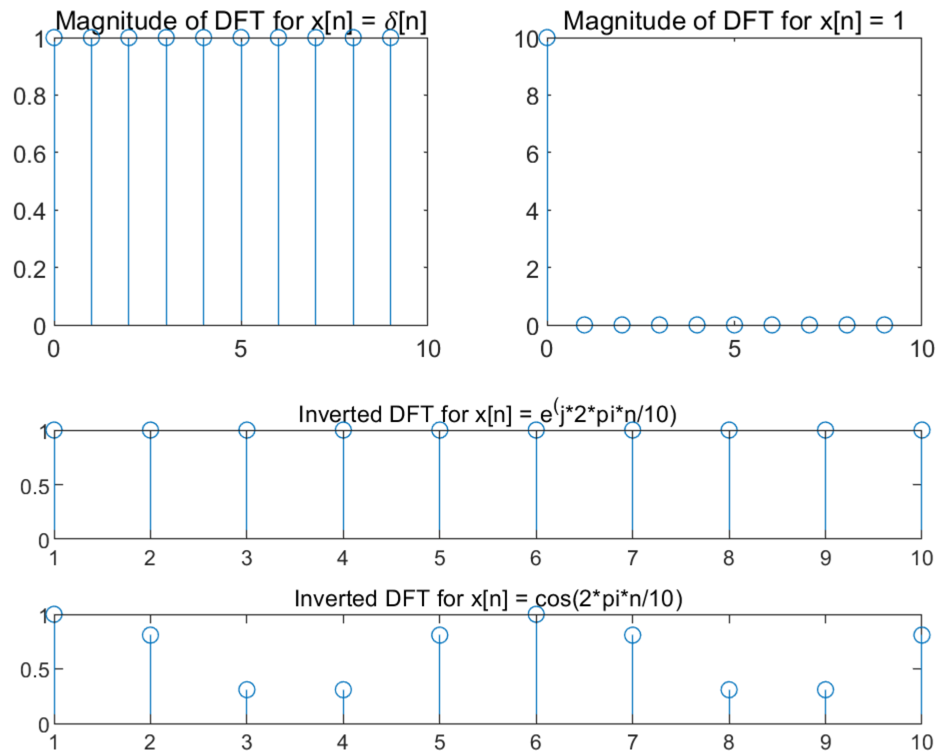
4.3.1 Computing the DFT

```
N = 10;
n=[0:9]%????????
```

```
n = 1×10
     0     1     2     3     4     5     6     7     8     9
```

