02_SignalsAndSystems

January 16, 2017

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

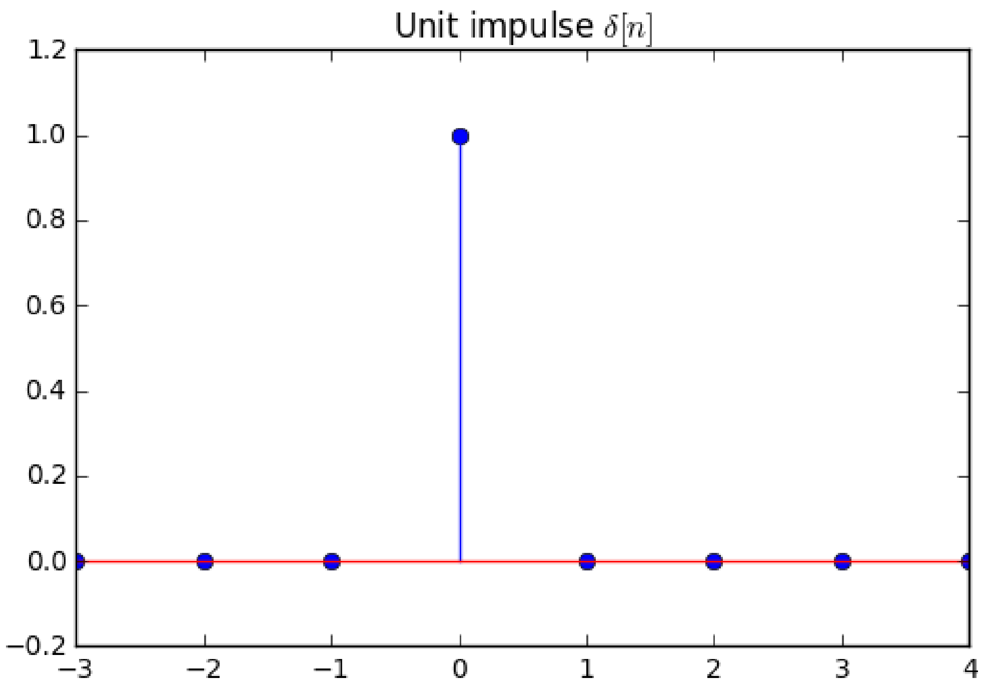
Some elementary signals are presented below.

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 [39]: %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[39]: [-3, 4, -0.2, 1.2]
```

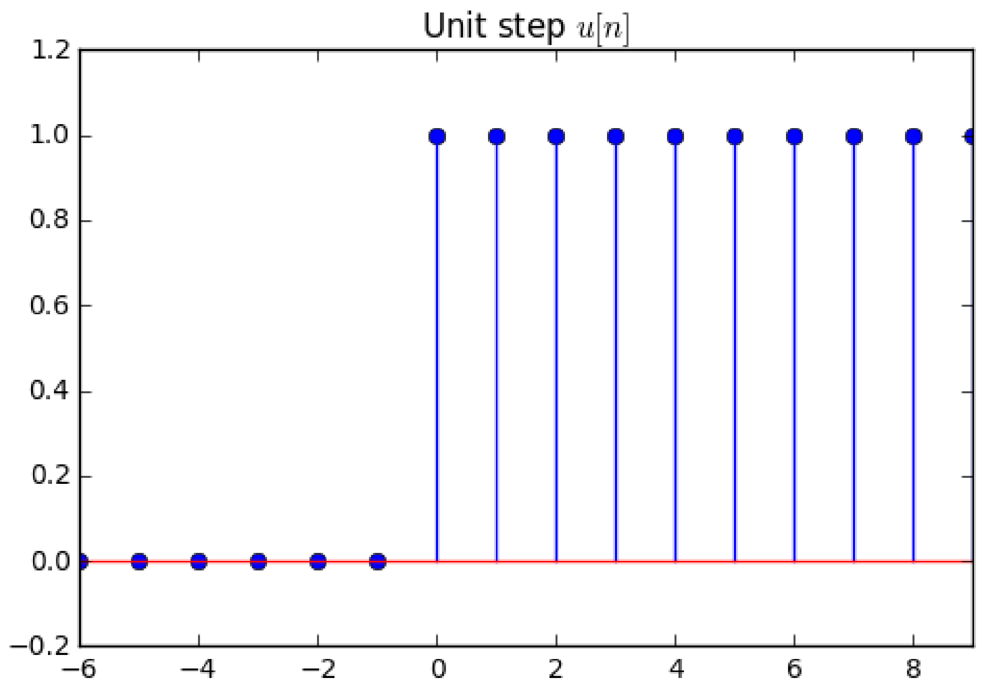


Unit step It is denoted with $u[n]$.

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

```
In [40]: %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 step $u[n]$')
         plt.axis([t[0],t[-1],-0.2,1.2])

Out[40]: [-6, 9, -0.2, 1.2]
```

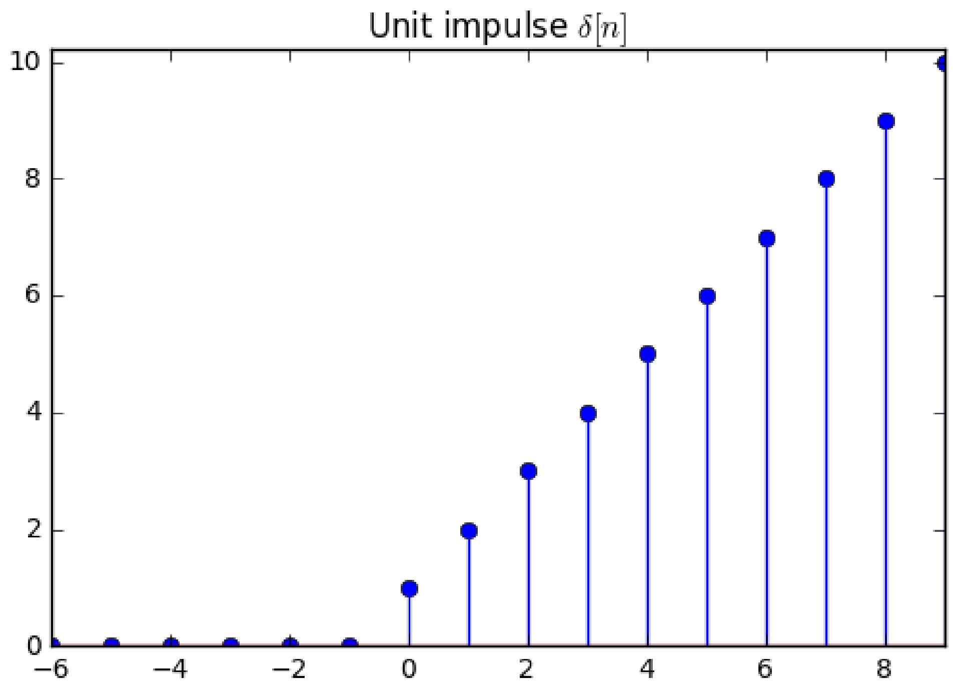


Unit ramp It is denoted with $u_r[n]$.

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

```
In [41]: %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[41]: [-6, 9, 0, 10.2]
```



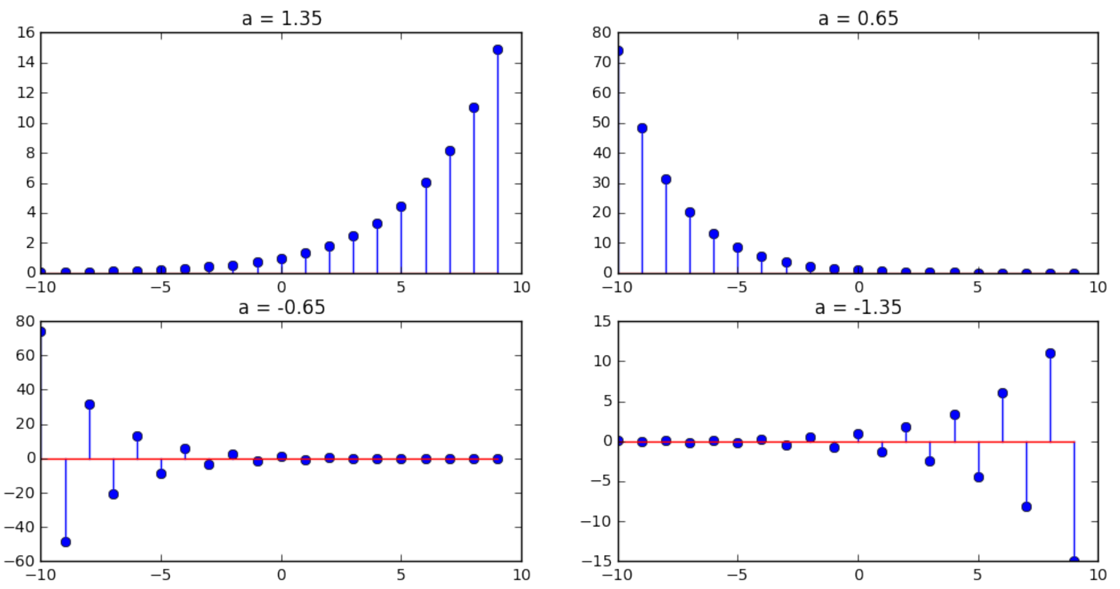
Exponential signal It does not have a special notation. It is defined by:

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

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 [42]: %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:

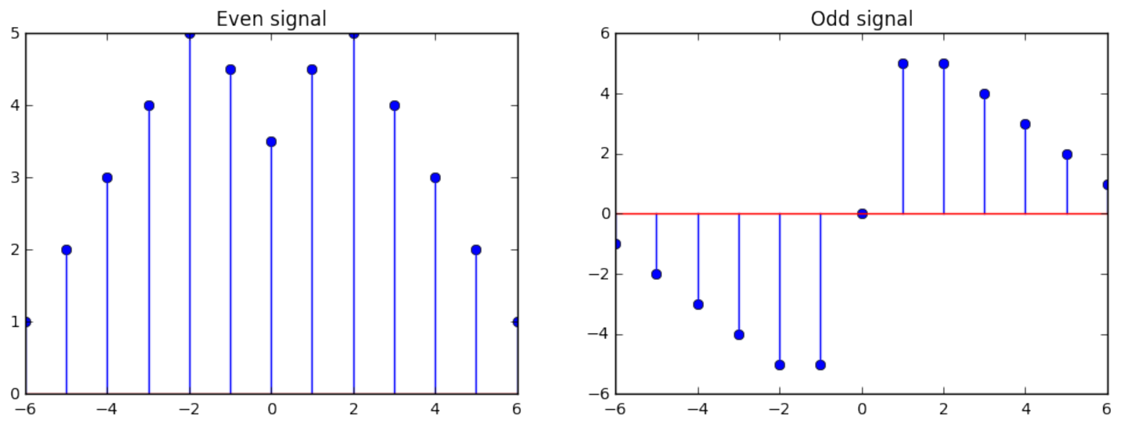
$$x[n] = x[-n], \forall n.$$

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 [43]: %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:

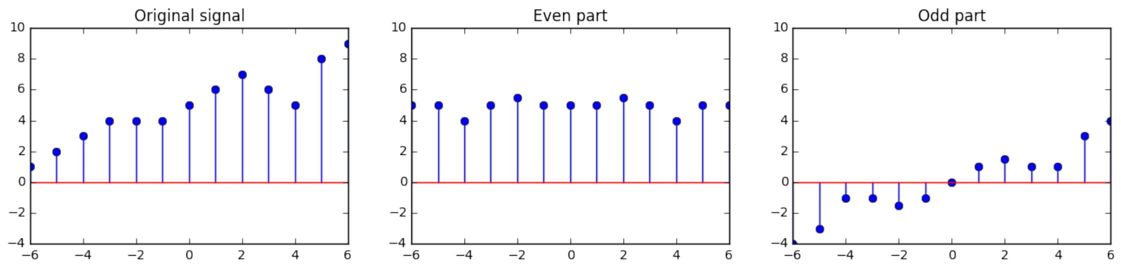
$$x_e[n] = \frac{x[n] + x[-n]}{2}.$$

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

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

