

Project #3

Transformations of the Independent Variable

In this project, we will see how to implement a class of simple signal transformations that are based on the modification of the independent variable, which is mostly the “time” in the context of signals and systems.

Part 1: Continuous Time Case

In order to see the effects of transformations on the independent variable, we will first generate a test signal that we will play with. Our test signal will be as shown in Figure 3.1 (or Figure 3.2). As our signal has different linear segments, we can generate those parts one by one and then combine them to obtain the whole signal.

We will generate the segments of time axis first and then use *linspace*¹ commands to create the corresponding signal segments. Linspace command is useful generating linearly changing or equally spaced points between two limits. The two limits can be the same, in which case the linspace command produces vectors filled with that same limit or constant.

The 1st segment of the signal goes from $t = -2$ to 0.

```
» t1=-2:0.05:0; MCL 1
```

We have just generated a time vector of length 41. Now, we will create the corresponding signal segment.

```
» x1=linspace(0,0,41); MCL 2
```

It is better to use Matlab’s capabilities wherever possible. One little service that we can get from Matlab is to ask for the length of vectors via the *length* command. If we utilize this at MCL 2, it would be as follows.

¹ `linspace(x1,x2,N)` generates a row vector of N linearly or equally spaced points between x1 and x2.

```
» x1=linspace(0,0,length(t1));
```

Instead of using MCL 2, we could have also generated x_1 , the 1st signal segment, by the following method that utilizes Matlab's *zeros*² utility.

```
» x1=zeros(1,length(t1));
```

We can now generate the other segments in a similar way.

```
» t2=0.05:0.05:1;                               MCL 3
» x2=linspace(1,1,length(t2));                   MCL 4
» t3=1.05:0.05:2;                               MCL 5
» x3=linspace(1,0,length(t3));                   MCL 6
» t4=2.05:0.05:3;                               MCL 7
» x4=linspace(0,0,length(t4));                   MCL 8
```

Note how we selected the limits of consecutive time segments above.

We can now combine both time and signal segments into single vectors and do a plot of our signal.

```
» t=[t1 t2 t3 t4];                               MCL 9
» x=[x1 x2 x3 x4];                               MCL 10
» plot(t,x);                                     MCL 11
```

We observe in Figure 3.1 that Matlab's automatic axis scaling did not produce a good-looking plot. We can override Matlab's default axes scales by using the *axis*³ command.

```
» axis([-2 3 -0.5 1.5])                          MCL 12
```

This is how we obtained Figure 3.2, plot of our signal with a different axis scaling.

² A close relative of the *zeros* utility is the *ones* utility, which can be used to fill up matrices with ones.

³ In its most commonly used form, the *axis* command takes the vector $[x_{min} \ x_{max} \ y_{min} \ y_{max}]$ as its argument to set the x and y-axis scales.

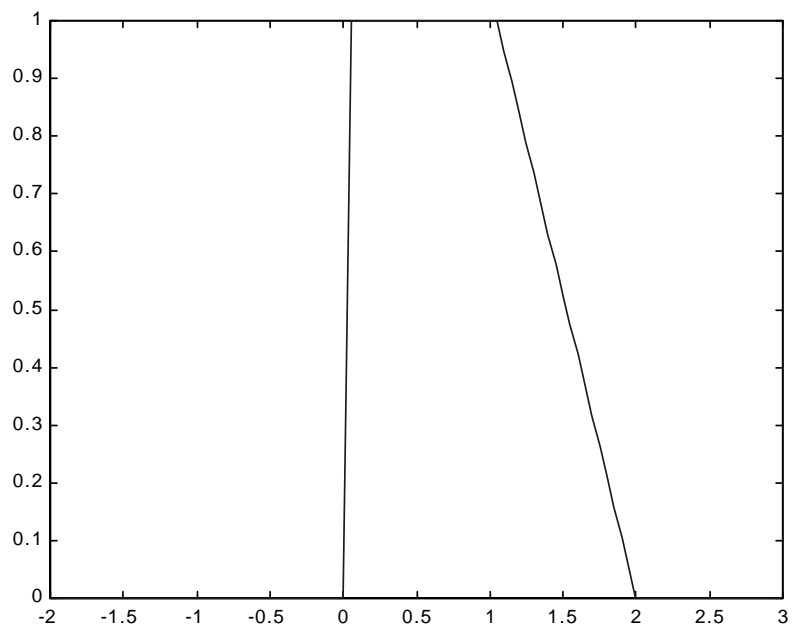


Figure 3.1 The test signal that will be used in transformation examples.

