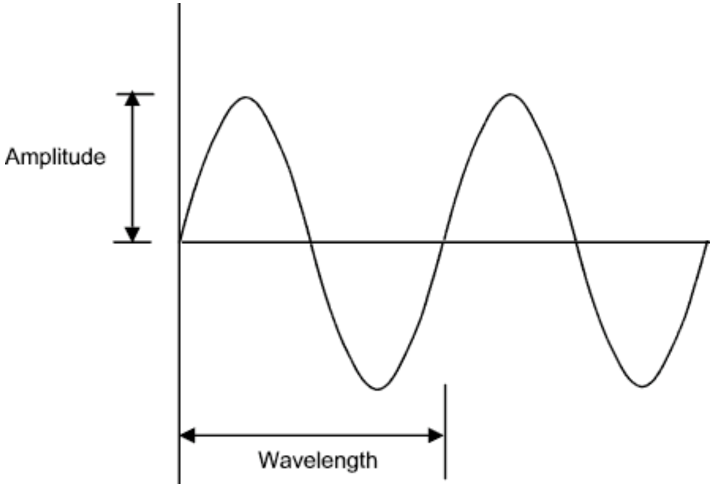
Basic Digital Audio Signal Processing

In this section we look at some basic aspects of Digital Audio Signal Processing:

- Some basic definitions and principles
- Filtering
- Basic Digital Audio Effects

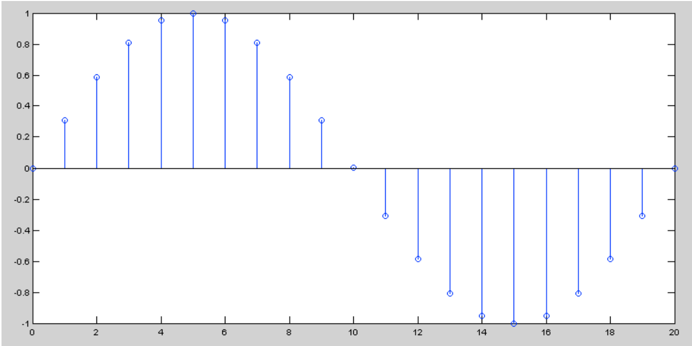
Simple Waveforms



- **Frequency** is the number of cycles per second and is measured in Hertz (Hz)
- **Wavelength** is *inversely proportional* to frequency

  i.e. Wavelength varies as $\frac{1}{frequency}$

The Sine Wave and Sound



The general form of the sine wave we shall use (quite a lot of) is as follows:

$$y = A.sin(2\pi.n.F_w/F_s)$$

where:

A is the amplitude of the wave,
F_w is the frequency of the wave,
F_s is the sample frequency,
n is the sample index.

MATLAB function: `sin()` used — works in radians

MATLAB Sine Wave Radian Frequency Period

Basic 1 period Simple Sine wave — **1 period is 2π radians**

```
% Basic 1 period Simple Sine wave

i=0:0.2:2*pi;
y = sin(i);
figure(1)
plot(y);

% use stem(y) as alternative plot as in lecture notes to
% see sample values

title('Simple 1 Period Sine Wave');
```

MATLAB Sine Wave Amplitude

Sine Wave Amplitude is -1 to +1.

To change amplitude multiply by some gain (`amp`):

```
% Now Change amplitude

amp = 2.0;

y = amp*sin(i);

figure(2)
plot(y);
title('Simple 1 Period Sine Wave Modified Amplitude');
```

MATLAB Sine Wave Frequency

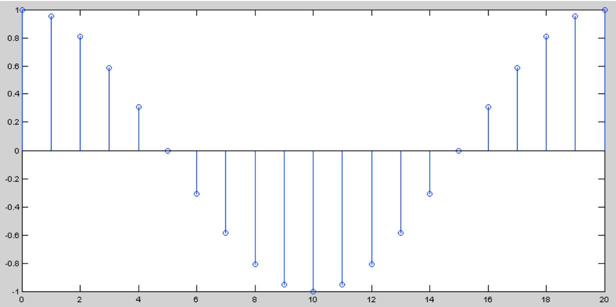
```
% Natural frequency is 2*pi radians
% If sample rate is F_s HZ then 1 HZ is 2*pi/F_s
% If wave frequency is F_w then freequency is F_w* (2*pi/F_s)
% set n samples steps up to sum duration nsec*F_s where nsec is the
% duration in seconds
% So we get y = amp*sin(2*pi*n*F_w/F_s);

F_s = 11025;
F_w = 440;
nsec = 2;
dur= nsec*F_s;

n = 0:dur;

y = amp*sin(2*pi*n*F_w/F_s);

figure(3)
plot(y(1:500));
title('N second Duration Sine Wave');
```