```
In [44]: %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

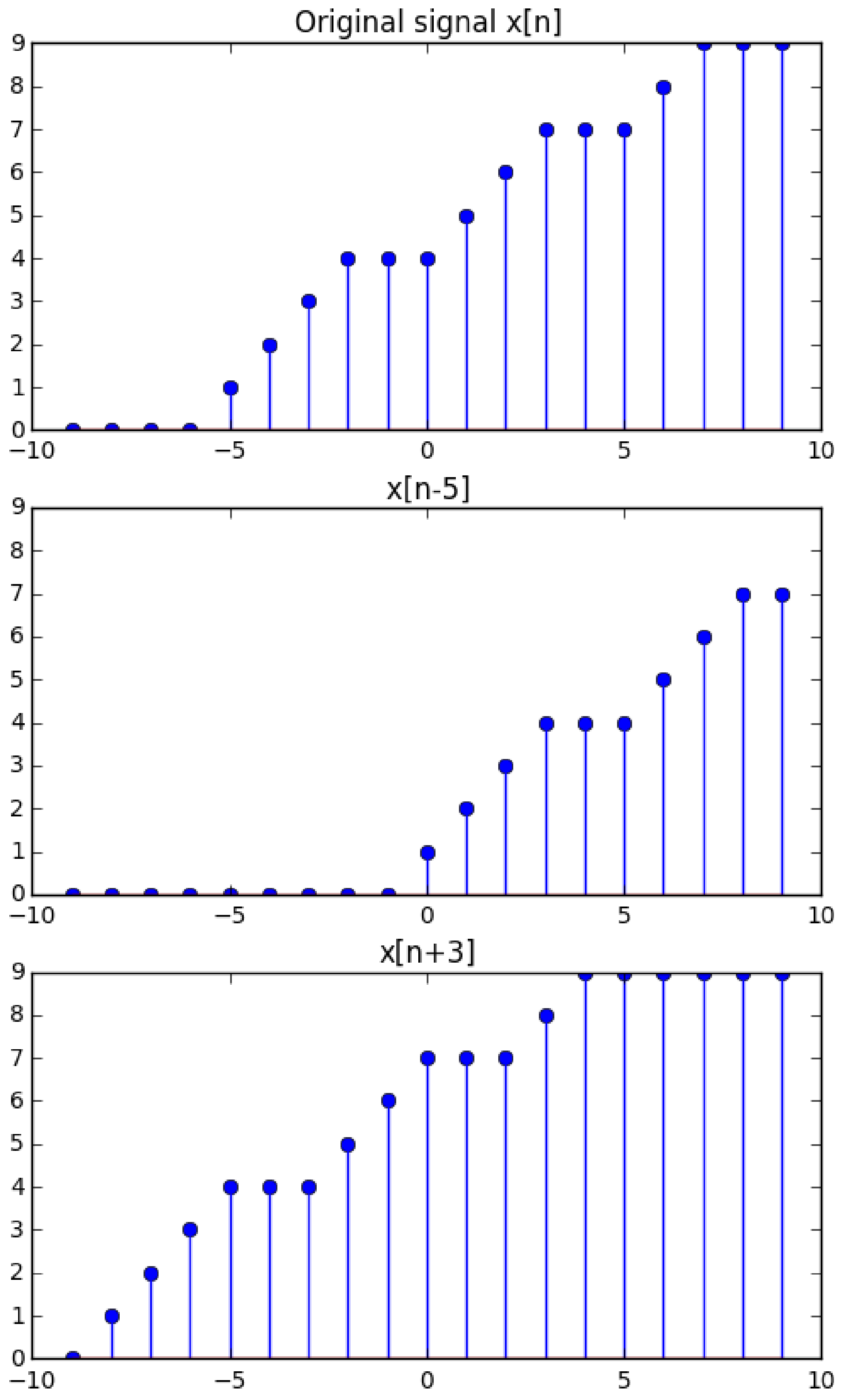
Let $x[n]$ be a signal.