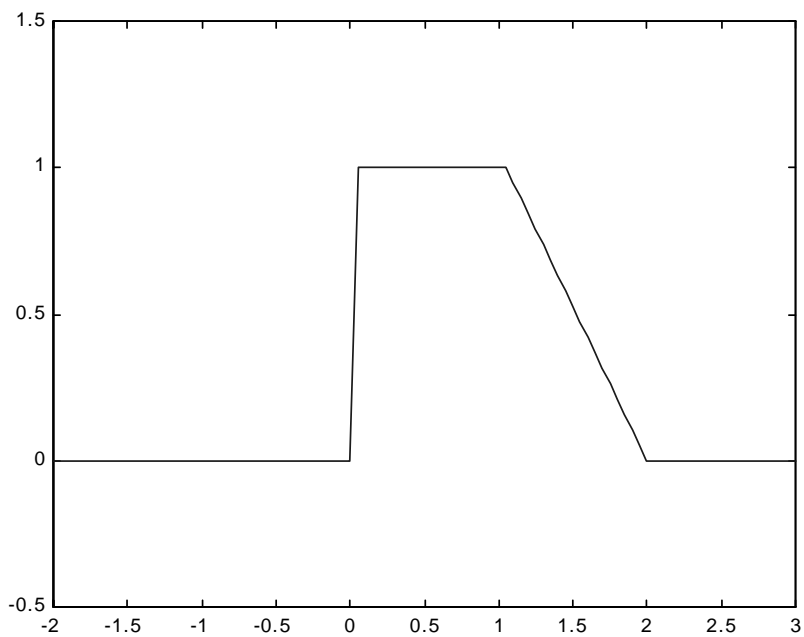


Figure 3.2 The test signal with better axes scaling.

We spent a big effort to create our test signal but the remaining part is relatively easy. Let us see the effect of *time shift* first. For instance, let us try to find out how $x(t+1)$ versus t will look. We know that $x(t+1)$ should look like $x(t)$ shifted towards left by 1 unit in time. You may think that we simply can do this by the following line.

```
» plot(t+1,x)
```

But this is not true! Look at the result of this misleading idea at Figure 3.3, which we produced by the following lines.

```
» subplot(2,1,1);plot(t,x);axis([-3 4 -0.5 1.5])      MCL 13
```

```
» subplot(2,1,2);plot(t+1,x);axis([-3 4 -0.5 1.5])  MCL 14
```

