02_SignalsAndSystems

September 25, 2016

1 II. Discrete signals and systems

1.1 II.1 Discrete signals

1.1.1 Representation

A discrete signal can represented:

- graphically
- in table form
- as a vector: $x[n] = [..., 0, 0, 1, 3, 4, 5, 0, ...]$, with an **arrow** indicating the origin of time ($n = 0$). If the arrow is missing, the origin of time is at the first element. The dots ... indicate that the value remains the same from that point onwards

  Examples: blackboard
  $x[4]$ represents the value of the fourth sample in the signal $x[n]$.

1.1.2 Basic signals

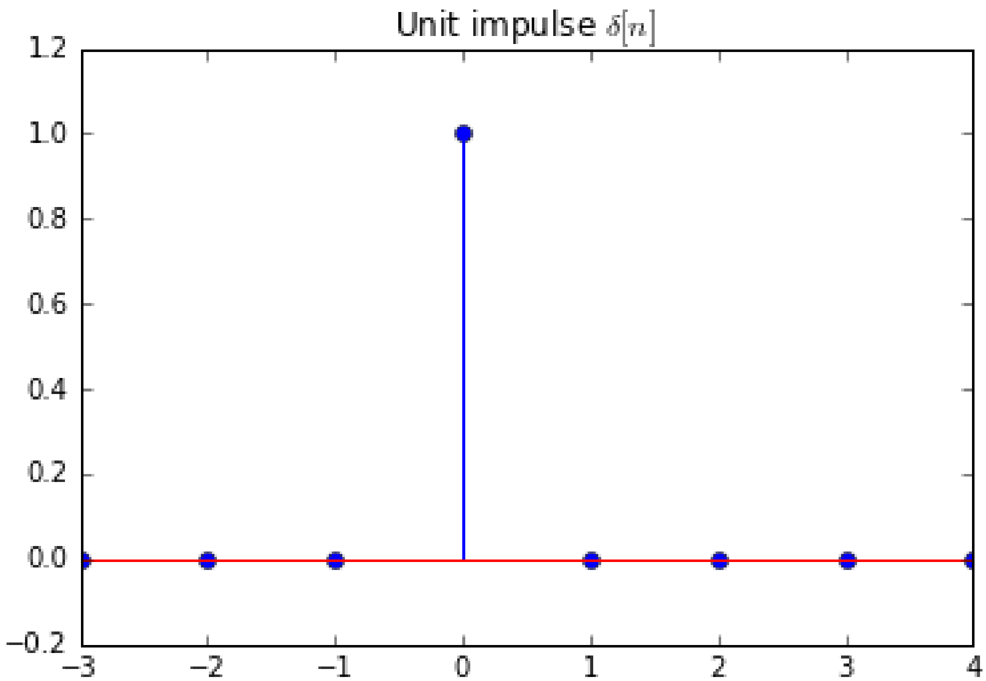
Some elementary signals are presented below.

Unit impulse Contains a single non-zero value of 1 located at time 0. It is denoted with $\delta[n]$.

$$\delta[n] = \begin{cases} 1 & \text{if } n = 0 \\ 0 & \text{otherwise} \end{cases}.$$

```
In [84]: %matplotlib inline
         import matplotlib.pyplot as plt, numpy as np
         delta = [0,0,0,1,0,0,0,0]
         t = range(-3,-3+len(delta))
         plt.stem(t,delta); plt.title ('Unit impulse $\delta[n]$')
         plt.axis([t[0],t[-1],-0.2,1.2])

Out[84]: [-3, 4, -0.2, 1.2]
```