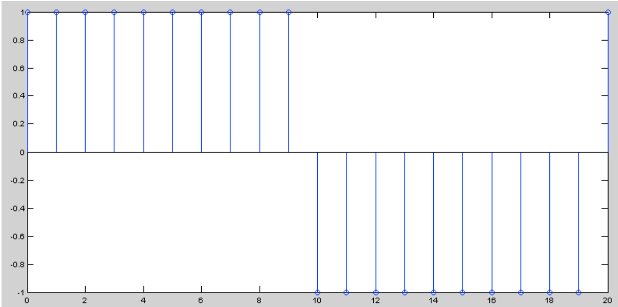
MATLAB Sine Wave Plot of n cycles

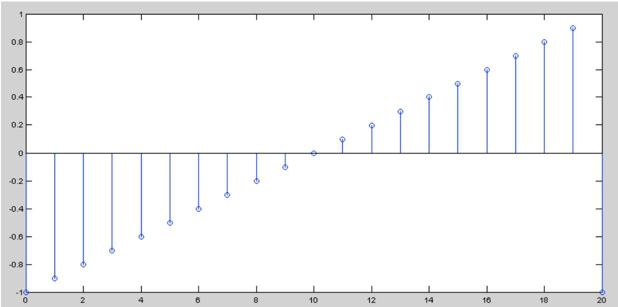
```
% To plot n cycles of a waveform

ncyc = 2;

n=0:floor(ncyc*F_s/F_w);

y = amp*sin(2*pi*n*F_w/F_s);

figure(4)
plot(y);
title('N Cycle Duration Sine Wave');
```

Cosine, Square and Sawtooth Waveforms

MATLAB functions `cos()` (cosine), `square()` and `sawtooth()` similar.



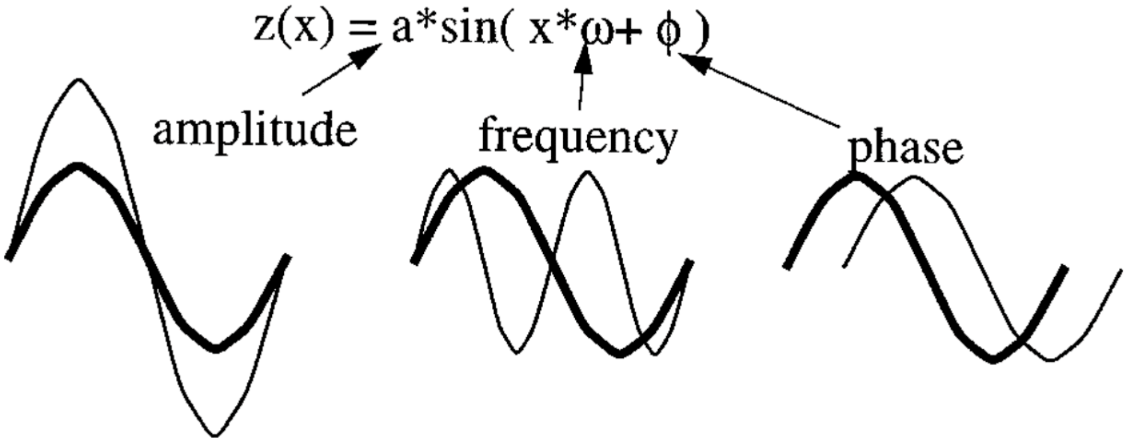




MATLAB Cos v Sin Wave

```
% Cosine is same as Sine (except 90 degrees out of phase)

yc = amp*cos(2*pi*n*F_w/F_s);

figure(5);
hold on
plot(yc,'b');
plot(y,'r');
title('Cos (Blue)/Sin (Red) Plot (Note Phase Difference)');
hold off;
```

Relationship Between Amplitude, Frequency and Phase



Amplitudes of a Sine Wave

```
% Simple Sin Amplitude Demo

samp_freq = 400;
dur = 800; % 2 seconds
amp = 1; phase = 0; freq = 1;
s1 = mysin(amp,freq,phase,dur,samp_freq);

axisx = (1:dur)*360/samp_freq; % x axis in degrees
plot(axisx,s1);
set(gca,'XTick',[0:90:axisx(end)]);

fprintf('Initial Wave: \t Amplitude = ...\n', amp, freq, phase,...);

% change amplitude
amp = input('\nEnter Ampltude:\n\n');

s2 = mysin(amp,freq,phase,dur,samp_freq);
hold on;
plot(axisx, s2,'r');
set(gca,'XTick',[0:90:axisx(end)]);
```

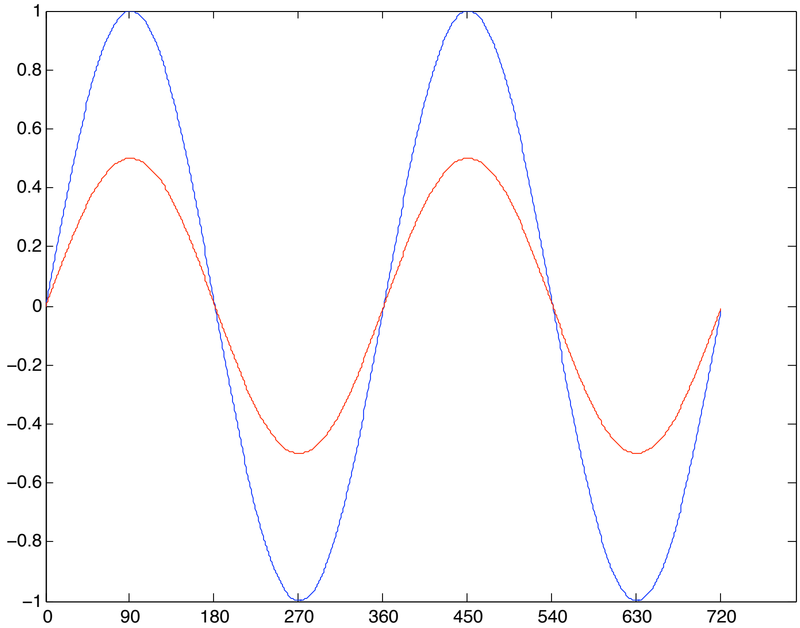
The code is sinampdemo.m

mysin MATLAB code

The above call function mysin.m which a simple modified version of previous MATLAB sin function to account for phase.

```
function s = mysin(amp,freq,phase,dur, samp_freq)
% example function to so show how amplitude,frequency and phase
% are changed in a sin function
% Inputs: amp - amplitude of the wave
%         freq - frequency of the wave
%         phase - phase of the wave in degree
%         dur - duration in number of samples
%         samp_freq - sample frequncy


x = 0:dur-1;
phase = phase*pi/180;

s = amp*sin( 2*pi*x*freq/samp_freq + phase);
```