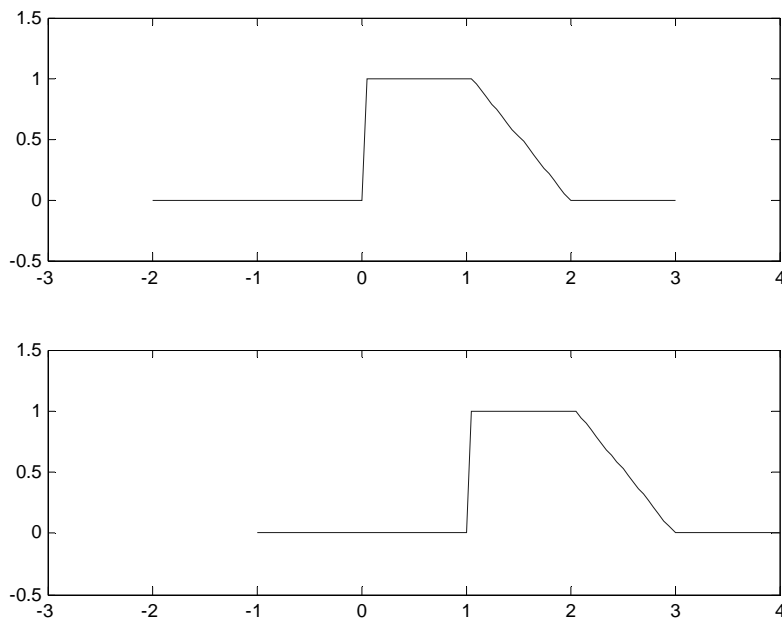


Figure 3.3 Plot of $x(t)$ versus t (upper panel) and erroneous plot of $x(t)$ versus $t+1$ (lower panel).

Above, we utilized the *subplot* command that enabled us to divide the figure window into several parts where we can examine different variables or plots simultaneously. To return back to the default single axes (or plot) view on the figure window, we can either issue `subplot(1,1,1)` or `clf` (clear figure) commands.

Coming back to the real issue, we realize that what we did is actually the plot of $x(t)$ versus $t+1$, but we were after $x(t+1)$ versus t . How can we accomplish this, i.e. do the plot of $x(t+1)$ versus t ? Let $y(t) = x(t+1)$ and $u = t+1$; then

$$t = u - 1 \text{ and } y(u - 1) = x(u).$$

As u is just a dummy variable, we can replace it with t and obtain

$$y(t - 1) = x(t).$$

This indicates that our new function $y(t) = x(t+1)$ takes the value $x(t)$ at $t-1$. Therefore, if we plot our original function $x(t)$ with respect to a new time variable or vector that we will form as $t-1$, we will get $x(t+1)$ versus t . Let us realize this idea now.

```

» subplot(2,1,1);plot(t,x);axis([-3 4 -0.5 1.5])      MCL 15
» xlabel('t');ylabel('x(t)')                          MCL 16
» title('x(t) versus t')                              MCL 17
» subplot(2,1,2);plot(t-1,x);axis([-3 4 -0.5 1.5])   MCL 18
» xlabel('t');ylabel('x(t+1)')                       MCL 19
» title('x(t+1) versus t')                            MCL 20

```

We again used subplot commands to see the plots of $x(t)$ and $x(t+1)$ at the same time, i.e. in the same figure window. The result is shown in Figure 3.4 and as we expected $x(t+1)$ is same as $x(t)$ shifted towards left by 1 unit in time. Here, we also used *xlabel* and *title* commands to make our plots more informative.

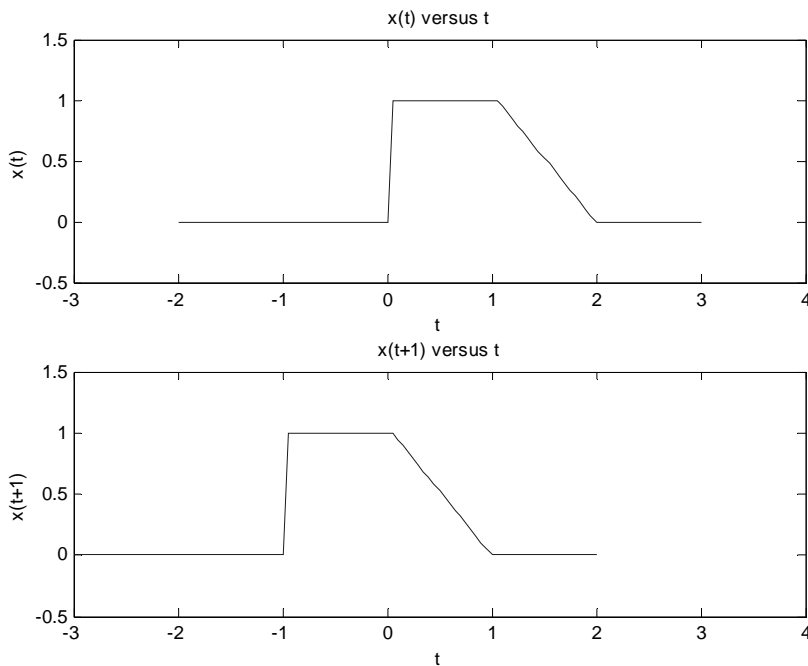


Figure 3.4 Plot of $x(t)$ versus t (upper panel) and $x(t+1)$ versus t (lower panel).