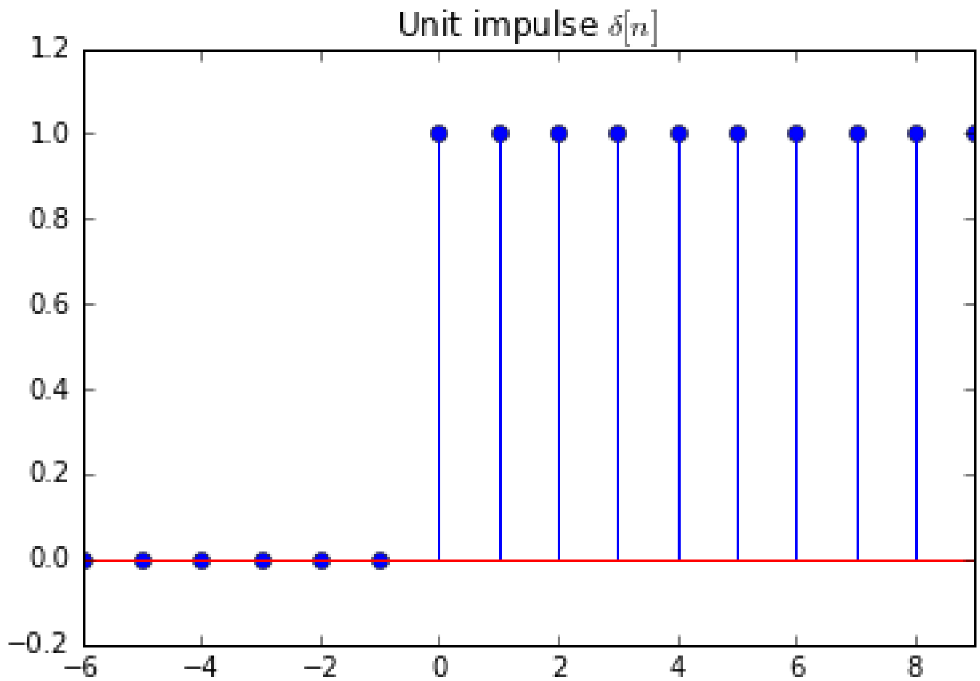


Unit step It is denoted with $u[n]$.

$$u[n] = \begin{cases} 1 & \text{if } n \geq 0 \\ 0 & \text{otherwise} \end{cases}.$$

In [85]:
```
%matplotlib inline
import matplotlib.pyplot as plt, numpy as np
delta = [0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1]
t = range(-6,-6+len(delta))
plt.stem(t,delta); plt.title ('Unit impulse $\delta[n]$')
plt.axis([t[0],t[-1],-0.2,1.2])
```

Out[85]:
```
[-6, 9, -0.2, 1.2]
```

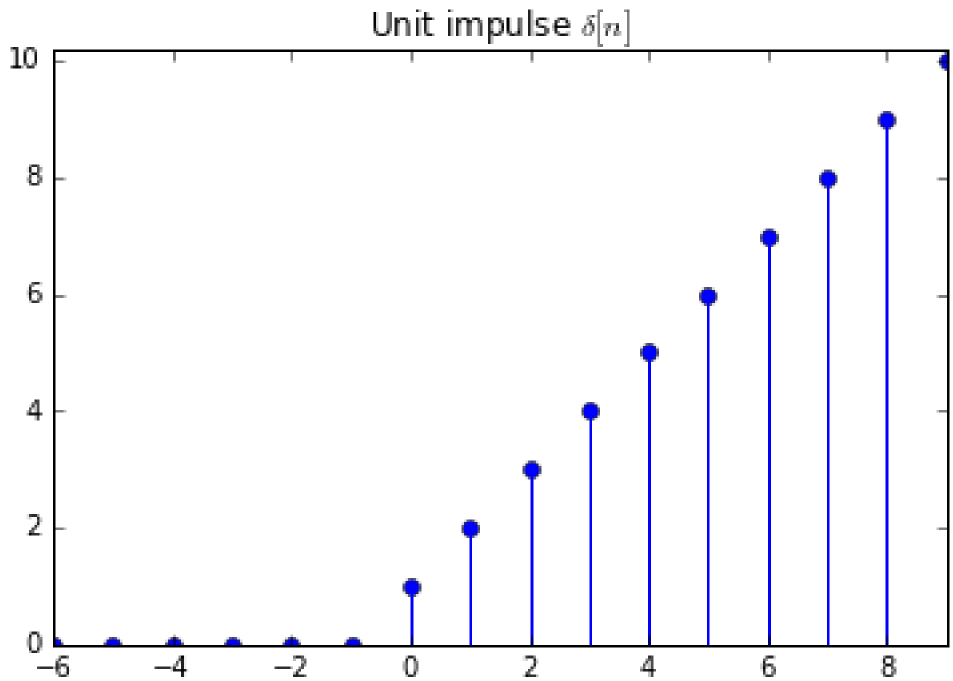


Unit ramp It is denoted with $u_r[n]$.

$$u_r[n] = \begin{cases} n & \text{if } n \geq 0 \\ 0 & \text{otherwise} \end{cases}.$$

```
In [86]: %matplotlib inline
         import matplotlib.pyplot as plt, numpy as np
         delta = [0,0,0,0,0,0,1,2,3,4,5,6,7,8,9,10]
         t = range(-6,-6+len(delta))
         plt.stem(t,delta); plt.title ('Unit impulse $\delta[n]$')
         plt.axis([t[0],t[-1],0,10.2])

Out[86]: [-6, 9, 0, 10.2]
```