Amplitudes of a Sine Wave: `sinampdemo` output

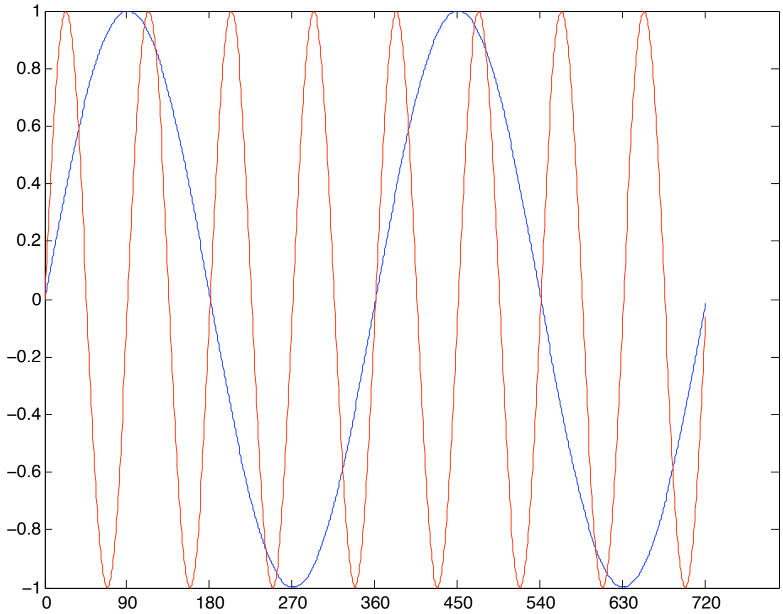


Frequencies of a Sine Wave

```
% Simple Sin Frequency Demo

samp_freq = 400;
dur = 800; % 2 seconds
amp = 1; phase = 0; freq = 1;
s1 = mysin(amp,freq,phase,dur,samp_freq);

axisx = (1:dur)*360/samp_freq; % x axis in degrees
plot(axisx,s1);
set(gca,'XTick',[0:90:axisx(end)]);

fprintf('Initial Wave: \t Amplitude = ...\n', amp, freq, phase,...);

% change amplitude
freq = input('\nEnter Frequency:\n\n');

s2 = mysin(amp,freq,phase,dur,samp_freq);
hold on;
plot(axisx, s2,'r');
set(gca,'XTick',[0:90:axisx(end)]);
```

The code is sinfreqdemo.m

Amplitudes of a Sine Wave: `sinfreqdemo` output

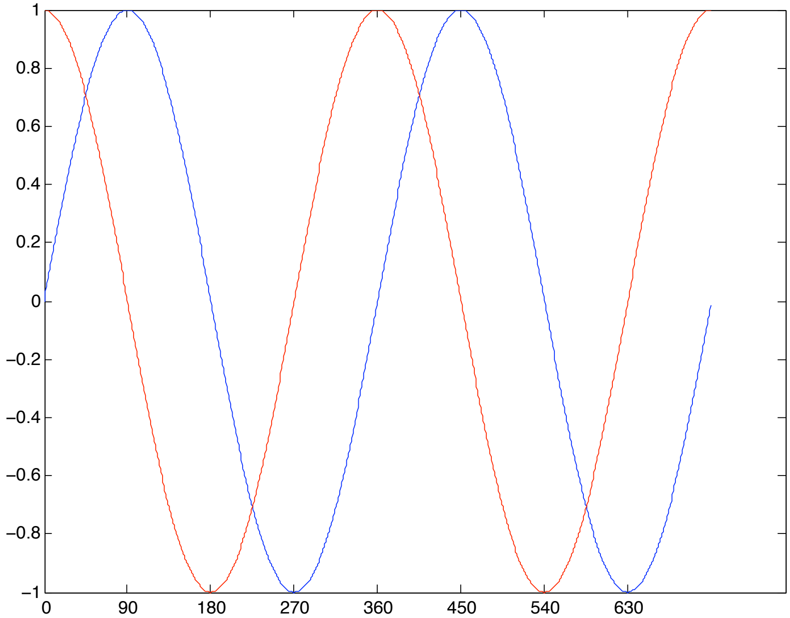


Phases of a Sine Wave

```
% Simple Sin Phase Demo

samp_freq = 400;
dur = 800; % 2 seconds
amp = 1; phase = 0; freq = 1;
s1 = mysin(amp,freq,phase,dur,samp_freq);

axisx = (1:dur)*360/samp_freq; % x axis in degrees
plot(axisx,s1);
set(gca,'XTick',[0:90:axisx(end)]);

fprintf('Initial Wave: \t Amplitude = ...\n', amp, freq, phase,...);

% change amplitude
phase = input('\nEnter Phase:\n\n');

s2 = mysin(amp,freq,phase,dur,samp_freq);
hold on;
plot(axisx, s2,'r');
set(gca,'XTick',[0:90:axisx(end)]);
```