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 [45]: %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]')
```

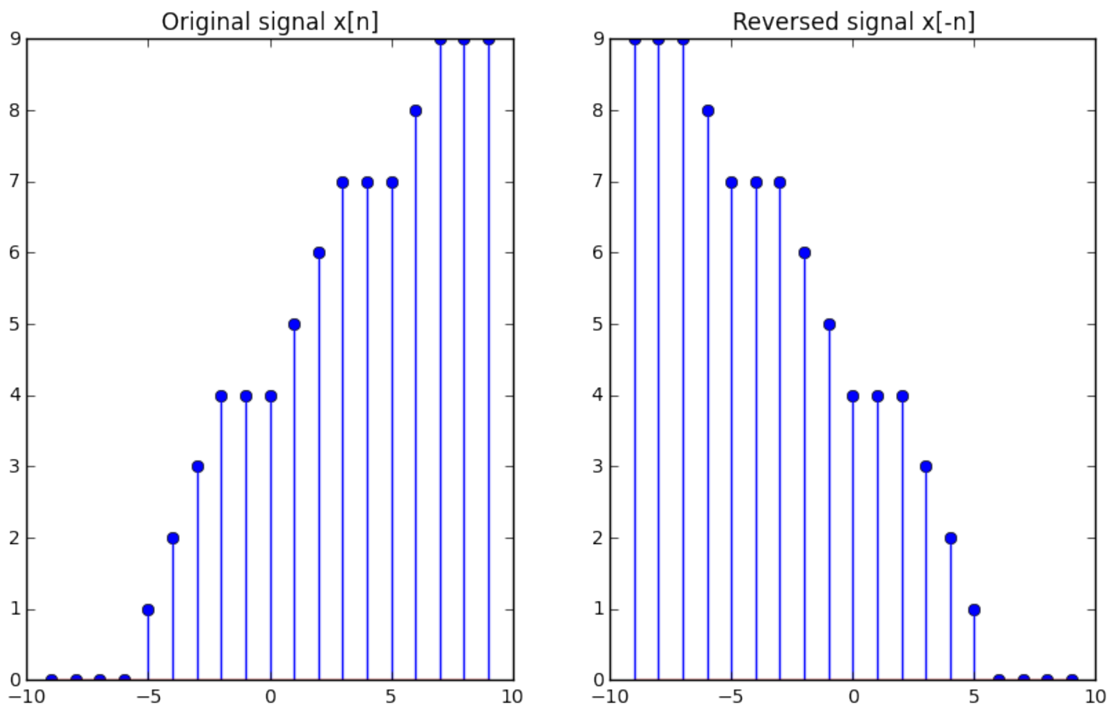
```
Out[45]: <matplotlib.text.Text at 0xaa19294c>
```


Original signal x[n]
x[n-5]
x[n+3]
9
8
7
6
5
4
3
2
1
0
−10
−5
0
5
10

1.3.2 Time reversal

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

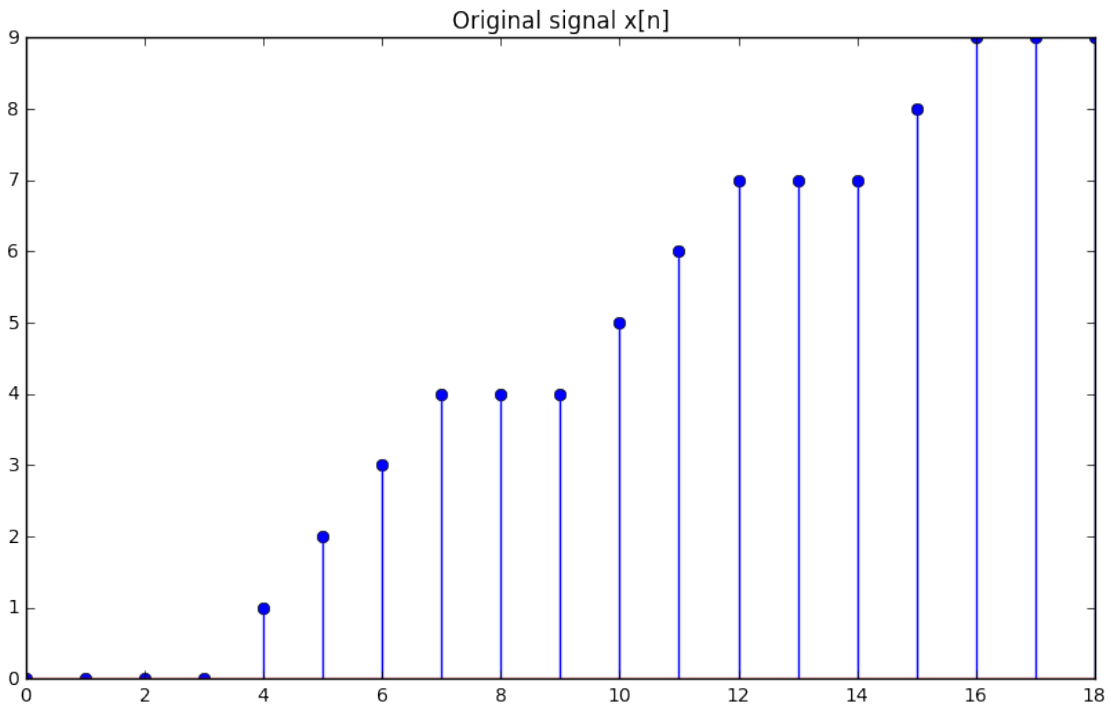
```
In [46]: %matplotlib inline
         import matplotlib.pyplot as plt, numpy as np
         n = np.arange(-9,10)
         x1 = [0, 0, 0, 0, 1, 2, 3, 4, 4, 4, 5, 6, 7, 7, 7, 8, 9, 9, 9]
         x2 = np.flipud(x1)
         plt.figure(figsize=(10,6));
         plt.subplot(1,2,1); plt.stem(n,x1); plt.title ('Original signal x[n]')
         plt.subplot(1,2,2); plt.stem(n,x2); plt.title ('Reversed signal x[-n]');
```

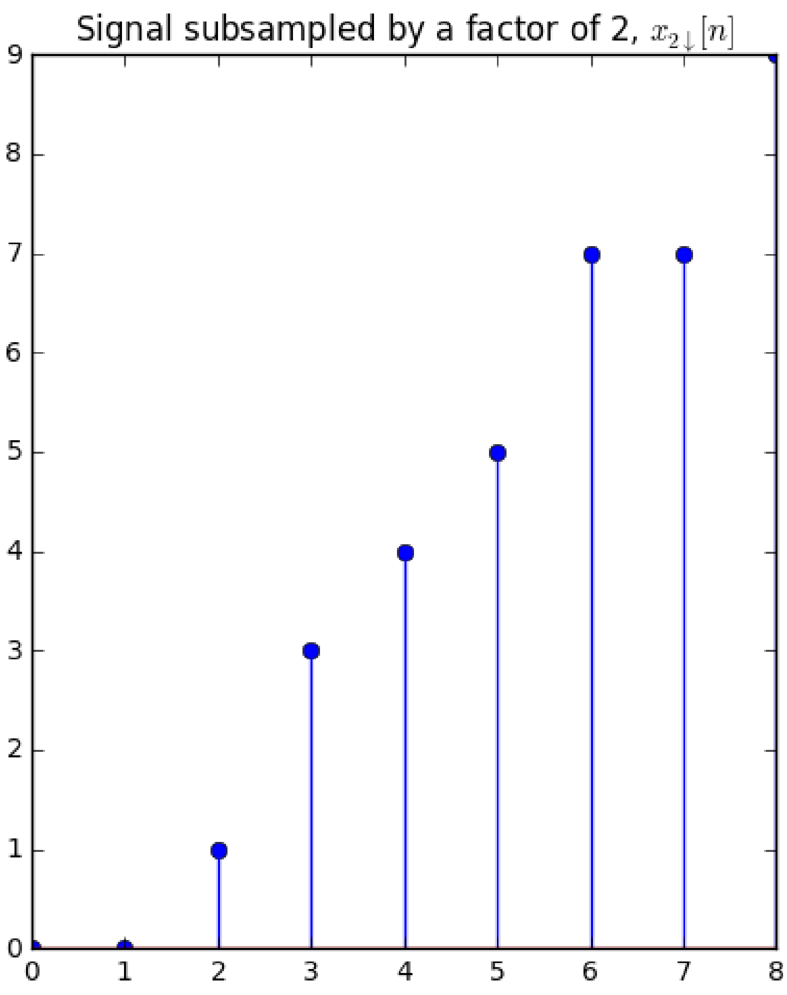


1.3.3 Subsampling

$x_{M\downarrow}[n] = x[Mn]$ 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 [47]: %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\dow
```