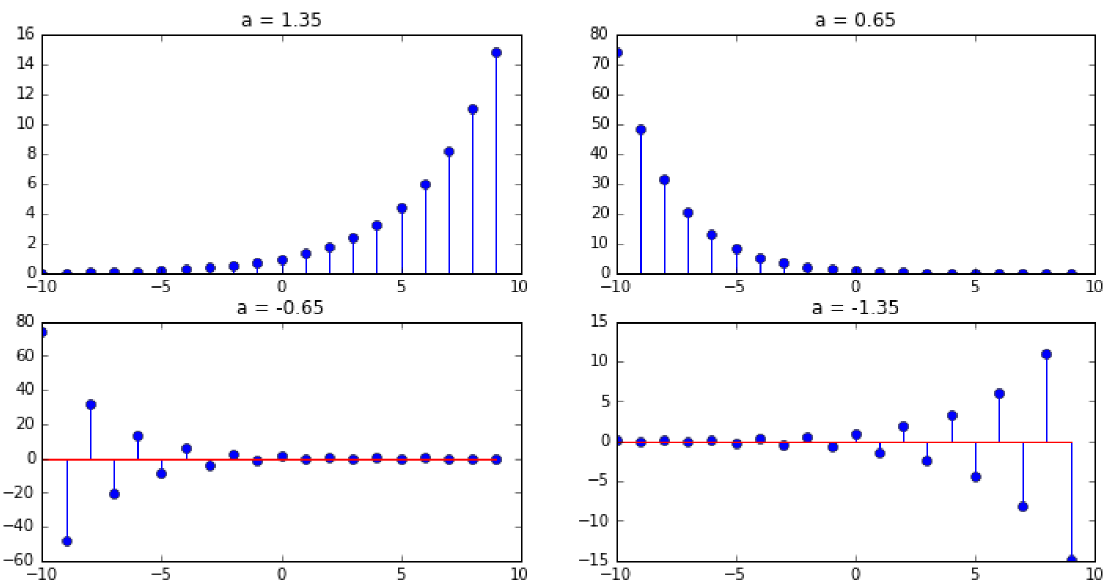
Exponential signal It does not have a special notation. It is defined by:

$$x[n] = a^n.$$

a can be a real or a complex number. Here we consider only the case when a is real. Depending on the value of a, we have four possible cases:

1. $a \geq 1$
2. $0 \leq a < 1$
3. $-1 < a < 0$
4. $a \leq 1$

```
In [87]: %matplotlib inline
         import matplotlib.pyplot as plt, numpy as np
         n = np.arange(-10,10)     # n = [-4,-3,...,4,8]
         x1 = 1.35**n
         x2 = 0.65**n
         x3 = (-0.65)**n
         x4 = (-1.35)**n
         plt.figure(figsize=(12,6))
         plt.subplot(2,2,1); plt.stem(n,x1); plt.title ('a = 1.35');
         plt.subplot(2,2,2); plt.stem(n,x2); plt.title ('a = 0.65');
         plt.subplot(2,2,3); plt.stem(n,x3); plt.title ('a = -0.65');
         plt.subplot(2,2,4); plt.stem(n,x4); plt.title ('a = -1.35');
```



1.2 II.2 Types of discrete signals

1.2.1 Signals with finite energy and finite power

The **energy of a discrete signal** is defined as

$$E = \sum_{n=-\infty}^{\infty} (x[n])^2.$$

If E is finite, the signal is said to have finit energy.
Examples: unit impulse has finite energy; unit step does not.
The **average power of a discrete signal** is defined as

$$P = \lim_{N\to\infty} \frac{\sum_{n=-N}^{N} (x[n])^2}{2N+1}.$$

In other words, the average power is the average energy per sample.
If P is finite, the signal is said to have finite power.
A signal with finite energy has finite power ($P = 0$ if the signal has infinite length). A signal with infinite energy can have finite or infinite power.
Example: unit step has finite power $P = \frac{1}{2}$ (see proof at blackboard).

1.2.2 Periodic and non-periodic signals

A signal is called **periodic** if its values repeat themselves after a certain time (known as **period**).

$$x[n] = x[n+N]), \forall t$$

The **fundamental period** of a signal is the minimum value of N.
Periodic signals have infinite energy, and finite power equal to the power of a single period.

1.2.3 Even and odd signals

A signal is **even** if it satisfies the following symmetry:

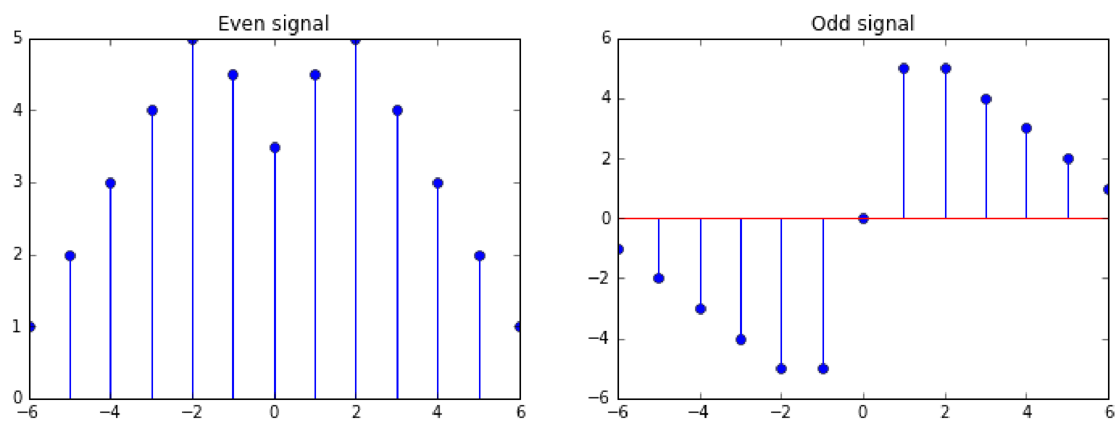
$$x[n] = x[-n], \forall n.$$

A signal is **odd** if it satisfies the following anti-symmetry:

$$-x[n] = x[-n], \forall n.$$

There exist signals which are neither even nor odd.

```
In [91]: %matplotlib inline
         import matplotlib.pyplot as plt, numpy as np
         n = np.arange(-6,7)
         x1 = [1, 2, 3, 4, 5, 4.5, 3.5, 4.5, 5, 4, 3, 2, 1]
         x2 = [-1, -2, -3, -4, -5, -5, 0, 5, 5, 4, 3, 2, 1]
         plt.figure(figsize=(12,4))
         plt.subplot(1,2,1); plt.stem(n,x1); plt.title ('Even signal');
         plt.subplot(1,2,2); plt.stem(n,x2); plt.title ('Odd signal');
```



Every signal can be written as the sum of an even signal and an odd signal:

$$x[n] = x_e[n] + x_o[n].$$

The even and the odd parts of the signal can be found as follows:

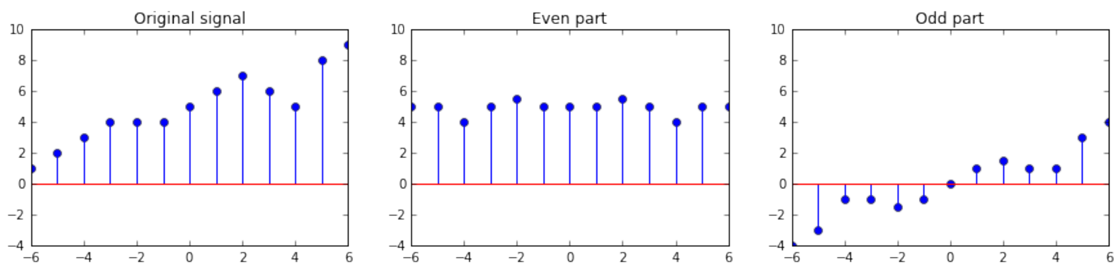
$$x_e[n] = \frac{x[n] + x[-n]}{2}.$$

$$x_o[n] = \frac{x[n] - x[-n]}{2}.$$

Proof: check that $x_e[n]$ is even, $x_o[n]$ is odd, and their sum is $x[n]$
Example

```
In [89]: %matplotlib inline
         import matplotlib.pyplot as plt, numpy as np
         n = np.arange(-6,7)
         x1 = [1, 2, 3, 4, 4, 4, 5, 6, 7, 6, 5, 8, 9]
         xe = (x1 + np.flipud(x1))/2.0
         xo = (x1 - np.flipud(x1))/2.0
         print 'x[n] = ',x1
         print 'xe[n] = ',xe
         print 'xo[n] = ',xo
         plt.figure(figsize=(15,3));
         plt.subplot(1,3,1); plt.stem(n,x1); plt.title ('Original signal'); plt.axi
         plt.subplot(1,3,2); plt.stem(n,xe); plt.title ('Even part'); plt.axis([-6,
         plt.subplot(1,3,3); plt.stem(n,xo); plt.title ('Odd part'); plt.axis([-6,6
```

```
x[n] =  [1, 2, 3, 4, 4, 4, 5, 6, 7, 6, 5, 8, 9]
xe[n] =  [ 5.   5.   4.   5.   5.5  5.   5.   5.   5.5  5.   4.   5.   5. ]
xo[n] =  [-4.  -3.  -1.  -1.  -1.5 -1.   0.   1.   1.5  1.   1.   3.   4. ]
```