Let us now make the plot of $x(-2t)$ versus t to study the effects of *time reversal* and *time scaling*. We expect $x(-2t)$ to be a time reversed and compressed version of $x(t)$. To verify this, let $y(t) = x(-2t)$ and $u = -2t$; then

$$t = -u/2 \text{ or } y(-u/2) = x(u).$$

We replace u by t and obtain $y(-t/2) = x(t)$. Hence, $y(t) = x(-2t)$ takes the value $x(t)$ at $-t/2$. In other words, plot of $x(-2t)$ versus t is equivalent to the plot of $x(t)$ versus $-t/2$. The following lines will plot $x(t)$ and $x(-2t)$. The result is shown in Figure 3.5.

```

» subplot(2,1,1);plot(t,x);axis([-3 4 -0.5 1.5])      MCL 21
» xlabel('t');ylabel('x(t)')                          MCL 22
» title('x(t) versus t')                              MCL 23
» subplot(2,1,2);plot(-t/2,x);axis([-3 4 -0.5 1.5]) MCL 24
» xlabel('t');ylabel('x(-2t)')                       MCL 25
» title('x(-2t) versus t')                            MCL 26

```

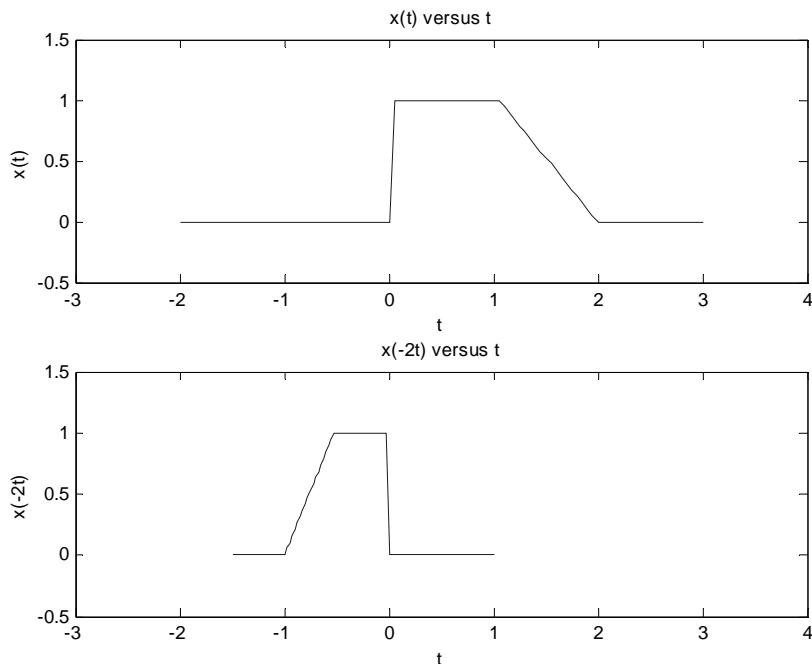


Figure 3.5 Plot of $x(t)$ versus t (upper panel) and $x(-2t)$ versus t (lower panel).

As another exercise, let us plot $x((t/3)-1)$ versus t . This example involves time scaling and time shifting. We will use the same approach again, where we find the time at which the new or transformed signal $y(t)$ takes the value of the original signal $x(t)$. This way, we keep the original signal the same and plot it with respect to a transformed time variable or vector, to find the transformed signal.