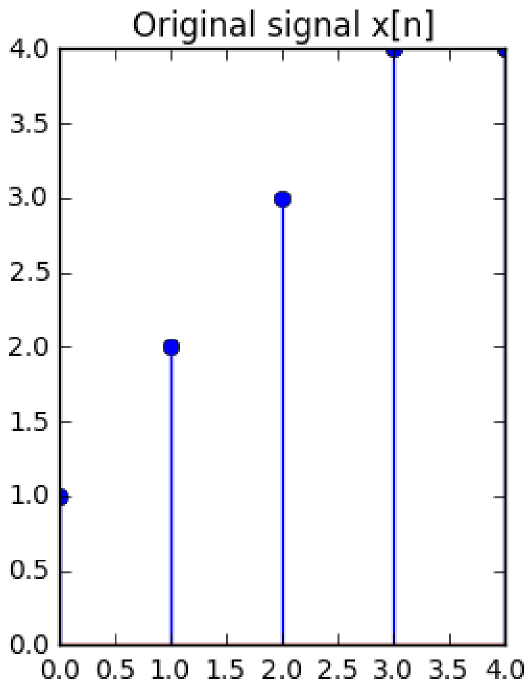


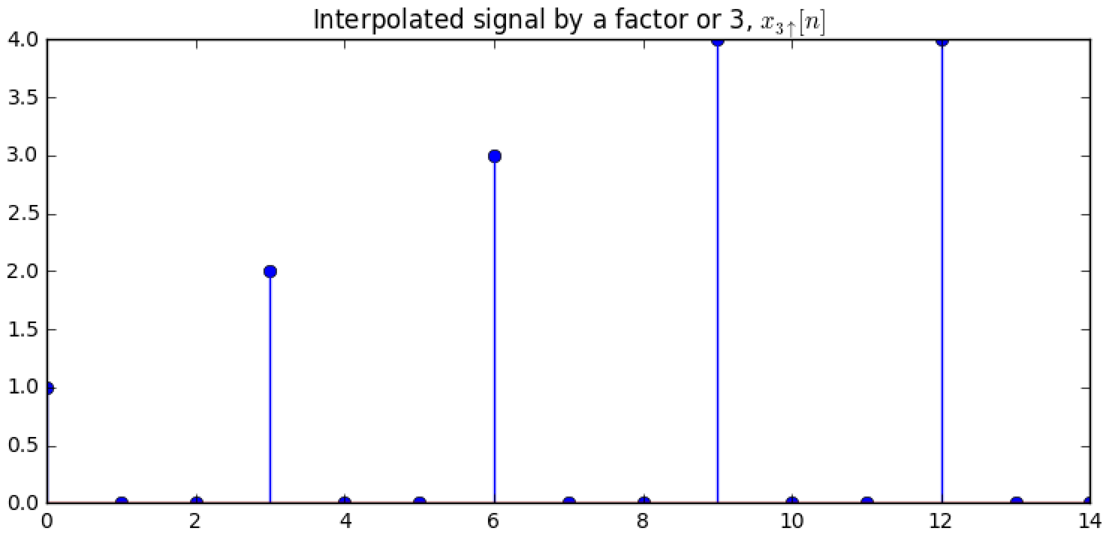
1.3.4 Interpolation

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 [48]: %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\upar
```





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:

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

1.4 II.4 Discrete systems

A **system** is a device or algorithm which produces an **output signal** based on an **input signal**. We will only consiuder systems with a single input and a single output.

Figure here - blackboard.

Common notation: x[n] is the input, y[n] is the output, H is the system.

The relation between the signals can be written as

$$y[n] = H\left[x[n]\right],$$

meaning *"the system H applied to the input x[n] produces the output y[n]"*.

It can also be represented as

$$x[n] \xrightarrow{H} y[n],$$

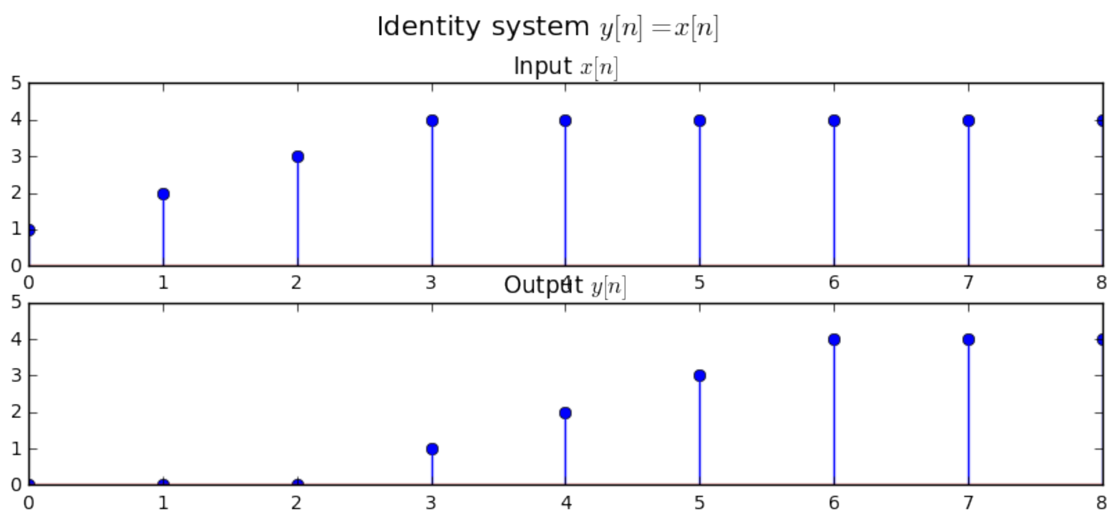
suggesting *"the input $x[n]$ is transformed by the system H into y[n]"*.

Usually, a system is described by the **input-output equation** which expains how $y[n]$ is defined in terms of $x[n]$.

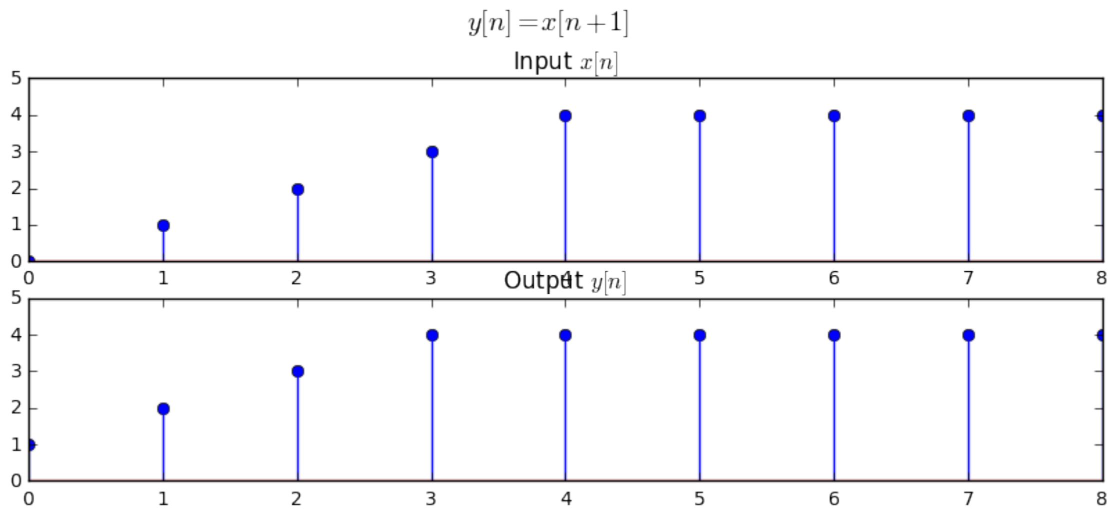
Examples:

1. $y[n] = x[n]$ (the identity system)
2. $y[n] = x[n-3]$
3. $y[n] = x[n+1]$
4. $y[n] = \frac{1}{3}(x[n+1] + x[n] + x[n-1])$
5. $y[n] = \max(x[n+1], x[n], x[n-1])$
6. $y[n] = (x[n])^2 + \log_{10} x[n-1]$
7. $y[n] = \sum_{k=-\infty}^{n} x[k] = x[n] + x[n-1] + x[n-2] + ...$

```
In [49]: %matplotlib inline
         import matplotlib.pyplot as plt, numpy as np
         x = [1, 2, 3, 4, 4, 4, 4, 4, 4, 4, 4]
         y = [0, 0, 0, 1, 2, 3, 4, 4, 4, 4, 4]
         plt.figure(figsize=(10,4));
         plt.subplot(2,1,1); plt.stem(x); plt.title ('Input $x[n]$'); plt.axis([0,
         plt.subplot(2,1,2); plt.stem(y); plt.title ('Output $y[n]$'); plt.axis([0,
         plt.suptitle('Identity system $y[n] = x[n]$', fontsize='x-large')
         plt.gcf().subplots_adjust(top=0.85)
```



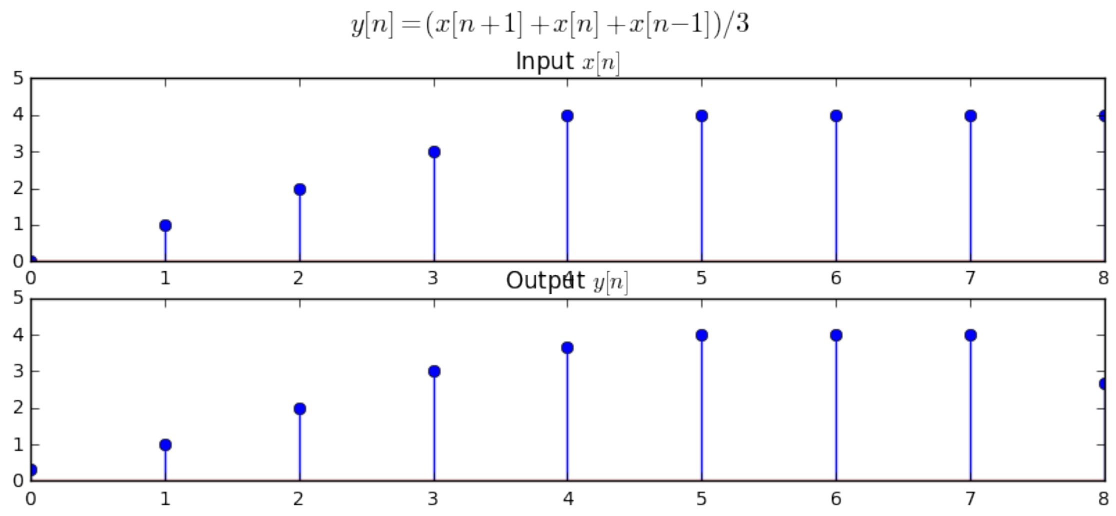
```
In [50]: %matplotlib inline
         import matplotlib.pyplot as plt, numpy as np
         x = [0, 1, 2, 3, 4, 4, 4, 4, 4]
         y = [1, 2, 3, 4, 4, 4, 4, 4, 4]
         plt.figure(figsize=(10,4));
         plt.subplot(2,1,1); plt.stem(x); plt.title ('Input $x[n]$'); plt.axis([0,8
         plt.subplot(2,1,2); plt.stem(y); plt.title ('Output $y[n]$'); plt.axis([0,
         plt.suptitle('$y[n] = x[n+1]$', fontsize='x-large')
         plt.gcf().subplots_adjust(top=0.85)
```



```
In [51]: %matplotlib inline
         import matplotlib.pyplot as plt, numpy as np
```

```
        x = [0, 1, 2, 3, 4, 4, 4, 4, 4]
        for i in range(len(x)):
            if i==0:
                y[i] = (x[i] + x[i+1])/3.0;    # x[-1] does not exist in the vecto
            elif i == len(x)-1:
                y[i] = (x[i-1] + x[i])/3.0;    # x[len(x)] does not exist in the ve
            else:
                y[i] = (x[i-1] + x[i] + x[i+1])/3.0;
        print 'x = ',x
        print ('y = ['+', '.join(['%.2f']*len(y))%tuple(y)+']')
        plt.figure(figsize=(10,4));
        plt.subplot(2,1,1); plt.stem(x); plt.title ('Input $x[n]$'); plt.axis([0,8
        plt.subplot(2,1,2); plt.stem(y); plt.title ('Output $y[n]$'); plt.axis([0,
        plt.suptitle('$y[n] = (x[n+1] + x[n] + x[n-1])/3$', fontsize='x-large')
        plt.gcf().subplots_adjust(top=0.85)
```

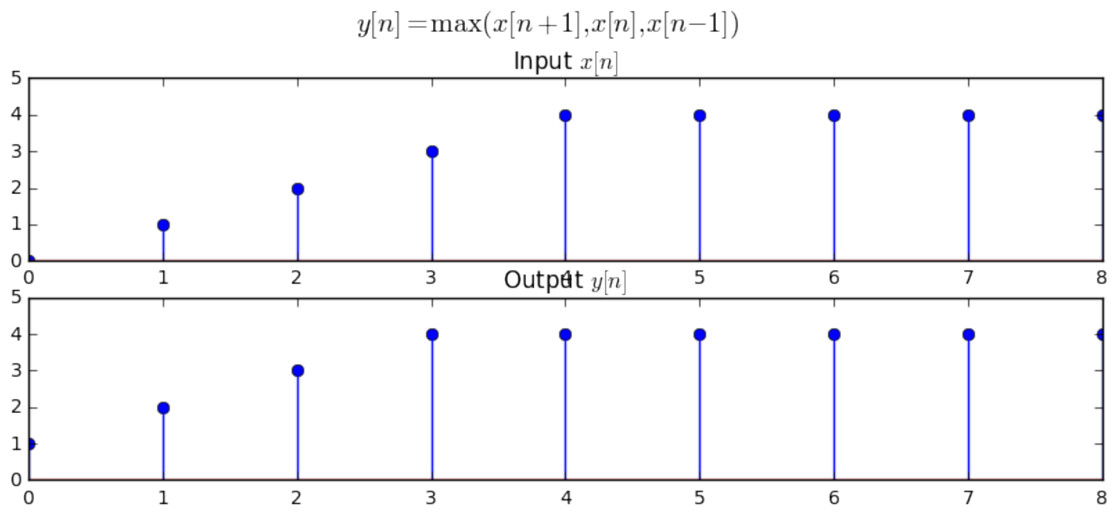
```
x =  [0, 1, 2, 3, 4, 4, 4, 4, 4]
y = [0.33, 1.00, 2.00, 3.00, 3.67, 4.00, 4.00, 4.00, 2.67]
```