1.3 II.3 Basic operations with discrete signals

1.3.1 Time shifting

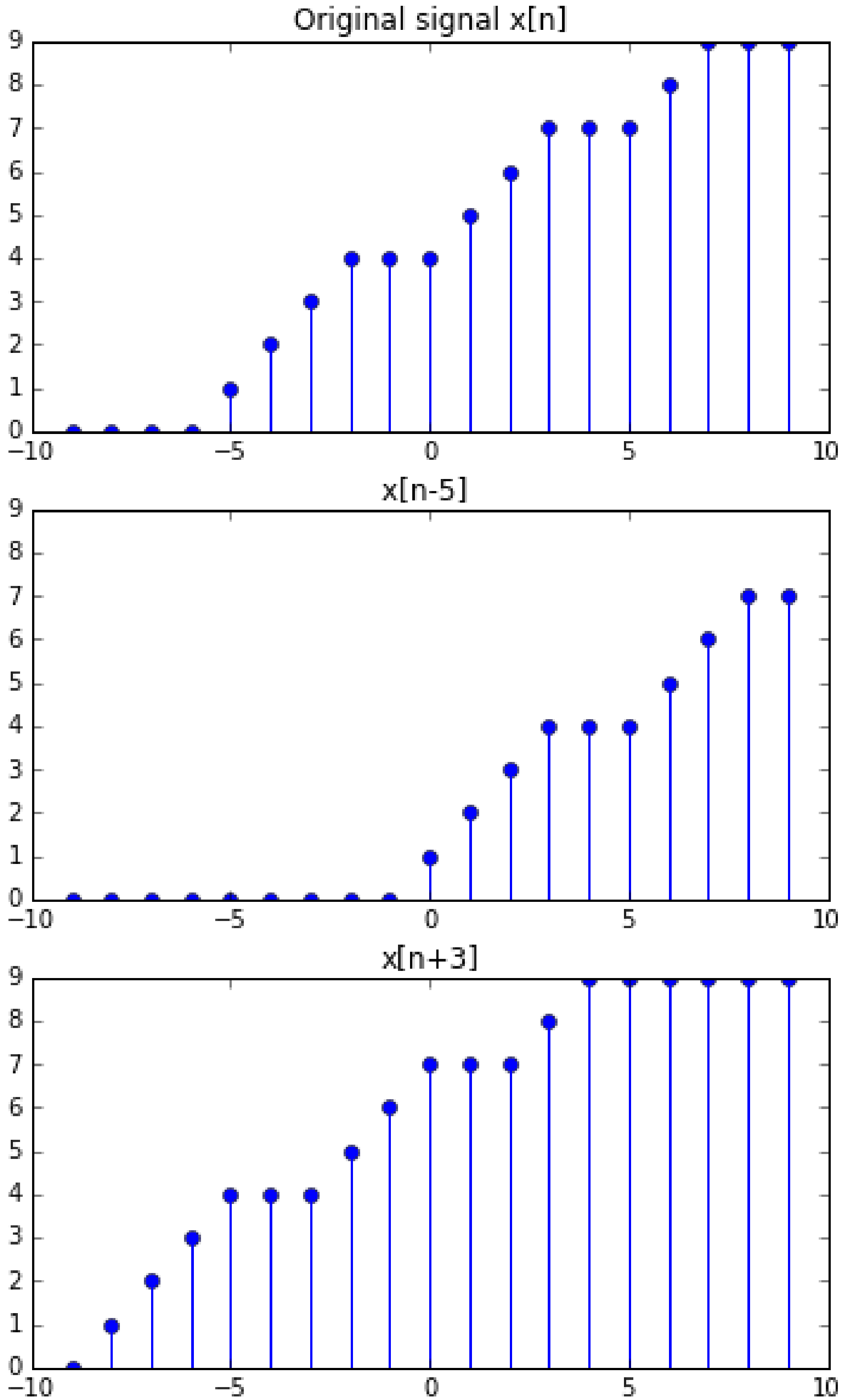
Let $x[n]$ be a signal.

The signal $x[n-k]$ **is** $x[n]$ **delayed with** k **time units**. Graphically, $x[n-k]$ is shifted k units to the **right** compared to the original signal.

The signal $x[n+k]$ **is** $x[n]$ **anticipated with** k **time units**. Graphically, $x[n+k]$ is shifted k units to the **left** compared to the original signal.

```
In [90]: %matplotlib inline
         import matplotlib.pyplot as plt, numpy as np
         n = np.arange(-9,10)
         x1 = [0, 0, 0, 0, 1, 2, 3, 4, 4, 4, 5, 6, 7, 7, 7, 8, 9, 9, 9]
         x2 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 4, 4, 5, 6, 7, 7]
         x3 = [0, 1, 2, 3, 4, 4, 4, 5, 6, 7, 7, 7, 8, 9, 9, 9, 9, 9, 9]
         plt.figure(figsize=(6,10));
         plt.subplot(3,1,1); plt.stem(n,x1); plt.title ('Original signal x[n]')
         plt.subplot(3,1,2); plt.stem(n,x2); plt.title ('x[n-5] '); plt.axis([-10,1
         plt.subplot(3,1,3); plt.stem(n,x3); plt.title ('x[n+3]')
```