Let $y(t) = x((t/3)-1)$ and $u = (t/3)-1$; then $t = 3(u+1)$ and $y(3(u+1)) = x(u)$. We replace u by t and obtain $y(3(t+1)) = x(t)$. We can now complete our task by the following lines. The result is shown in Figure 3.6 and as we see $x((t/3)-1)$ is expanded/stretched and right shifted (by 3 units in time) version of $x(t)$.

```
» subplot(2,1,1);plot(t,x);axis([-4 14 -0.5 1.5])    MCL 27
» xlabel('t');ylabel('x(t)')                        MCL 28
```

```

» title('x(t) versus t')                                MCL 29
» tnew=3*(t+1);                                         MCL 30
» subplot(2,1,2);plot(tnew,x);axis([-4 14 -0.5 1.5]) MCL 31
» xlabel('t');ylabel('x((t/3)-1)')                     MCL 32
» title('x((t/3)-1) versus t')                         MCL 33

```

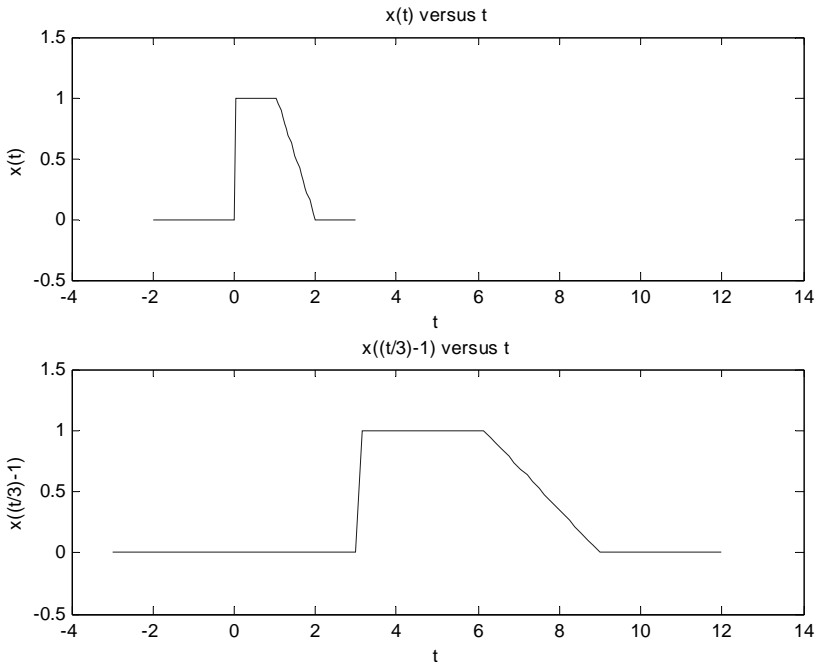


Figure 3.6 Plot of $x(t)$ versus t (upper panel) and $x((t/3)-1)$ versus t (lower panel).

Part 2: Discrete Time Case

We will again create a test signal first and play with it to observe how different transformations of the independent variable (index n , this time) affect our signal. Let our test signal be a triangular one of length 21 (points). We will generate this signal as follows.

```

» n=0:20;                                               MCL 34
» xn=[0:10 9:-1:0];                                    MCL 35

```

Note how we combined two row vectors, forming increasing and decreasing parts of our signal, into a single vector above. To generate x_n ,

instead of using `MCL 35`, we could have issued the following command.

```
» xn=[0 1 2 3 4 5 6 7 8 9 10 9 8 7 6 5 4 3 2 1 0];
```

However, as you may appreciate, typing the above line is much harder than typing the `MCL 35` and as the length of signals -which we define as vectors in Matlab- increases, the *colon*⁴ operator becomes more practical.

Let us explore our signal a little more detail now. Let us, for instance, try to see our signal's first few values. How do we do this? We may think that issuing the following line will give us the values of `xn` for $n=0, 1, 2$.

```
» xn(0:2)
```

But we got an error message saying that “index into matrix is negative or zero” instead. What does this mean? This simply points out the fact that indices of vectors/arrays in Matlab start with 1. Table 3.1 shows how our variables are stored or kept in Matlab's workspace, i.e. the chunk of memory occupied and/or used by Matlab.

Table 3.1: Matlab's indexing start at 1.