```
In [52]: %matplotlib inline
         import matplotlib.pyplot as plt, numpy as np
         x = [0, 1, 2, 3, 4, 4, 4, 4, 4]
         for i in range(len(x)):
             if i==0:
                 y[i] = max(x[i], x[i+1])    # x[-1] does not exist in the vector
             elif i == len(x)-1:
                 y[i] = max(x[i-1], x[i])    # x[len(x)] does not exist in the vecto
             else:
                 y[i] = max(x[i-1], x[i], x[i+1])
```

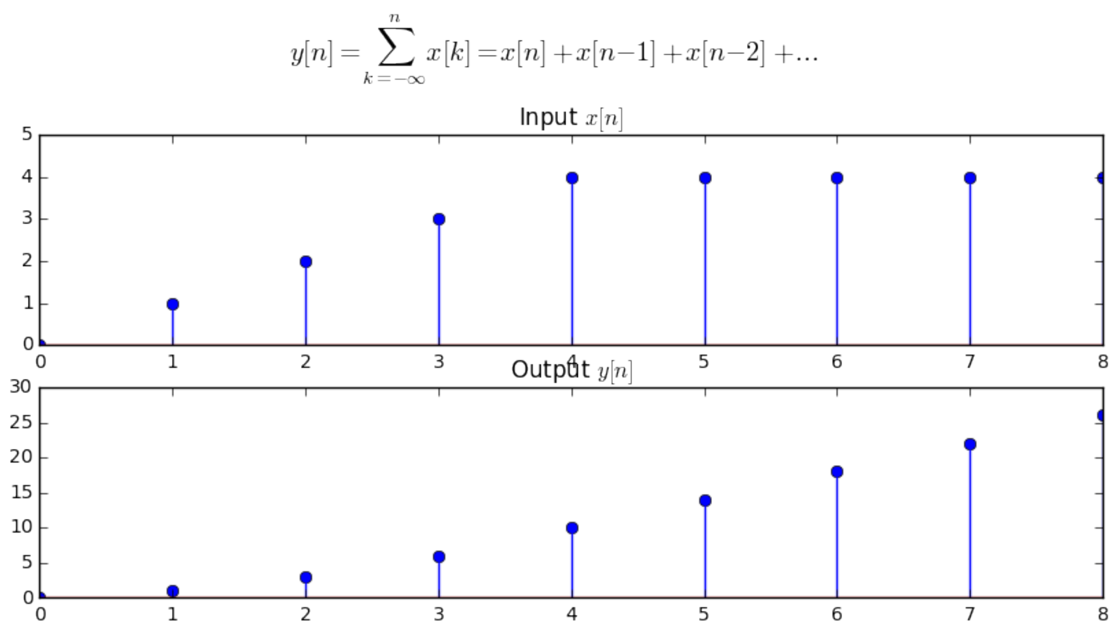
```
        print 'x = ',x
        print ('y = ['+', '.join(['%.2f']*len(y))%tuple(y)+']')
        plt.figure(figsize=(10,4));
        plt.subplot(2,1,1); plt.stem(x); plt.title ('Input $x[n]$'); plt.axis([0,8
        plt.subplot(2,1,2); plt.stem(y); plt.title ('Output $y[n]$'); plt.axis([0,
        plt.suptitle('$y[n] = \max(x[n+1], x[n], x[n-1])$', fontsize='x-large')
        plt.gcf().subplots_adjust(top=0.85)
```

```
x =  [0, 1, 2, 3, 4, 4, 4, 4, 4]
y = [1.00, 2.00, 3.00, 4.00, 4.00, 4.00, 4.00, 4.00, 4.00]
```



```
In [53]: %matplotlib inline
         import matplotlib.pyplot as plt, numpy as np
         x = [0, 1, 2, 3, 4, 4, 4, 4, 4]
         for i in range(len(x)):
             y[i] = sum(x[0:(i+1)])
         print 'x = ',x
         print ('y = ['+', '.join(['%.2f']*len(y))%tuple(y)+']')
         plt.figure(figsize=(10,5));
         plt.subplot(2,1,1); plt.stem(x); plt.title ('Input $x[n]$'); plt.axis([0,8
         plt.subplot(2,1,2); plt.stem(y); plt.title ('Output $y[n]$'); plt.axis([0,
         plt.suptitle('$y[n] = \sum_{k=-\infty}^{n} x[k] = x[n] + x[n-1] + x[n-2] +
         plt.gcf().subplots_adjust(top=0.80)
```

```
x =  [0, 1, 2, 3, 4, 4, 4, 4, 4]
y = [0.00, 1.00, 3.00, 6.00, 10.00, 14.00, 18.00, 22.00, 26.00]
```



1.4.1 Recursive systems

The last system $y[n] = \sum_{k=-\infty}^{n} x[k] = x[n] + x[n-1] + x[n-2] + \ldots$ can be also written in **recursive form**

$$y[n] = y[n-1] + x[n],$$

assuming that we start from a suitable **initial condition**

$$y[n_0] = \sum_{k=-\infty}^{n_0} x[k].$$

Recursive systems always have one or more initial conditions.

For recursive systems, the output signal depends on the input signal and on initial conditions as well. Thus, the initial conditions must always be specified for a recursive system.

If the initial conditions are omitted for a recursive system, it is implicitly assumed that they are equal to 0. When the initial conditions are all 0, the system is said to be **relaxed**.

Note that a recursive system with non-zero initial conditions can produce an output signal even in the absence of an input ($x[n] = 0$). See example.

1.4.2 Representation of systems

The operation of a system can be described graphically (see examples on blackboard):

- summation of two signals
- scaling of a signal with a constant
- multiplication of two signals

- delay element
- anticipation element
- other blocks for more complicated math operations

1.5 II.4 Classification of discrete systems

1.5.1 Memoryless / systems with memory

A system is **memoryless (or static)** if the output at some time n depends only on the input **from the same moment** n. Otherwise, the system **has memory (dynamic)**.

Examples:

- memoryless: $y[n] = (x[n])^3 + 5$
- with memory: $y[n] = (x[n])^3 + x[n-1]$

For systems with memory, if the output at time n $y[n]$ depends only the current input and on the last N inputs, $x[n-N], x[n-(N-1)], ...x[n]$, then the system has **memory N**. If N is finite, the system has **finite memory**; if $N = \infty$, the system has infinite memory.

Examples: - finite memory of order 4: $y[n] = x[n] + x[n-2] + x[n-4]$ - infinite memory: $y[n] = \sum_{k=-\infty}^{n} x[k] = x[n] + x[n-1] + x[n-2] + ...$

1.5.2 Time-Invariant and Time-Variant systems

A relaxed system H is **time-invariant** if and only if

$$x[n] \xrightarrow{H} y[n]$$

implies

$$x[n-k] \xrightarrow{H} y[n-k],$$

$\forall x[n], \forall k$.

Delaying the input signal with k will only delay the output with the same amount, otherwise the output is not affected. For all input signals, for all possible delays (positive or negative).

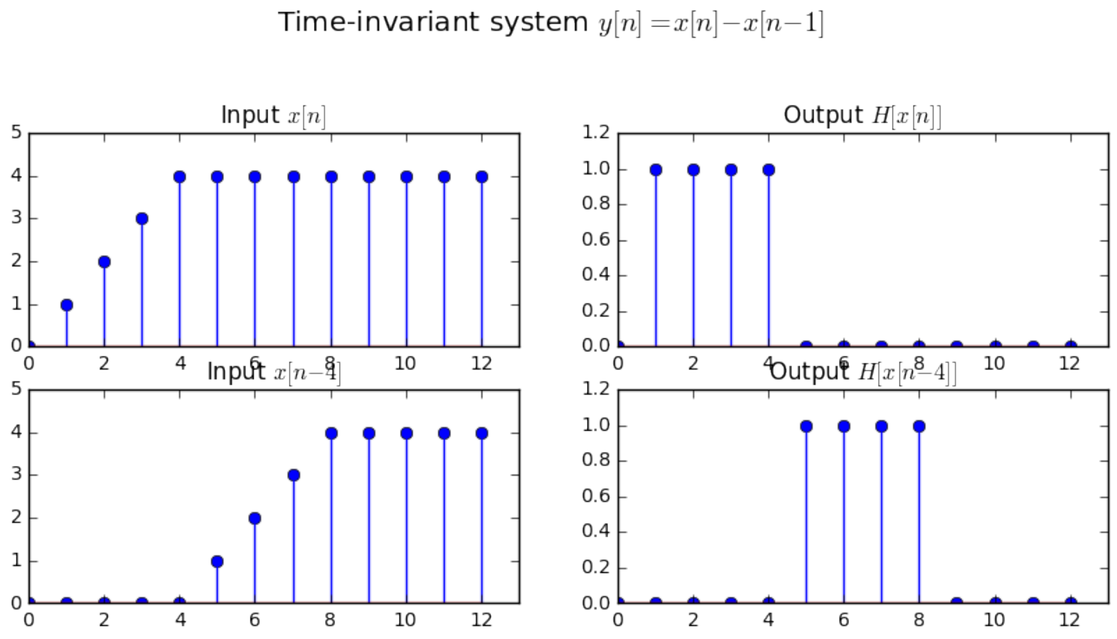
Otherwise, the system is said to be **time-variant**.

Examples: - $y[n] = x[n] - x[n-1]$ is time-invariant - $y[n] = n \cdot x[n]$ is not time-invariant

A system is time-invariant if it depends on n only through the input signal $x[n]$.

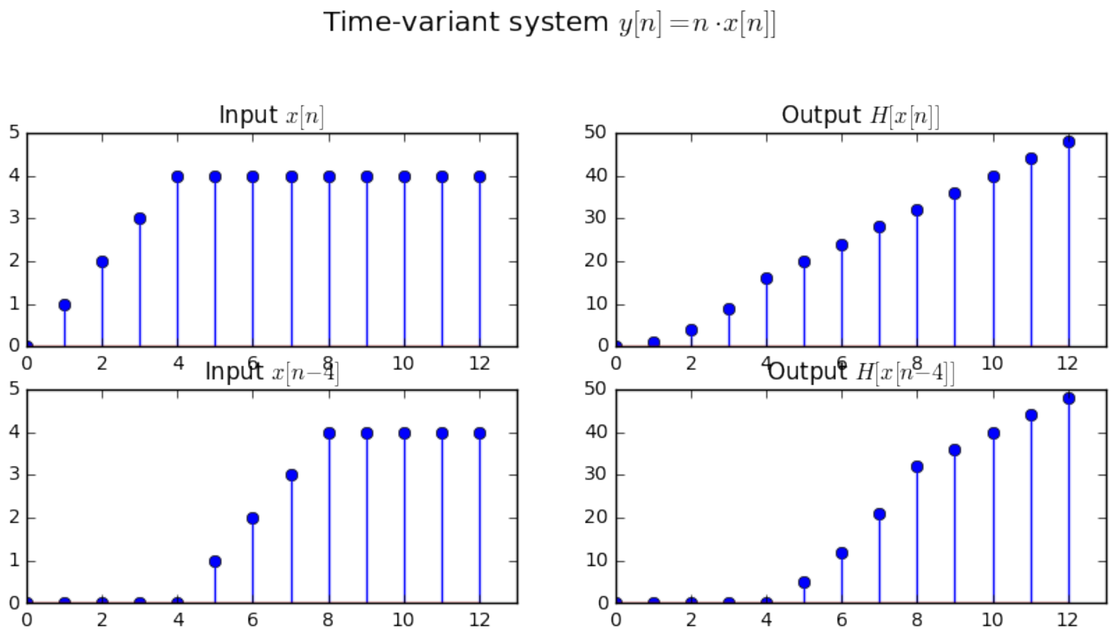
```
In [54]: %matplotlib inline
         import matplotlib.pyplot as plt, numpy as np
         n = range(0,13)
         x = [0, 1, 2, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4]
         xd = [0, 0, 0, 0, 0, 1, 2, 3, 4, 4, 4, 4, 4]
         def H(x):
           y = [0]*len(x)
           for i in range(len(x)):
               if i == 0:
                   y[i] = x[i]
               else:
                   y[i] = x[i] - x[i-1]
           return y
```