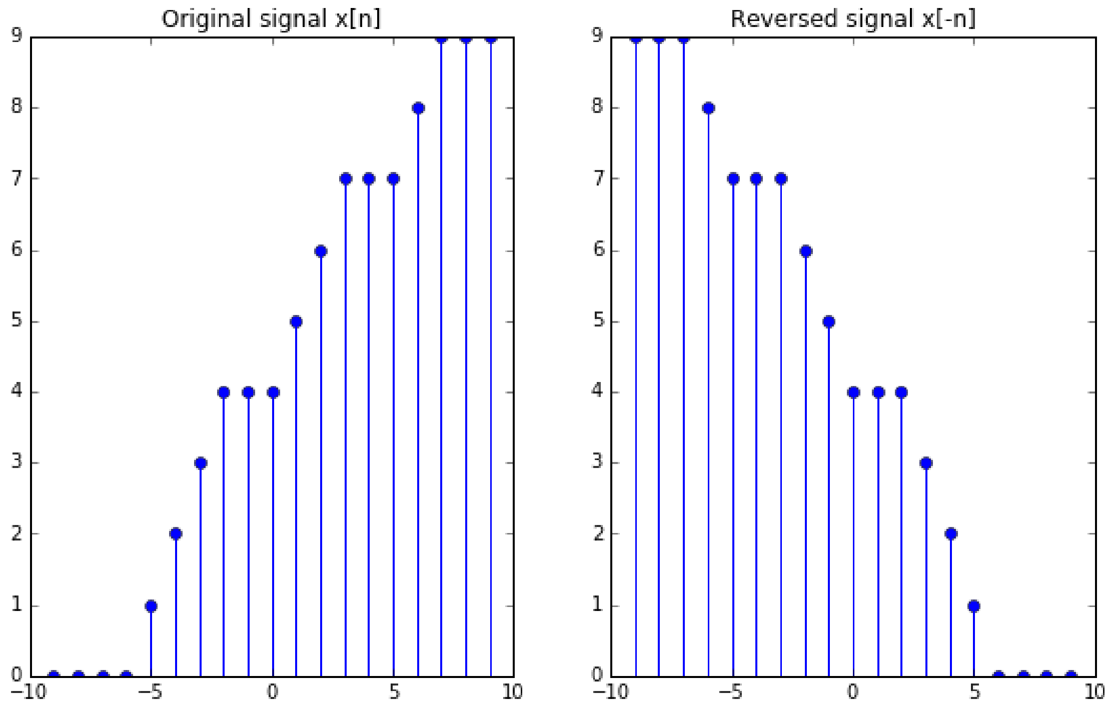
```
Out[90]: <matplotlib.text.Text at 0xac5de0ec>
```



1.3.2 Time reversal

Changing the variable n to $-n$ produces a signal $x[-n]$ which mirrors $x[n]$.

```
In [94]: %matplotlib inline
         import matplotlib.pyplot as plt, numpy as np
         n = np.arange(-9,10)
         x1 = [0, 0, 0, 0, 1, 2, 3, 4, 4, 4, 5, 6, 7, 7, 7, 8, 9, 9, 9]
         x2 = np.flipud(x1)
         plt.figure(figsize=(10,6));
         plt.subplot(1,2,1); plt.stem(n,x1); plt.title ('Original signal x[n]')
         plt.subplot(1,2,2); plt.stem(n,x2); plt.title ('Reversed signal x[-n]');
```