Matlab Index	1	2	3	...	10	11	12	...	19	20	21
<code>n</code>	0	1	2	...	9	10	12	...	18	19	20
<code>xn</code>	0	1	2	...	9	10	9	...	2	1	0

Therefore, the correct way of viewing the first samples/points of the index `n` and the signal `xn` would be as follows.

```
» n(1:3), xn(1:3)
```

Note that we used a *comma* (instead of a *semicolon* which suppresses the output) to separate the two Matlab commands so that we can see the output. Similarly, we can explore the last few points of `n` and `xn` as follows.

⁴ `x:y:z` produces a vector of numbers starting from `x` going up (or down) to `z`, with increments (or decrements) of size `y`, depending on whether `y` is positive or negative. If `y` is omitted in this notation, it is assumed to be 1 by default.

```
» n(19:21), xn(19:21)
```

As our first example of transformation in discrete time, we will make the plot of $x[n-5]$ versus n . Let us now see how our test signal looks. (The result is shown in the upper panel of Figure 3.7.)

```
» subplot(2,1,1);stem(n,xn);axis([-1 26 0 11])      MCL 36
» xlabel('n');ylabel('x[n]');title('x[n] versus n')  MCL 37
```

Similar to what we did in continuous time, we call $x[n-5]$ as $y[n]$, and find out that $y[n+5]=x[n]$. We can therefore do the plot of $y[n]=x[n-5]$ versus n as follows. (The result is shown in the lower panel of Figure 3.7.)

```
» subplot(2,1,2);stem(n+5,xn);axis([-1 26 0 11])  MCL 38
» xlabel('n');ylabel('x[n-5]')                    MCL 39
» title('x[n-5] versus n')                          MCL 40
```

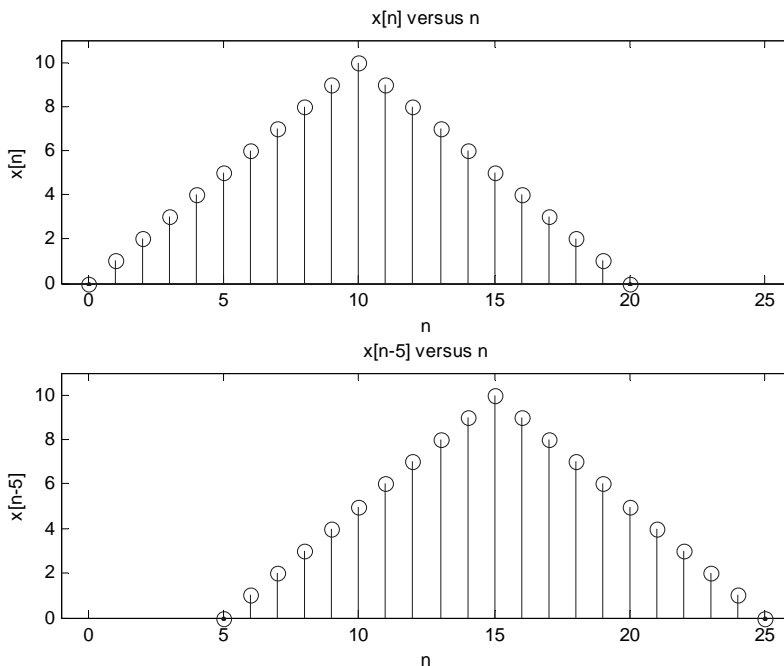


Figure 3.7 Plot of $x[n]$ versus n (upper panel) and $x[n-5]$ versus n (lower panel).