```
#print 'x = ',x
#print ('y = ['+', '.join(['%.2f']*len(y))%tuple(y)+']')
plt.figure(figsize=(10,5));
plt.subplot(2,2,1); plt.stem(n,x); plt.title ('Input $x[n]$'); plt.axis([[
plt.subplot(2,2,2); plt.stem(n,H(x)); plt.title ('Output $H[x[n]]$'); plt.
plt.subplot(2,2,3); plt.stem(n,xd); plt.title ('Input $x[n-4]$'); plt.axis
plt.subplot(2,2,4); plt.stem(n,H(xd)); plt.title ('Output $H[x[n-4]]$'); p
plt.suptitle('Time-invariant system $y[n] = x[n] - x[n-1]$', fontsize='x-l
plt.gcf().subplots_adjust(top=0.80)
```



```
In [55]: %matplotlib inline
         import matplotlib.pyplot as plt, numpy as np
         n = range(0,13)
         x = [0, 1, 2, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4]
         xd = [0, 0, 0, 0, 0, 1, 2, 3, 4, 4, 4, 4, 4]
         def H(n,x):
           y = [n[i] * x[i] for i in range(len(x))]
           return y
         print 'H[x[n]] = ',H(n,x)
         print 'H[x[n-4]] = ',H(n,xd)
         plt.figure(figsize=(10,5));
         plt.subplot(2,2,1); plt.stem(n,x); plt.title ('Input $x[n]$'); plt.axis([[
         plt.subplot(2,2,2); plt.stem(n,H(n,x)); plt.title ('Output $H[x[n]]$'); pl
         plt.subplot(2,2,3); plt.stem(n,xd); plt.title ('Input $x[n-4]$'); plt.axis
         plt.subplot(2,2,4); plt.stem(n,H(n,xd)); plt.title ('Output $H[x[n-4]]$');
         plt.suptitle('Time-variant system $y[n] = n \cdot x[n]$', fontsize='x-lar
         plt.gcf().subplots_adjust(top=0.80)
```

```
H[x[n]] =  [0, 1, 4, 9, 16, 20, 24, 28, 32, 36, 40, 44, 48]
H[x[n-4]] =  [0, 0, 0, 0, 0, 5, 12, 21, 32, 36, 40, 44, 48]
```



1.5.3 Linear and nonlinear systems

A system H is **linear** if it satisfies the following relation (called "superposition")

$$H\left[ax_1[n] + bx_2[n]\right] = aH\left[x_1[n]\right] + bH\left[x_2[n]\right].$$

Applying the system to a sum of two signals = applying the system to each signal, and adding the results.

Scaling the input signal with a constant a **is the same as scaling the output signal with** a.

The same relation will be true for a sum of many signals, not just two.

When considering a complicated input signal, the signal can be decomposed into a sum of smaller parts, and the system can be applied to each part independently, then the results are added.

Examples: - linear system: $y[n] = 3x[n] + 5x[n-2]$ - nonlinear system: $y[n] = 3(x[n])^2 + 5x[n-2]$

For a syste to be linear, the input samples $x[n]$ must not undergo non-linear transformations.

The only transformations of the input x[n] allowed to take place in a linear system are:

- scaling (multiplication) with a constant
- delaying
- summing different delayed versions of the signal (not summing with a constant)

Examples: at blackboard

1.5.4 Causal and non-causal systems

A system is **causal** if the output $y[n]$ depends only on the current input $x[n]$ and the past values $x[n-1], x[n-2]\dots$, but not on the future samples $x[n+1], x[n+2]\dots$

Otherwise the system is **non-causal**.

A causal system can function in real-time, because to compute the current output sample we need only the input samples from the past.

Examples:

- $y[n] = x[n] - x[n-1]$ is causal
- $y[n] = x[n+1] - x[n-1]$ is non-causal
- $y[n] = x[-n]$ is non-causal

1.5.5 Stable and unstable systems

A signal is **bounded** if there exists a value M such that all the samples of the signal or smaller than M, in absolute values.

$$x[n] \in [-M, M]$$

$$|x[n]| \leq M$$

A system is **stable** if for any bounded input signal it produces a bounded output signal (not necessarily with the same M). This known as BIBO (Bounded Input –> Bounded Output).

In other words, when the input signal has bounded values, the output signal does not go towards ∞ or $-\infty$.

Examples: - $y[n] = (x[n])^3 - x[n+4]$ is stable - $y[n] = \frac{1}{x[n]-x[n-1]}$ is unstable - $y[n] = \sum_{k=-\infty}^{n} x[k] = x[n] + x[n-1] + x[n-2] + ...$ is unstable

1.6 Impulse response of Linear Time-Invariant (LTI) systems

Notation: An **LTI** system (**L**inear **T**ime-**I**nvariant) is a system which is simultaneously **linear** and **time-invariant**.

LTI systems can be described via either (or both):

1. the **impulse response** $h[n]$
2. the **difference equation**

$$\begin{aligned} y[n] &= -\sum_{k=1}^{N} a_k y[n-k] + -\sum_{k=1}^{M} b_k x[n-k] \\ &= -a_1 y[n-1] - a_2 y[n-2] - ... - a_N y[n-N] + b_0 x[n] + b_1 x[n-1] + ... + b_M x[n-M] \end{aligned}$$

1.6.1 The impulse response

The **impulse response** of a system is the output (response) of the system when the input signal is the impulse signal $\delta[n]$:

$$h[n] = H(\delta[n]).$$

The impulse response **fully characterizes the system**: based on $h[n]$ we can compute the response of the system to **any** input signal. All the properties of LTI systems can be described via characteristics of the impulse response

1.6.2 Signals are a sum of impulses

Every discrete signal is composed of impulses, so any signal can be composed as **a sum of scaled and delayed impulses** $\delta[n]$.

Example: $x[n] = 3, 1, -5, 0, 2 = 3\delta[n] + \delta[n-1] - 5\delta[n-2] + 2\delta[n-2]$

In general

$$x[n] = \sum_{k=-\infty}^{\infty} x[k]\delta[n-k],$$

i.e. a sum of impulses $\delta[n]$, delayed with k and scaled with the corresponding value $x[k]$.

1.6.3 Convolution

Because the system is linear and time-invariant, the response of the system to a sum of impulses, delayed with k and scaled with $x[k]$, **is a sum of impulse responses, delayed with k and scaled with $x[k]$**.

Basically, the input signal is composed of a "bunch" of impulses. Since the system is LTI, each impulse will generate its own response. The output signal is the sum of impulse responses, delayed and scaled appropriately.

$$\begin{aligned} y[n] =& H\left(x[n]\right) \\ =& H\left(\sum_{k=-\infty}^{\infty} x[k]\delta[n-k]\right) \\ =& \sum_{k=-\infty}^{\infty} x[k]H\left(\delta[n-k]\right) \\ =& \sum_{k=-\infty}^{\infty} x[k]h[n-k]. \end{aligned}$$

This operation is known as the **convolution** of two signals $x[n]$ and $h[n]$

$$x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

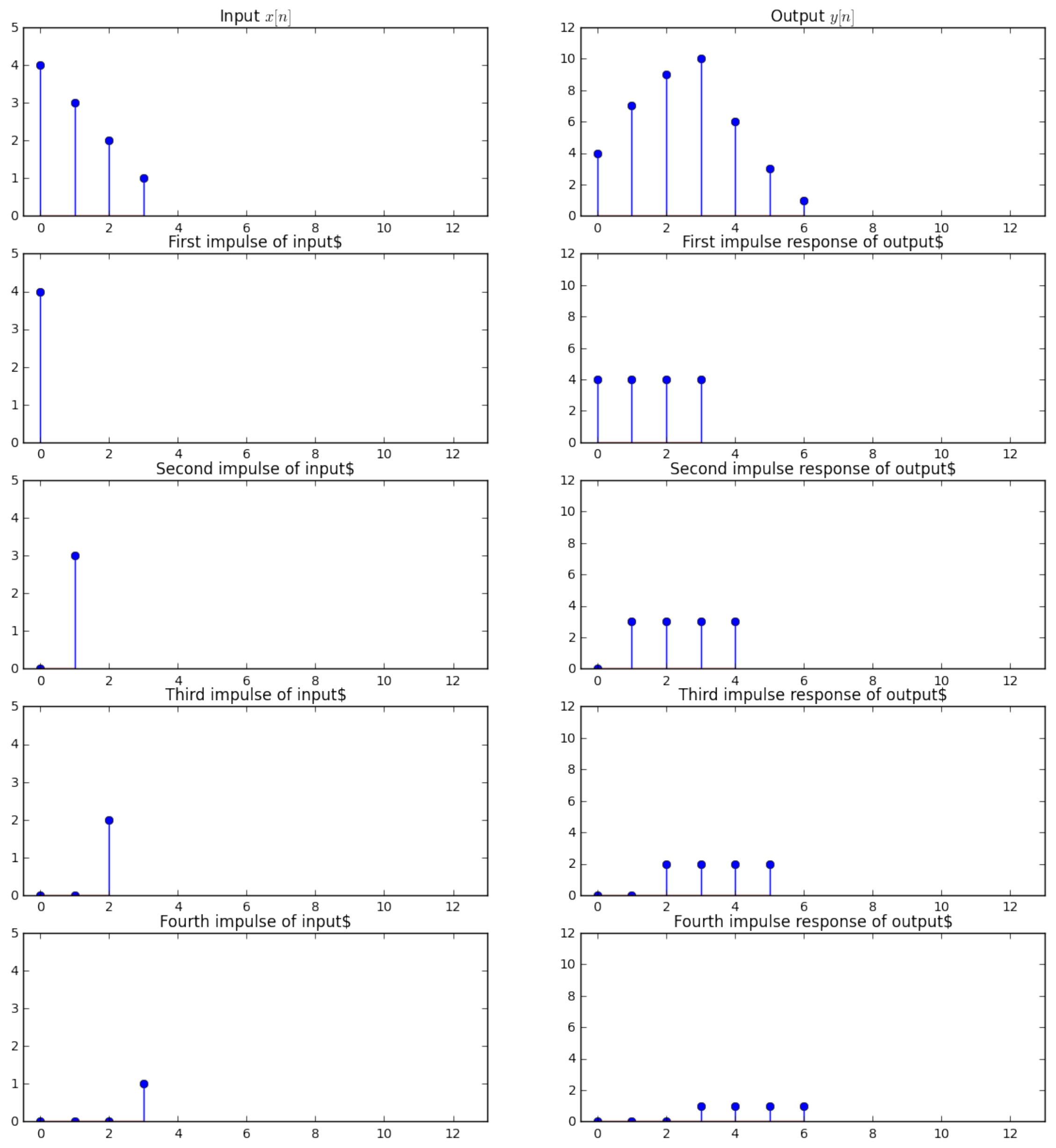
.

The response of a LTI system to an input signal x[n] is the convolution of x[n] with the system's impulse response h[n].