The code is sinphasedemo.m

Amplitudes of a Sine Wave: `sinphasedemo` output



MATLAB Square and Sawtooth Waveforms

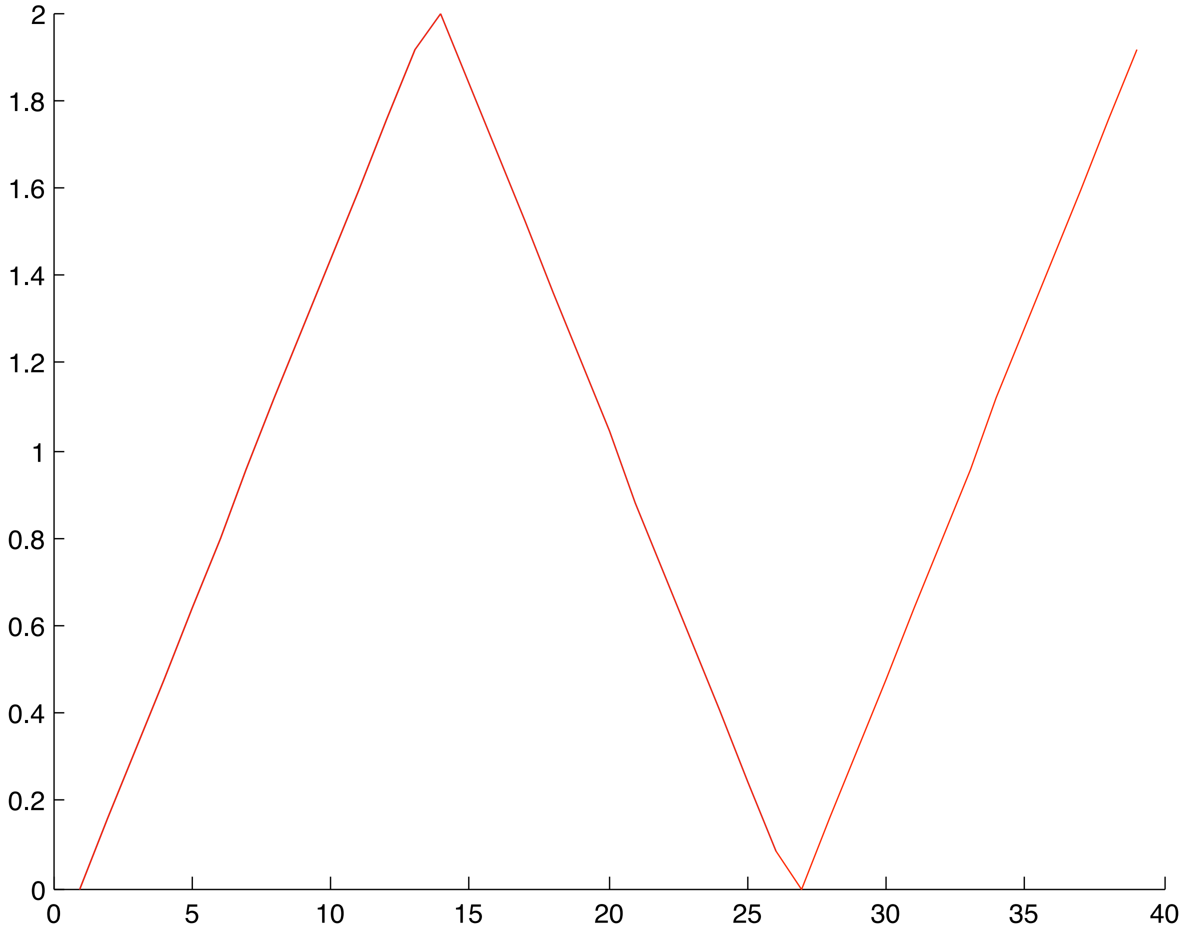
```
% Square and Sawtooth Waveforms created using Radians

ysq = amp*square(2*pi*n*F_w/F_s);
ysaw = amp*sawtooth(2*pi*n*F_w/F_s);

figure(6);
hold on
plot(ysq,'b');
plot(ysaw,'r');
title('Square (Blue)/Sawtooth (Red) Waveform Plots');
hold off;
```

Triangular Waveform

MATLAB function `sawtooth(t,width = 0.5)` can create a triangular waveform, but its easy to build one ourselves (later we make a smoother sounding sawtooth in similar fashion):

```
% Half Frequency
delta = 2*F_w/F_s;
% min and max values of simple waveform
minf=0;maxf=1;
% create triangle wave of centre frequency values
figure(7); hold on
ytri = [];
% plot n cycles
while(length(ytri) < floor(ncyc*F_s/F_w) )
    ytri = [ ytri amp*(minf:delta:maxf) ]; %upslope
    doplot = input('\nPlot Figure? y/[n]:\n\n', 's');
    if doplot == 'y',
       plot(ytri,'r');
       figure(7);
    end
    lasti = length(ytri);
    ytri = [ ytri amp*(maxf:-delta:minf) ]; %downslope
    doplot = input('\nPlot Figure? y/[n]:\n\n', 's');
    if doplot == 'y',
        plot(ytri,'b');
        figure(7);
    end
end
title('Triangular Waveform Plots');  hold off;
```