We have just seen an example of time shift; let us now try to plot $x[n/2]$ versus n , as an example of time scaling. Let $y[n]=x[n/2]$; we

immediately note that $y[n]$ is not defined when n is not divisible by two, i.e. when n is odd. One way of overcoming this difficulty is to redefine $y[n]$ as follows.

$$y[n] = \begin{cases} x[n/2], & \text{when } n \text{ is even (i.e. } n = 2m) \\ 0 & \text{, when } n \text{ is odd (i.e. } n = 2m + 1) \end{cases}$$

Therefore, we can generate $y[n]$ with the following lines.

```

» nnew=0:40;                               MCL 41
» yn=zeros(1,41);                           MCL 42
» yn(1:2:41)=xn;                             MCL 43
    
```

Since n , the index for xn , extends from 0 to 20, $nnew$, the index for yn extends from 0 to 40. We defined this new index at MCL 41 and filled yn up with zeros at MCL 42. What we did at MCL 43 is a little bit puzzling. This multi-assignment statement matches or assigns the entries in the target vector (on the right hand side of the assignment or '=' operator) with the entries in the source vector (on the left hand side). Therefore, MCL 43 assigns $xn(1)$ to $yn(1)$, $xn(2)$ to $yn(3)$, $xn(3)$ to $yn(5)$, and so on. We illustrate the effect of this assignment in Table 3.2.

Table 3.2: Explanation of the functioning of the assignment in MCL 43.

Matlab Index	1	2	3	4	5	6	7	...	20	21	22	...	38	39	40	41
n	0	1	2	3	4	5	6	...	19	20	-	...	-	-	-	-
xn	0	1	2	3	4	5	6	...	1	0	-	...	-	-	-	-
nnew	0	1	2	3	4	5	6	...	19	20	21	...	37	38	39	40
yn after MCL 42	0	0	0	0	0	0	0	...	0	0	0	...	0	0	0	0
yn after MCL 43	0	0	1	0	2	0	3	...	0	10	0	...	0	1	0	0

We can now plot $y[n]$ versus n , along with our original signal, as follows.

```

» subplot(2,1,1);stem(n,xn);axis([-2 42 -1 11])      MCL 44
» xlabel('n');ylabel('x[n]')                        MCL 45
» subplot(2,1,2);stem(nnew,yn);axis([-2 42 -1 11])  MCL 46
» xlabel('n');ylabel('y[n]')                        MCL 47

```

As $2n$ stretches from 0 to 40, we rescaled both subplots' x-axes to cover this range, to see the effect of transformation clearly. The result is shown below in Figure 3.8.

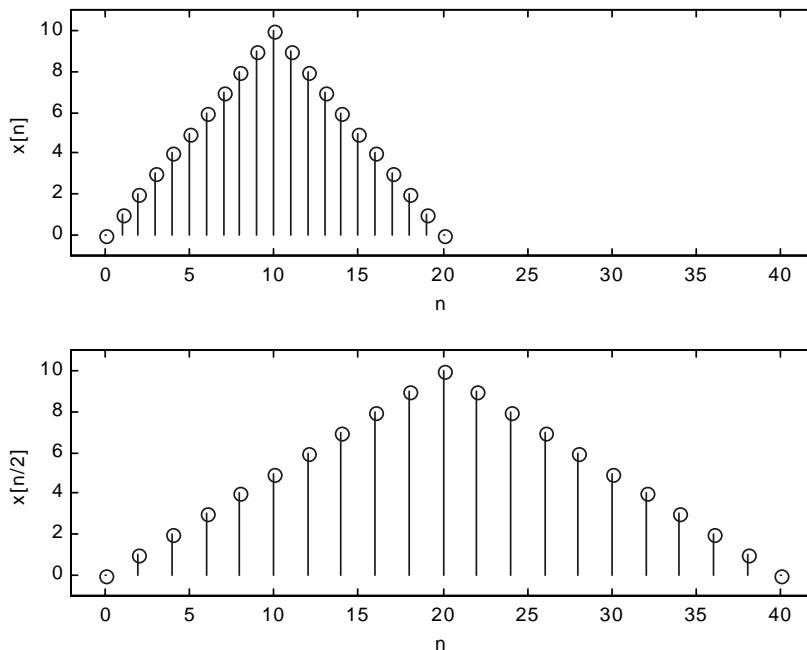


Figure 3.8 Plot of $x[n]$ versus n (upper panel) and $x[n/2]$ versus n (lower panel).