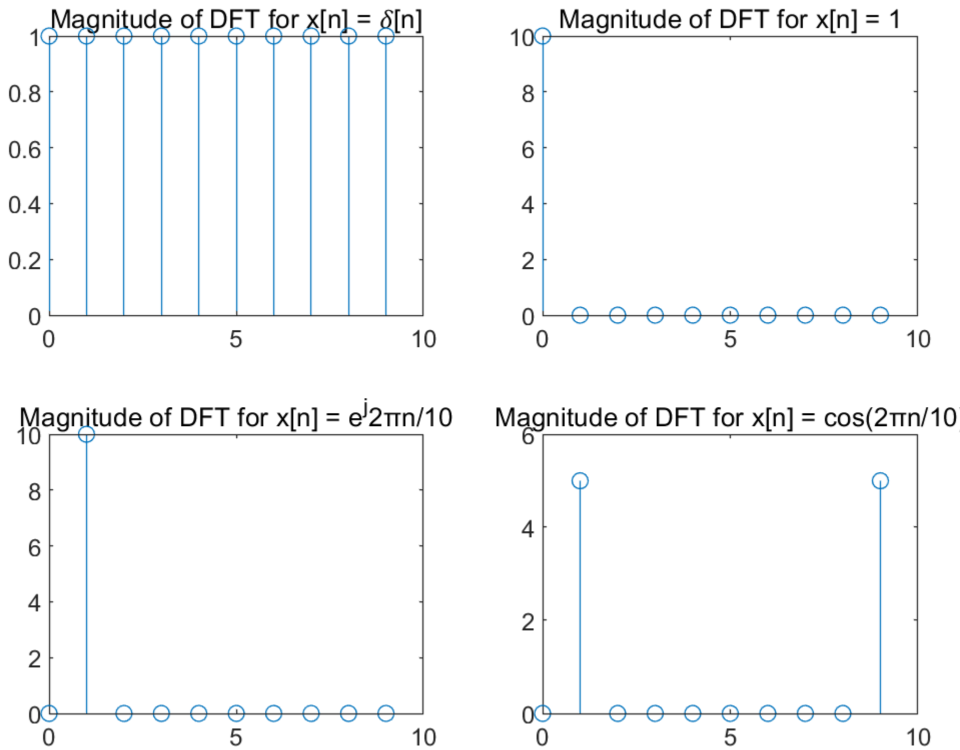
```
% : x[n] = �[n]
x1 = [1 zeros(1, N-1)];
X1 = DFTsum(x1);
subplot(2, 2, 1);
stem(n,abs(X1));
title('Magnitude of DFT for x[n] = �[n]');

%x[n] = 1
x2 = ones(1, N);
X2 = DFTsum(x2);
subplot(2, 2, 2);
stem(n,abs(X2));
title('Magnitude of DFT for x[n] = 1');
```




```
% x[n] = e^j2πn/10
j = sqrt(-1);
x3 = exp(j*2*pi*(0:N-1)/N);
X3 = DFTsum(x3);
subplot(2, 2, 3);
stem(n,abs(X3));
title('Magnitude of DFT for x[n] = e^j2πn/10');

% x[n] = cos(2πn/10)
x4 = cos(2*pi*(0:N-1)/N);
X4 = DFTsum(x4);
subplot(2, 2, 4);
stem(n,abs(X4));
title('Magnitude of DFT for x[n] = cos(2πn/10)');
```



逆变换

IDFT

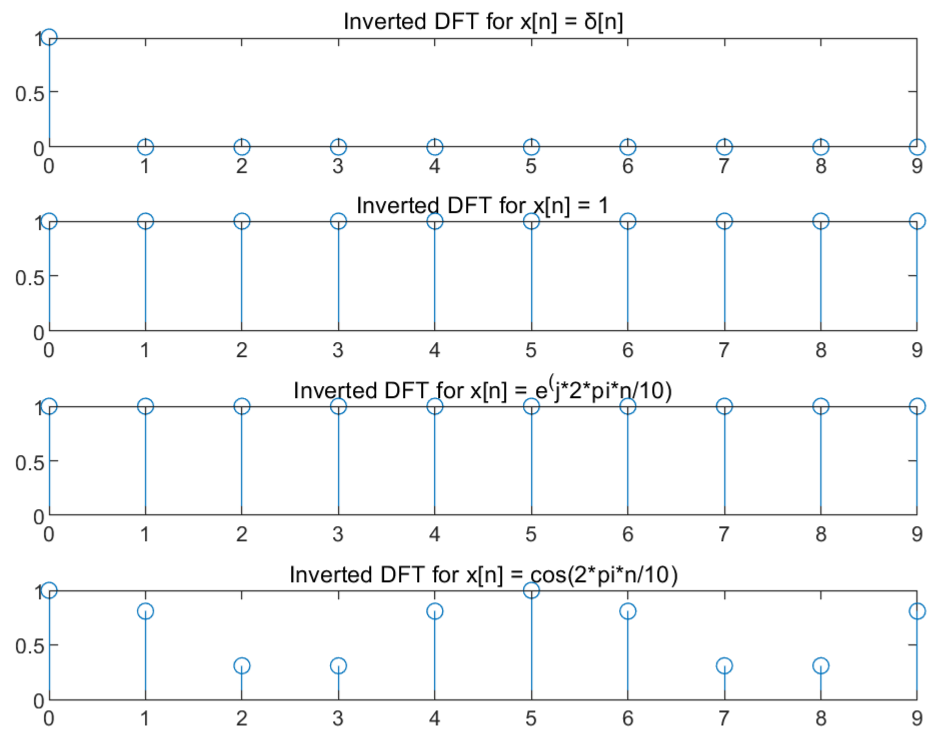
```
x1_inv = IDFTsum(X1);
x2_inv = IDFTsum(X2);
x3_inv = IDFTsum(X3);
x4_inv = IDFTsum(X4);


figure;
subplot(4,1,1);
stem(n,abs(x1_inv));
title('Inverted DFT for x[n] = δ[n]');

subplot(4,1,2);
stem(n,abs(x2_inv));
title('Inverted DFT for x[n] = 1');

subplot(4,1,3);
stem(n,abs(x3_inv));
title('Inverted DFT for x[n] = e^(j*2*pi*n/10)');
```

```
subplot(4,1,4);
stem(n,abs(x4_inv));
title('Inverted DFT for x[n] = cos(2*pi*n/10)');
```