Triangular Waveform Display



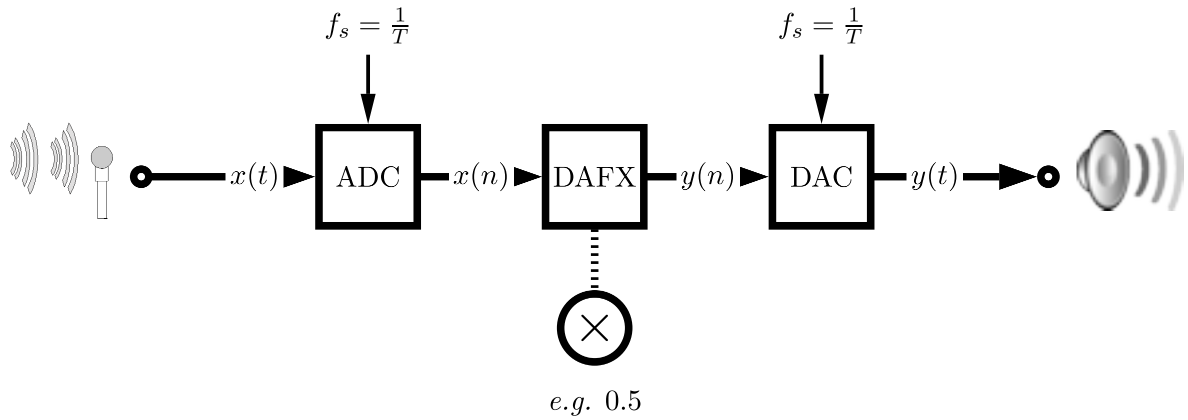
Using these Waveforms

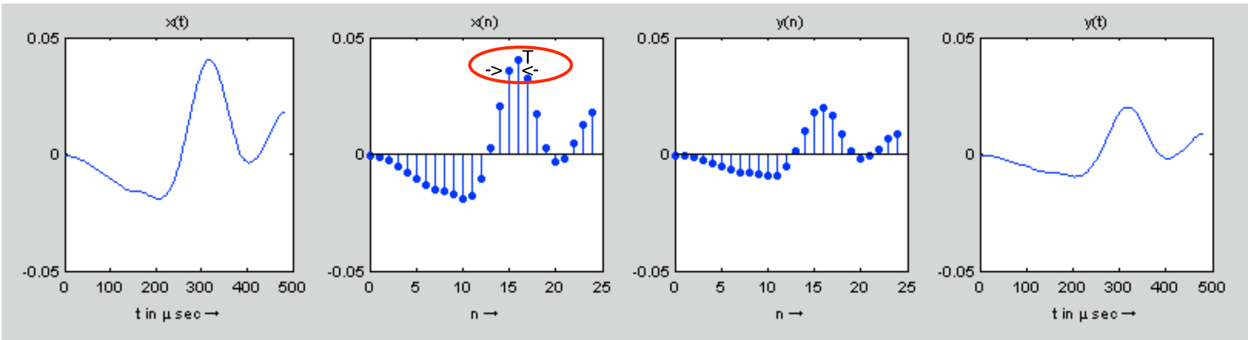
All above waveforms used (as seen in Lecture notes):

- Modulators and Carrier waveforms for various Digital Audio effects.
  - Low Frequency Oscillators (LFO) to vary filter cut-off frequencies and delay times
- Base waveforms for various forms of synthesis: Subtractive, FM, Additive (**CM0340 Multimedia Year 3**)

Simple Digital Audio Effects Example

Over the next few slides consider the following example:

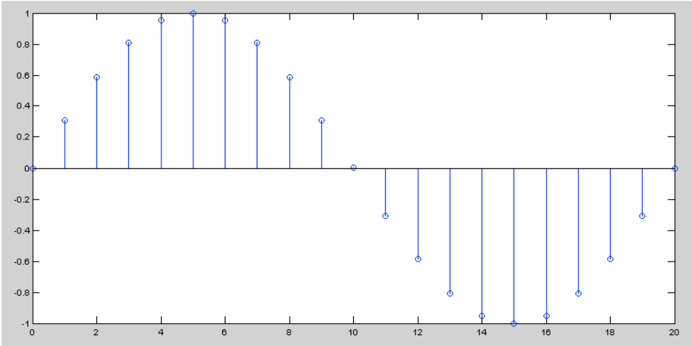




Sample Interval and Sample Frequency

- An **analog signal,** $x(t)$ with signal amplitude continuous over time, t.
- Following **ADC** the signal is converted into a **a discrete-time and quantised amplitude signal,** $x(n)$ — a stream of samples over discrete time index, n
  - The time distance between two consecutive samples, **the sample interval,** T (or sampling period).
  - The **the sampling frequency** is $f_s = \frac{1}{T}$ — the number of samples per second measured in Hertz (Hz).
- Next we apply some simple **DAFX** — *E.g* here we multiply the signal by a factor of 0.5 to produce $y(n) = 0.5.x(n)$.
- The signal $y(n)$ is then forwarded to the **DAC** which reconstruct an analog signal $y(t)$