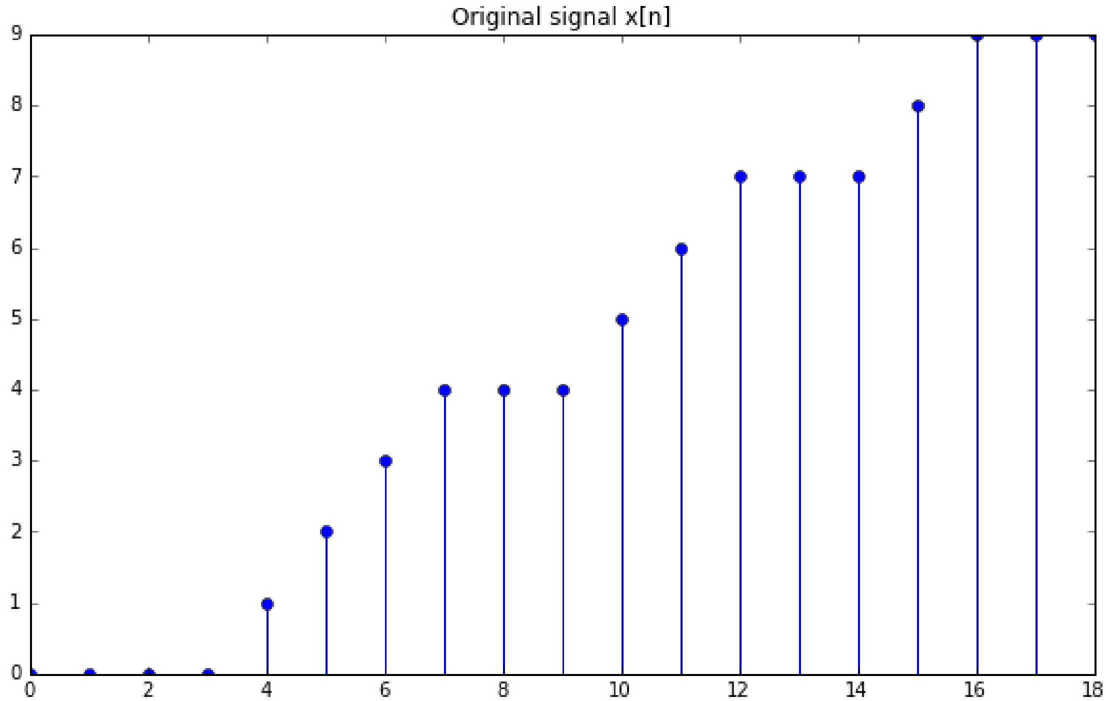


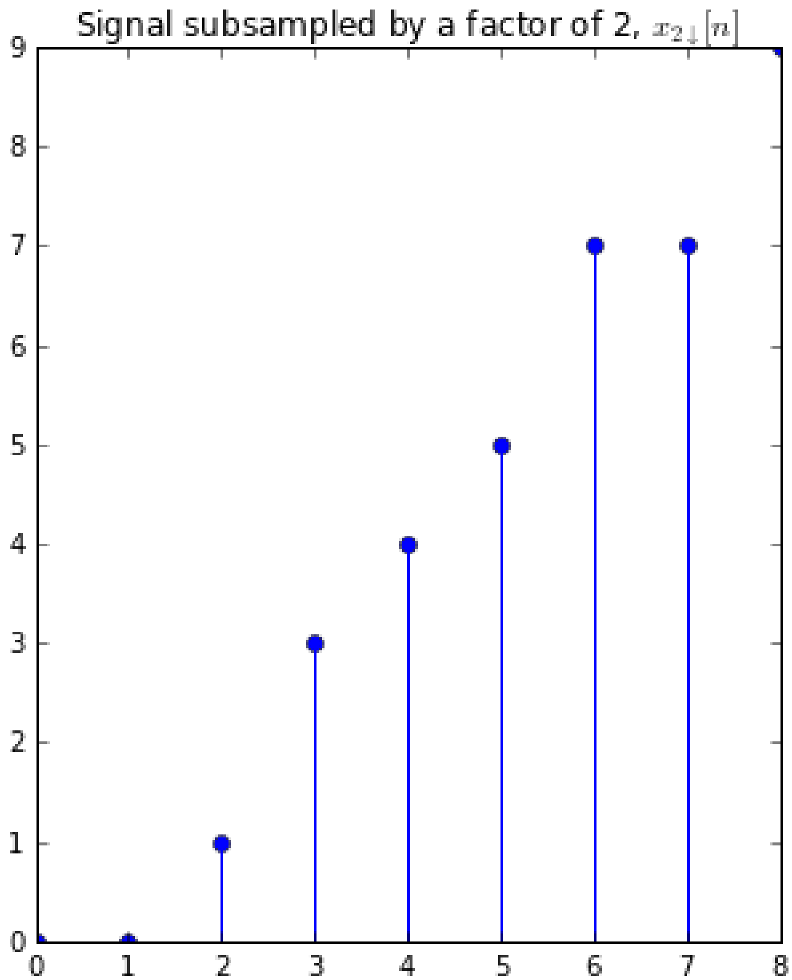
1.3.3 Subsampling

$x_{M\downarrow}[n] = x[Mn]$ is a **subsampled** version of $x[n]$ with a factor of M.
Only 1 sample out of M are kept from the original signal $x[n]$, the rest are discarded.

```
In [117]: %matplotlib inline
          import matplotlib.pyplot as plt, numpy as np
          n = np.arange(0,19)
          x1 = [0, 0, 0, 0, 1, 2, 3, 4, 4, 4, 5, 6, 7, 7, 7, 8, 9, 9, 9]
          x2 = x1[0:-1:2]
```

```
n2 = np.array(n[0:-1:2])/2
plt.figure(figsize=(10,6));
plt.stem(n,x1); plt.title ('Original signal x[n]')
plt.figure(figsize=(5,6));
plt.stem(n2,x2); plt.title ('Signal subsampled by a factor of 2, $x_{2\do
```