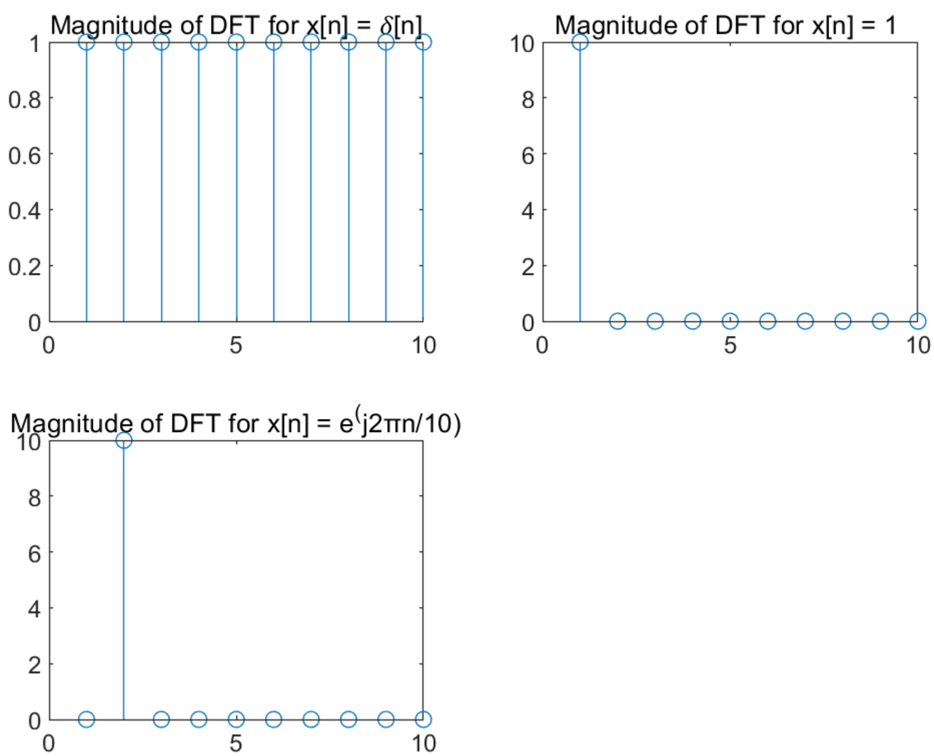
由于 DFT 和 IDFT 是一对互逆变换，所以我们可以验证，通过 IDFTsum 计算得到的原始信号的幅度图与输入到 DFTsum 的原始信号的幅度图一致

4.3.2 Matrix Representation of the DFT

```
N = 10;
A = DFTmatrix(N);
figure
% x[n] = �[n]
x1 = [1 zeros(1, N-1)].';
X1_matrix = A * x1;
subplot(2, 2, 1);
stem(abs(X1_matrix));
title('Magnitude of DFT for x[n] = �[n]');

%x[n] = 1
x2 = ones(N, 1);
X2_matrix = A * x2;
subplot(2, 2, 2);
stem(abs(X2_matrix));
title('Magnitude of DFT for x[n] = 1 ');
```

```
% x[n] = e^j2πn/10
j = sqrt(-1);
x3 = exp(j*2*pi*(0:N-1)/N).';
X3_matrix = A * x3;
subplot(2, 2, 3);
stem(abs(X3_matrix));
title('Magnitude of DFT for x[n] = e^(j2πn/10)');
```




```
% N=5 时的 DFT 矩阵 A
DFTmatrix(5)
```

```
ans = 5×5 complex
   1.0000 + 0.0000i   1.0000 + 0.0000i   1.0000 + 0.0000i   1.0000 + 0.0000i ···
   1.0000 + 0.0000i   0.3090 - 0.9511i  -0.8090 - 0.5878i  -0.8090 + 0.5878i
   1.0000 + 0.0000i  -0.8090 - 0.5878i   0.3090 + 0.9511i   0.3090 - 0.9511i
   1.0000 + 0.0000i  -0.8090 + 0.5878i   0.3090 - 0.9511i   0.3090 + 0.9511i
   1.0000 + 0.0000i   0.3090 + 0.9511i  -0.8090 + 0.5878i  -0.8090 - 0.5878i
```