The Sine Wave and Sound



The general form of the sine wave we shall use (quite a lot of) is as follows:

$$y = A.sin(2\pi.n.F_w/F_s)$$

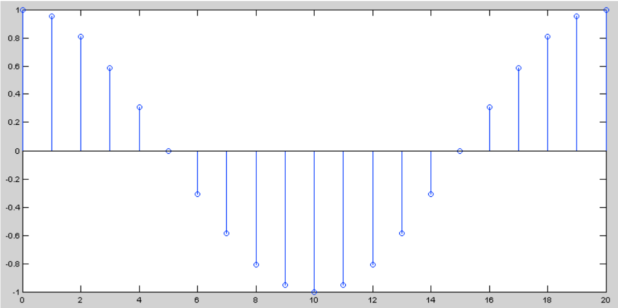
where:

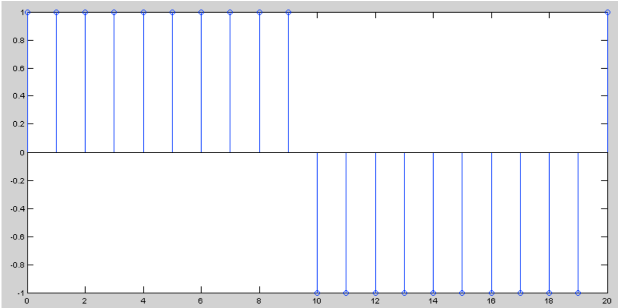
A is the amplitude of the wave,
F_w is the frequency of the wave,
F_s is the sample frequency,
n is the sample index.

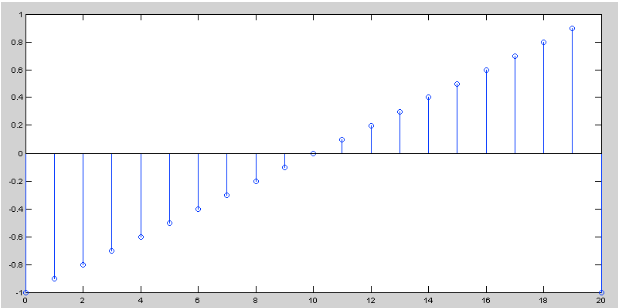
MATLAB function: `sin()` used — works in radians

Cosine, Square and Sawtooth Waveforms

MATLAB functions `cos()` (cosine), `square()` and `sawtooth()` similar.







The Decibel (dB)

When referring to measurements of power or intensity, we express these in decibels (dB):

$$X_{dB} = 10 \log_{10} \left(\frac{X}{X_0} \right)$$

where:

- X is the actual value of the quantity being measured,
- X_0 is a specified or implied reference level,
- X_{dB} is the quantity expressed in units of decibels, relative to X_0.
- X and X_0 **must** have the same dimensions — they must measure the same type of quantity in the the same units.
- The reference level itself is **always at 0 dB** — as shown by setting $X = X_0$ (**note:** $\log_{10}(1) = 0$).

Why Use Decibel Scales?

- When there is a large range in frequency or magnitude, logarithm units often used.
- If X is greater than X_0 then X_{dB} is positive (Power Increase)
- If X is less than X_0 then X_{dB} is negative (Power decrease).
- Power Magnitude $= |X(i)|^2|$ so (with respect to reference level)

$$\begin{aligned} X_{dB} &= 10\log_{10}(|X(i)^2|) \\ &= 20\log_{10}(|X(i)|) \end{aligned}$$

which is an expression of dB we often come across.

Decibel and acoustics

- dB is commonly used to quantify sound levels relative to some 0 dB reference.
- The reference level is typically set at the *threshold of human perception*
- Human ear is capable of detecting a very large range of sound pressures.

Examples of dB measurement in Sound

Threshold of Pain : The ratio of sound pressure that causes permanent damage from short exposure to the limit that (undamaged) ears can hear is above a million:

- The ratio of the maximum power to the minimum power is above one (short scale) trillion (10^{12}).
- The log of a trillion is 12, so this ratio represents a **difference of 120 dB**.

Speech Sensitivity : Human ear is not equally sensitive to all the frequencies of sound within the entire spectrum:

- Noise levels at maximum human sensitivity — between 2 and 4 kHz (Speech) are factored more heavily into sound descriptions using a process called frequency weighting.

Examples of dB measurement in Sound (cont.)

6dB per bit : In digital audio sample representation (**linear pulse-code modulation (PCM)**),

- The first bit (least significant bit, or LSB) produces residual quantization noise (bearing little resemblance to the source signal)
- Each subsequent bit offered by the system **doubles** the resolution, corresponding to a 6 ($= 10 * \log_{10}(4)$) dB.
- So a 16-bit (linear) audio format offers 15 bits beyond the first, for a dynamic range (between quantization noise and clipping) of (15 x 6) = 90 dB, meaning that the maximum signal is 90 dB above the theoretical peak(s) of quantisation noise.
- 8-bit linear PCM similarly gives (7 x 6) = 42 dB.
- 48 dB difference between 8- and 16-bit which is (48/6 (dB)) 8 times as noisy.

Signal to Noise

Signal-to-noise ratio is a term for the power ratio between a signal (meaningful information) and the background noise:

$$SNR = \frac{P_{signal}}{P_{noise}} = \left(\frac{A_{signal}}{A_{noise}}\right)^2$$

where P is average power and A is RMS amplitude.