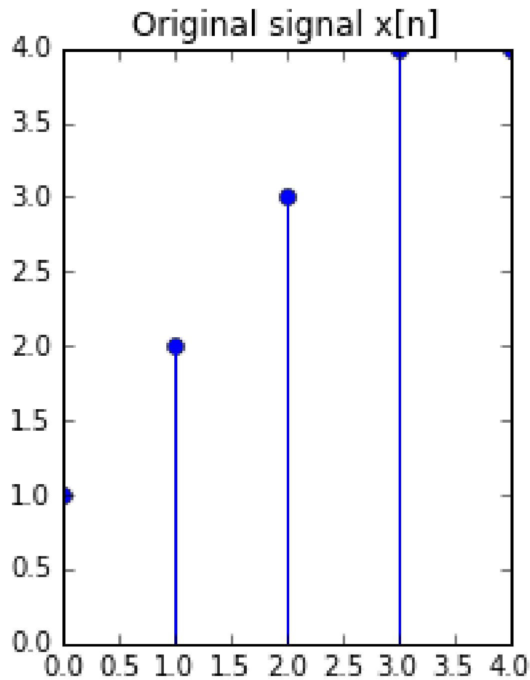


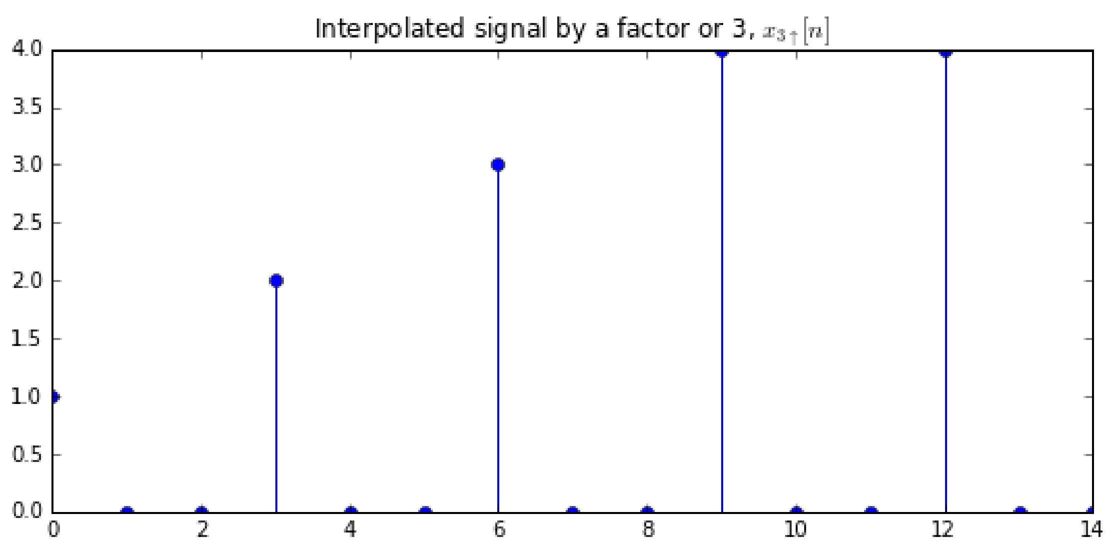
1.3.4 Interpolation

Interpolation by a factor of L adds L of zeros between two samples in the original signal.

$$x_{L\uparrow} = \begin{cases} x[\frac{n}{L}] & \text{if } \frac{n}{L} \in \mathbb{N} \\ 0 & \text{otherwise} \end{cases}.$$

```
In [124]: %matplotlib inline
          import matplotlib.pyplot as plt, numpy as np
          x1 = [1, 2, 3, 4, 4]
          x2 = [1, 0, 0, 2, 0, 0, 3, 0, 0, 4, 0, 0, 4, 0, 0, ]
          plt.figure(figsize=(3,4));
          plt.stem(x1); plt.title ('Original signal x[n]')
          plt.figure(figsize=(9,4));
          plt.stem(x2); plt.title ('Interpolated signal by a factor or 3, $x_{3\upa
```





1.3.5 Mathematical operations

A signal $x[n]$ can be **scaled** by a constant A, i.e. each sample is multipled by A.

$$y[n] = Ax[n].$$

Two signals $x_1[n]$ and $x_2[n]$ can be **summed** by summing the individual samples:

$$y[n] = x_1[n] + x_2[n]$$

Two signals $x_1[n]$ and $x_2[n]$ can be **multiplied** by multiypling the individual samples:

$$y[n] = x_1[n] \cdot x_2[n]$$