计算 N 点 DFT 需要的乘法次数：使用矩阵方法计算 N 点 DFT，其实就是进行一个 N x N 矩阵与 N x 1 向量的乘法，这需要 N^2 次乘法

1. 逆 DFT 矩阵�的：��� = �^�2�(�−1)(�−1)/�，这与 DFT 矩阵 A 的元素表达式相似，只是指数的符号变为正。
2. 下面是用于生成 N x N 的逆 DFT 矩阵 B 的 Matlab 函数 IDFTmatrix(N)。

```
N = 5;
```

```
A = DFTmatrix(N);
B = IDFTmatrix(N);
C = B * A;


disp(B);
```

```
   0.2000 + 0.0000i   0.2000 + 0.0000i   0.2000 + 0.0000i   0.2000 + 0.0000i   0.2000 + 0.0000i
   0.2000 + 0.0000i   0.0618 + 0.1902i  -0.1618 + 0.1176i  -0.1618 - 0.1176i   0.0618 - 0.1902i
   0.2000 + 0.0000i  -0.1618 + 0.1176i   0.0618 - 0.1902i   0.0618 + 0.1902i  -0.1618 - 0.1176i
   0.2000 + 0.0000i  -0.1618 - 0.1176i   0.0618 + 0.1902i   0.0618 - 0.1902i  -0.1618 + 0.1176i
   0.2000 + 0.0000i   0.0618 - 0.1902i  -0.1618 - 0.1176i  -0.1618 + 0.1176i   0.0618 + 0.1902i
```

```
disp(C);
```

```
   1.0000 + 0.0000i  -0.0000 - 0.0000i  -0.0000 - 0.0000i   0.0000 - 0.0000i   0.0000 - 0.0000i
  -0.0000 + 0.0000i   1.0000 - 0.0000i  -0.0000 - 0.0000i   0.0000 - 0.0000i   0.0000 - 0.0000i
   0.0000 + 0.0000i  -0.0000 + 0.0000i   1.0000 - 0.0000i  -0.0000 - 0.0000i  -0.0000 - 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 - 0.0000i   1.0000 + 0.0000i  -0.0000 - 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i  -0.0000 + 0.0000i  -0.0000 + 0.0000i   1.0000 + 0.0000i
```

矩阵 C 是一个单位矩阵，这是因为 DFT 矩阵 A 和其逆矩阵 B 的乘积应该得到单位矩阵，这是线性代数的基本性质。这也验证了我们的 DFT 矩阵和逆 DFT 矩阵的计算是正确的。

4.3.3 Computation Time Comparison

比较直接使用循环进行 DFT 和使用矩阵进行 DFT

在 Matlab 中应尽量避免使用循环，而尽可能地使用矩阵/向量乘法。

```
N = 4096;
x = cos(2*pi*(0:N-1)/10).'; % x[n]
A = DFTmatrix(N);

% 计算 DFTsum(x)的 CPU 时间
start_time = cputime;
X_DFTsum = DFTsum(x);
end_time = cputime;
DFTsum_time = end_time - start_time
```

```
DFTsum_time = 3.6250
```

```
% 计算 A*x 的 CPU 时间
start_time = cputime;
X_matrix = A * x;
end_time = cputime;
matrix_time = end_time - start_time
```

```
matrix_time = 0.0625
```