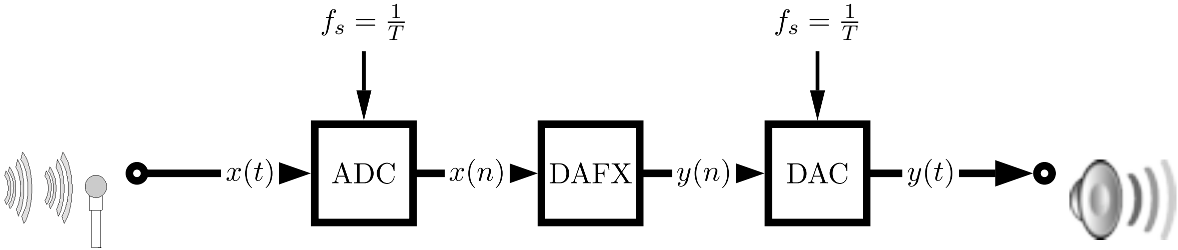
- Both signal and noise power (or amplitude) must be measured at the same or equivalent points in a system, and within the same system bandwidth.

Because many signals have a very wide dynamic range, SNRs are usually expressed in terms of the logarithmic decibel scale:

$$SNR_{dB} = 10\log_{10}\left(\frac{P_{signal}}{P_{noise}}\right) = 20\log_{10}\left(\frac{A_{signal}}{A_{noise}}\right)$$

Digital Systems: Representation and Definitions

Recall this Figure:



A **digital system** is represented by an algorithm which uses the input signal $x(n)$ as a sequence/stream of numbers and performs operations upon the input signal to produce and output sequence/stream of numbers — the output signal $y(n)$.

- *i.e.* the DAFX block in the above figure.

Classifying a Digital System: Block v. sample-by-sample processing

We can classify the way a digital system processes the data in two ways:

- Block v. sample-by-sample processing

  Block processing : data is transferred into a **memory buffer** and then processed each time the buffer is filled with new data.

  E.g. fast Fourier transforms (FFT), Discrete Cosine Transform (DCT), convolution — **more soon**

  Sample-by-sample processing : input is processed on individual sample data.

  E.g. volume control, envelope shaping, ring modulation.

Linear v. Non-linear Time Invariant Systems

A second means of classification:

Linear time invariant system (LTI) : Systems that **do not change** behaviour over time and satisfy the superposition theory. The output signal is signal changed in amplitude and phase.
I.e. A sine wave is still a sine wave just modified in amplitude and/or phase

E.g. Convolution, Filters

Non-linear time invariant system : Systems whose output is strongly shaped by non-linear processing that introduces harmonic distortion — *i.e.* harmonics that are not present in the original signal will be contained in the output.
I.e. if a sine wave is input the output may be a modified waveform or a sum of sine waves (*see Fourier Theory* later) whose frequencies may not be directly related to the input wave.

E.g. Limiters, Compressors, Exciters, Distortion, Enhancers.

Linear Time Invariant Systems

Linear time invariant system are classified by the relation to their input/output functions, these are based on the following terms, definitions and representations:

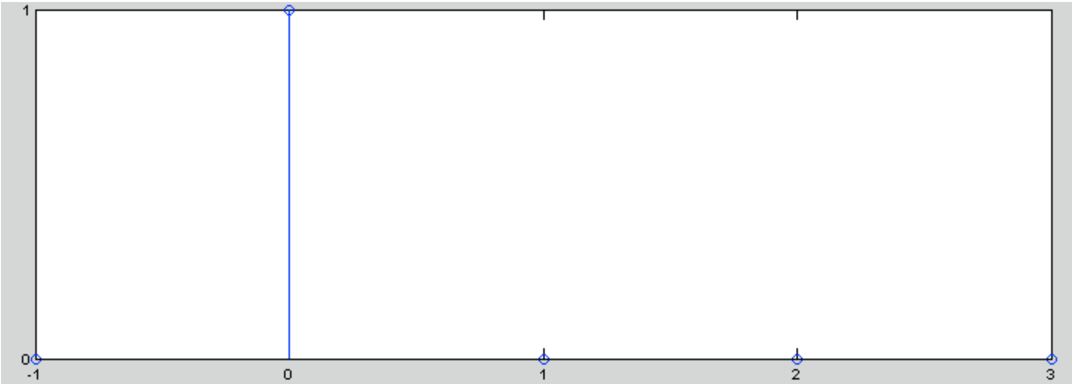
- Impulse Response and discrete convolution
- Algorithms and signal flow graphs

Impulse Response: Unit Impulse

Unit Impulse:

- A very useful test signal for digital systems
- Defined as:

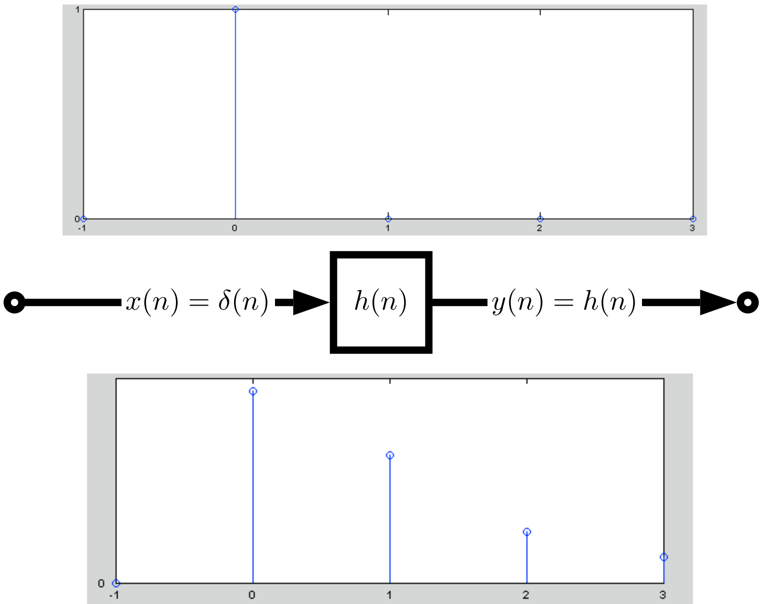
$$\delta(n) = \begin{cases} 1 & \text{if } n = 0 \\ 0 & \text{otherwise } (n \neq 0) \end{cases}$$



Impulse Response Definition

If we apply a unit sample (impulse) function to a digital system we get an output signal $y(n) = h(n)$

- h(n) is called the **impulse response** of the digital system.