```
In [56]: %matplotlib inline
         import matplotlib.pyplot as plt, numpy as np
         n = range(0,3)
         x = [4, 3, 2, 1]
         h = [1, 1, 1, 1]
```

```
y = np.convolve(x,h)
x0 = [4]
x1 = [0, 3]
x2 = [0, 0, 2]
x3 = [0, 0, 0, 1]
y0 = x[0]*np.array(h)
y1 = x[1]*np.array([0] + h)
y2 = x[2]*np.array([0, 0] + h)
y3 = x[3]*np.array([0, 0, 0] + h)
plt.figure(figsize=(14,15));
plt.subplot(5,2,1); plt.stem(x); plt.title ('Input $x[n]$'); plt.axis([-0.
plt.subplot(5,2,2); plt.stem(y); plt.title ('Output $y[n]$'); plt.axis([-(
plt.subplot(5,2,3); plt.stem(x0); plt.title ('First impulse of input$'); p
plt.subplot(5,2,4); plt.stem(y0); plt.title ('First impulse response of ou
plt.subplot(5,2,5); plt.stem(x1); plt.title ('Second impulse of input$');
plt.subplot(5,2,6); plt.stem(y1); plt.title ('Second impulse response of c
plt.subplot(5,2,7); plt.stem(x2); plt.title ('Third impulse of input$'); p
plt.subplot(5,2,8); plt.stem(y2); plt.title ('Third impulse response of ou
plt.subplot(5,2,9); plt.stem(x3); plt.title ('Fourth impulse of input$');
plt.subplot(5,2,10); plt.stem(y3); plt.title ('Fourth impulse response of
#plt.gcf().subplots_adjust(top=0.80)
```

```
Out[56]: [-0.5, 13, 0, 12]
```



1.6.4 Properties of convolution

- Convolution is commutative (the order of the two signals doesn't matter):

$$x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] = h[n] * x[n] = \sum_{k=-\infty}^{\infty} h[k]x[n-k]$$

.

Proof: make variable change $(n-k) \to l$, change all in equation

- Convolution is associative

$$(a[n] * b[n]) * c[n] = a[n] * (b[n] * c[n])$$

.

(No proof)

- The unit impulse is neutral element for convolution

$$a[n] * \delta[n] = \delta[n] * a[n] = a[n]$$

.

Proof: equation

- Convolution is a linear operation

$$(\alpha \cdot a[n] + \beta \cdot b[n]) * c[n] = \alpha(\cdot a[n] * c[n]) + \beta \cdot (b[n] * c[n])$$

Proof: by linearity of the corresponding system

1.6.5 Properties of LTI systems expressed with $h[n]$

1. Identity system A system with $h[n] = \delta[n]$ produces an response equal to the input, $y[n] = x[n], \forall x[n]$.

Proof: $\delta[n]$ is neutral element for convolution.

2. Series connection is commutative LTI systems connected in series can be interchanged in any order. Proof: by commutativity of convolution.

LTI systems connected in series are equivalent to a single system with

$$h_{equiv}[n] = h_1[n] * h_2[n] * ... * h_N[n]$$

3. Parallel connection means sum LTI systems connected in parallel are equivalent to a single system with

$$h_{equiv}[n] = h_1[n] + h_2[n] + ... * h_N[n]$$

4. Response of LTI systems to unit step If the input signal is $u[n]$, the response of the system is

$$s[n] = u[n] * h[n] = h[n] * u[n] = \sum_{k=-\infty}^{\infty} h[k]u[n-k] = \sum_{k=-\infty}^{n} h[k]$$

.

The signal $\sum_{k=-\infty}^{n} h[k]$ is a *discrete-time integration* of $h[n]$ (equivalent to the integral for continuous signals $\int_{-\infty}^{t} h(t)dt$, only that the signal is now discrete si the integral degenerates into a sum).

It follows that

$$h[n] = s[n] - s[n-1].$$

Note that the unit step $u[n]$ iteslf is the discrete-time integral of the unit impulse:

$$u[n] = \sum_{k=-\infty}^{n} \delta[k]$$

$$\delta[n] = u[n] - u[n-1]$$

Therefore the system response to the integral of the unit impulse is the integral of the system response to the input impulse.

The interchanging of the integration with the system is due to the linearity of the system and is valid for all signals:

$$H\left(\sum_{k=-\infty}^{n} x[k]\right) = \sum_{k=-\infty}^{n} H\left(x[k]\right)$$

1.6.6 Relation between LTI system properties and $h[n]$

1. Causal LTI systems and their $h[n]$ If a LTI system is causal, then $h[n] = 0, \forall n < 0$.

Proof: If $y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$, but $y[n]$ does not depend on $x[n+1], x[n+2], ...$, it means that these terms are multiplied with 0. The value $x[n+1]$ is multiplied with $h[n-(n+1)] = h[-1]$, $x[n+2]$ is multiplied with $h[n-(n+2)] = h[-2]$, and so on. Therefore:

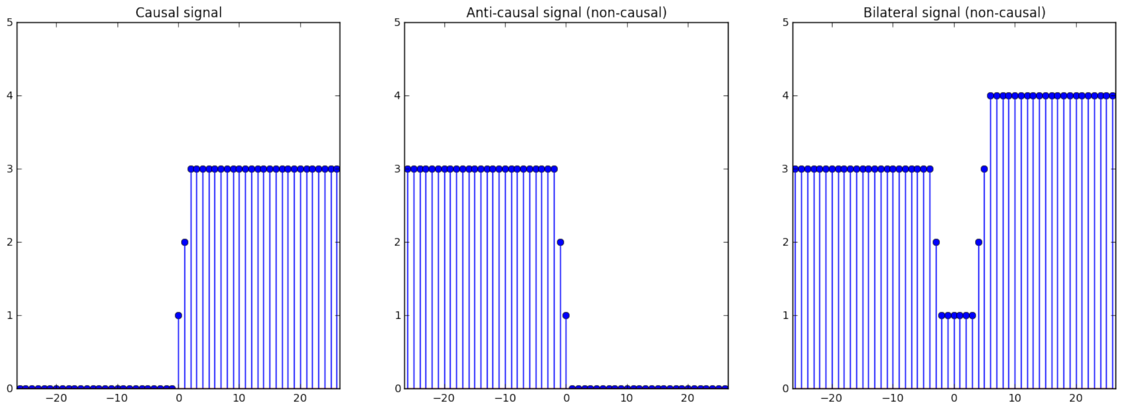
$$h[n] = 0, \forall n < 0$$

.

Such a signal which is 0 for $n < 0$ is called a *causal* ***signal***. Otherwise the signal is *non-causal*.
Therefore we can say that *a system is causal if and only if it has a causal impulse response*
Further definitions:

- a signal which 0 for $n > 0$ is called an *anti-causal* signal
- a signal which has non-zero values both for some $n > 0$ and for some $n < 0$ (and thus is neither causal nor non-causal) is called *bilateral*.

```
In [57]: %matplotlib inline
         import matplotlib.pyplot as plt, numpy as np
         l = 20
         n = range(-(l+6),(l+7))
         xc = [0]*l + [0, 0, 0, 0, 0, 0, 1, 2, 3, 3, 3, 3, 3] + [3]*l
         xn = [3]*l + [3, 3, 3, 3, 3, 2, 1, 0, 0, 0, 0, 0, 0] + [0]*l
         xb = [3]*l + [3, 3, 3, 2, 1, 1, 1, 1, 1, 1, 2, 3, 4] + [4]*l
         plt.figure(figsize=(18,6));
         plt.subplot(1,3,1); plt.stem(n,xc); plt.title ('Causal signal'); plt.axis
         plt.subplot(1,3,2); plt.stem(n,xn); plt.title ('Anti-causal signal (non-ca
         plt.subplot(1,3,3); plt.stem(n,xb); plt.title ('Bilateral signal (non-caus
```



2. Stable systems and their $h[n]$ Considering a bounded input signal, $|x[n]| \leq A$, the absolute value of the output is:

$$
\begin{aligned}
|y[n]| =& |\sum_{k=-\infty}^{\infty} x[k]h[n-k]| \\
\leq& \sum_{k=-\infty}^{\infty} |x[k]h[n-k]| \\
=& \sum_{k=-\infty}^{\infty} |x[k]||h[n-k]| \\
\leq& A \sum_{k=-\infty}^{\infty} |h[n-k]|
\end{aligned}
$$

The output signal is bounded (and hence the system is stable) if $\sum_{k=-\infty}^{\infty}|h[n]|$ is finite.
Therefore **a LTI system is stable if**

$$\sum_{k=-\infty}^{\infty} |h[n]| < \infty$$

.

1.6.7 3. Memoryless systems and their $h[n]$ (Exercise)

Exercise: What ca we say about the impulse response $h[n]$ of a memoryless system? What about a system with finite memory M?

1.6.8 FIR and IIR systems

The *support* of a discrete signal is the smallest interval of n such that the signal is 0 everywhere outside the interval.

Depending on the support of the impulse response, discrete LTI systems can be FIR or IIR systems.

FIR systems A **F**inite **I**mpulse **R**esponse (**FIR**) system has an impulse response with finite support, i.e. the impulse response is 0 outside a certain interval.

For a causal system, $h[n] = 0$ for $n < 0$, so we have $h[n] = 0$ for $n < 0$ or $n \geq M$, for some M. The convolution is thus reduced to:

$$y[n] = \sum_{k=0}^{M} h[k]x[n-k] = h[0] \cdot x[n] + h[1] \cdot x[n-1] + ...h[M] \cdot x[n-M].$$

For a causal FIR system, the output is a linear combination of the last M input samples (has finite memory M).

IIR systems An **I**nfinite **I**mpulse **R**esponse (**FIR**) system has an impulse response with infinite support, i.e. the impulse response never becomes completely 0 forever.

In the case of a causal system, from the convolution equation results that the output $y[n]$ potentially depends on all the preceding input samples. An IIR system has infinite memory.

1.6.9 Recursive and non-recursive implementation of systems

A ***recursive*** implementation of a system computes the output $y[n]$ based partly on the previous output samples $y[n-1], y[n-2], ...$

Generally, every LTI system can be expressed only based on the input samples $x[n], x[n-1], ...$, but in some cases it would need an infinite amount of memory (an infinite amount of previous input samples). Therefore the recursive expression is preferred, when possible.

Example:

$$y[n] = \frac{1}{n+1} \sum_{0}^{n} x[n]$$

This system (what is its memory?) can be rewritten in recursive form:

$$y[n] = n \cdot y[n-1] + x[n]$$

In general, the output $y[n]$ of a recursive system depends on the last N samples of the output, y[n-1], ... y[n-N], and the current and the last M samples of the input, x[0], x[1], ... x[n-M].

A ***non-recursive*** system is a system **for which the output $y[n]$ depends only on the the current and the last M samples of the input, x[0], x[1], ... x[n-M].**

FIR systems can always be implemented non-recursively, but it is possible to implemented them in a recursive way.

IIR systems can only be implemented recursively.

1.6.10 Initial conditions for recursive systems

Recursive systems rely on previous outputs. When the computations of the output starts, the previous values must be specified. These previous output values needed at the start moment are *the initial conditions* of the system.

Notes

- The output of a system always depends on the initial conditions, besides the input signal
- A system with initial conditions equal to 0 is called *relaxed*

- The output of a relaxed system to an input signal is called *zero-state response*, $y_{zs}[n]$, because the initial conditions (initial state) is 0, and the output depends only in the input signal. This is also called *forced response*.
- A system with non-zero initial conditions produces an output even when the input signal is zero. This output is called *zero-input response*, $y_{zi}[n]$, because the input signal is 0, and the output depends only on the initial conditions. This is also called *natural response*.
- For linear systems, the output of a system is always the sum of the forced response and the natural response:
$$y[n] = y_{zs}[n] + y_{zi}[n]$$

1.6.11 Correlation of discrete signals

The **correlation** (or cross-correlation) of two finite-energy discrete signals $x[n]$ and $y[n]$ is defined as:

$$r_{xy}[l] = \sum_{\infty}^{\infty} x[n]y[n-l]$$

$$r_{xy}[l] = \sum_{\infty}^{\infty} x[n+l]y[n]$$

This equation is similar to convolution, up to a reflection of the signal (the second term has $y[n-l]$ instead of $y[l-n]$). Thus correlation = convolution with a reflected signal, and vice-versa.

The argument l represents the delay of $y[n]$ compared to $x[n]$

The **auto-correlation** of a signal $x[n]$ is defined as the correlation of the signal with itself

$$r_{xx}[l] = \sum_{\infty}^{\infty} x[n]x[n-l] = \sum_{\infty}^{\infty} x[n]x[n+l]$$

1.6.12 Properties of correlation and auto-correlation

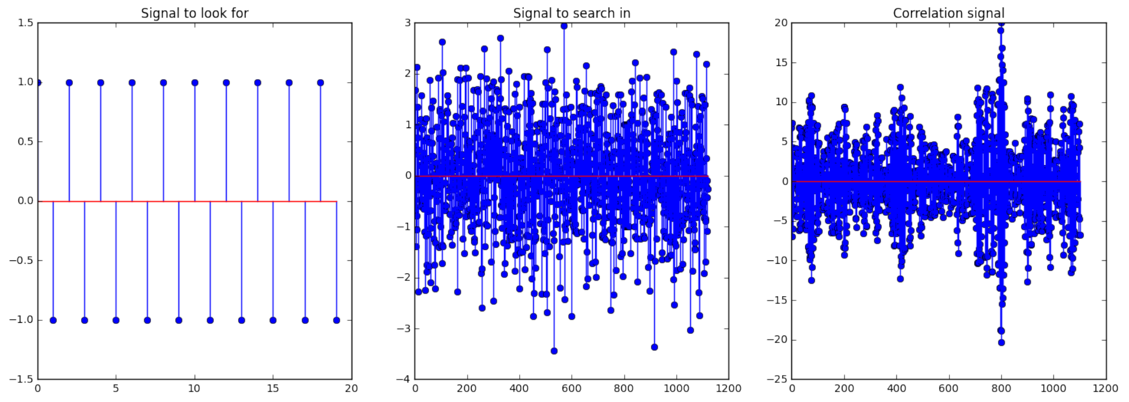
1. $r_{xy}[l] = r_{yx}[-l]$ Proof: at whiteboard
2. Auto-correlation is an **even** signal:
$$r_{xx}[l] = r_{xx}[-l]$$
Proof: based on first property
3. Auto-correlation is maximum for $l = 0$
$$r_{xx}[0] \geq r_{xx}[l], \forall l$$

1.6.13 Usages of correlation and auto-correlation

Besides significant theoretical importance in theory, there are some straightforward applications of correlation in practice. A few examples are given below

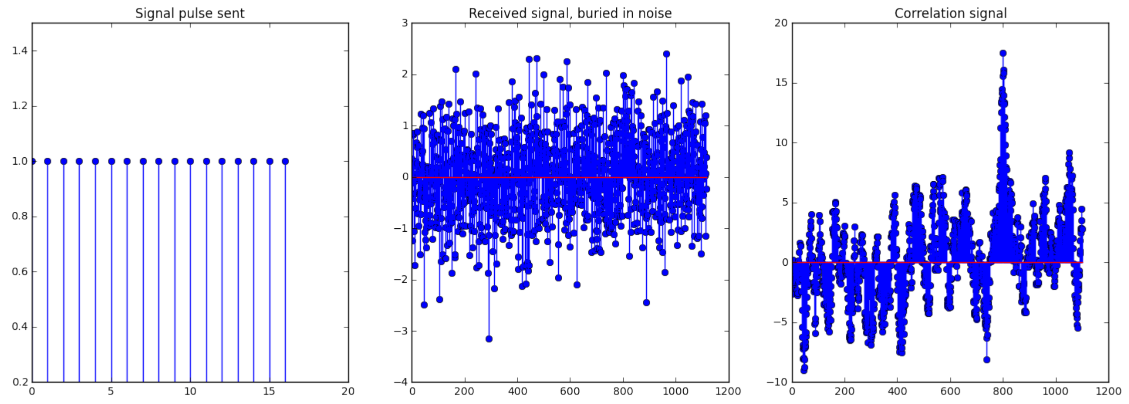
1. Searching for a certain part in a large signal When the two signals have both positive and negative values (roughly of similar length), the correlation signal will have a large value when the positive and negative areas match, and small values when they don't match. This can be used to locate a certain part within a large signal

```
In [72]: %matplotlib inline
         import matplotlib.pyplot as plt, numpy as np
         x1 = np.array([1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1,
         x2 = np.hstack((np.random.randn(800), x1, np.random.randn(300)))
         corr = np.correlate(x2, x1)
         plt.figure(figsize=(18,6));
         plt.subplot(1,3,1); plt.stem(x1); plt.title ('Signal to look for');plt.axi
         plt.subplot(1,3,2); plt.stem(x2); plt.title ('Signal to search in');
         plt.subplot(1,3,3); plt.stem(corr); plt.title ('Correlation signal');
```



2. Estimating the delay of a signal In radar-like systems, a signal pulse is sent from an emitter, gets reflected from a target and is received back with a lot of random noise added. We would like to estimate the delay of the received pulse. The correlation signal between the original block and the received signal will have a maximum when the original block overlaps with the block in the received signal. The position of the maximum indicates the delay.

```
In [79]: %matplotlib inline
         import matplotlib.pyplot as plt, numpy as np
         x1 = np.array([1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1])
         x2 = np.hstack((np.zeros(800), x1, np.zeros(300))) + 0.8*np.random.randn(8
         corr = np.correlate(x2, x1)
         plt.figure(figsize=(18,6));
         plt.subplot(1,3,1); plt.stem(x1); plt.title ('Signal pulse sent');plt.axis
         plt.subplot(1,3,2); plt.stem(x2); plt.title ('Received signal, buried in r
         plt.subplot(1,3,3); plt.stem(corr); plt.title ('Correlation signal');
```


Signal pulse sent
Received signal, buried in noise
Correlation signal