Impulse Response: Discrete Convolution

If we know the impulse response $h(n)$ of digital system we can calculate the output signal $y(n)$ for a given $x(n)$ by the **discrete convolution** formula:

$$y(n) = \sum_{k=-\infty}^{\infty} x(k).h(n-k) = x(n) * h(n)$$

- This is usually denoted as $y(n) = x(n) * h(n)$
- Computationally this usually computing using the **fast convolution** method using the **fast Fourier transform** — **more soon**
- MATLAB `y = conv(x,h)` function performs this task.
- Convolution has many DSP applications including denoising, deblurring and reverb — **more soon**.

Representation: Algorithms and Signal Flow Graphs

It is common to represent digital system signal processing routines as a visual **signal flow** graphs.

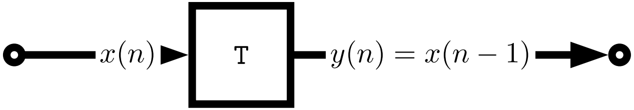
We use a simple *equation* relation to describe the algorithm.

We will need to consider *three* representations:

- Delay
- Multiplication
- Summation

Signal Flow Graphs: Delay

- We represent a delay of one sampling interval by a block with a `T` label:



- We describe the algorithm via the equation: $y(n) = x(n-1)$

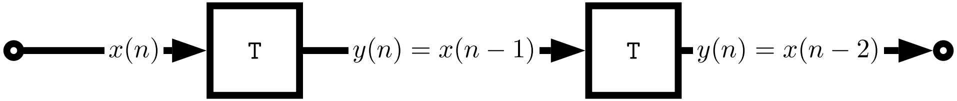
Signal Flow Graphs: Delay Example

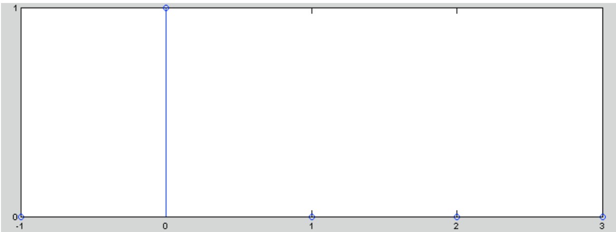
A delay of the input signal by **two** sampling intervals:

- We can describe the **algorithm** by:

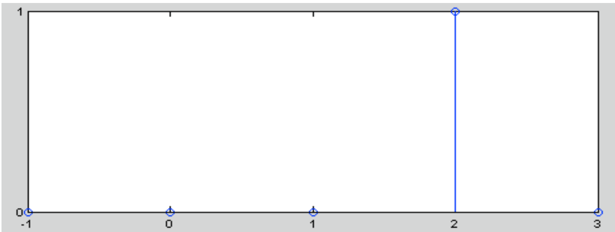
$$y(n) = x(n-2)$$

- We can use the block diagram to represent the **signal flow graph** as:





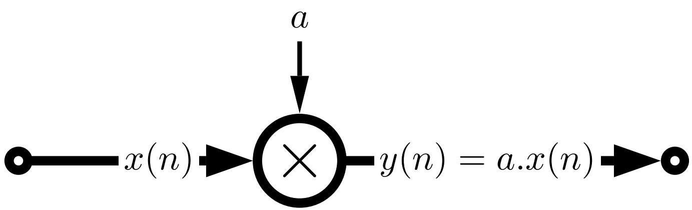
$x(n)$



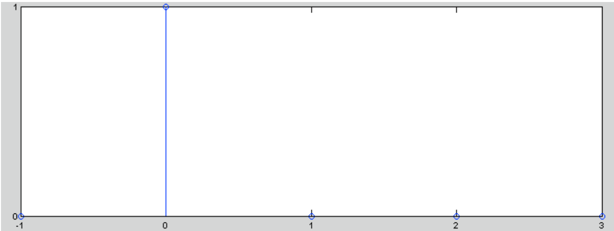
$y(n) = x(n-2)$

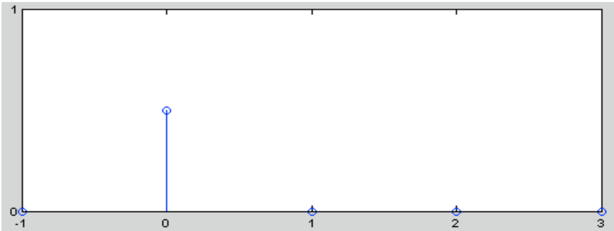
Signal Flow Graphs: Multiplication

- We represent a multiplication or weighting of the input signal by a circle with a $\times$ label.
- We describe the algorithm via the equation: $y(n) = a.x(n)$



$e.g.\ \ a = 0.5$



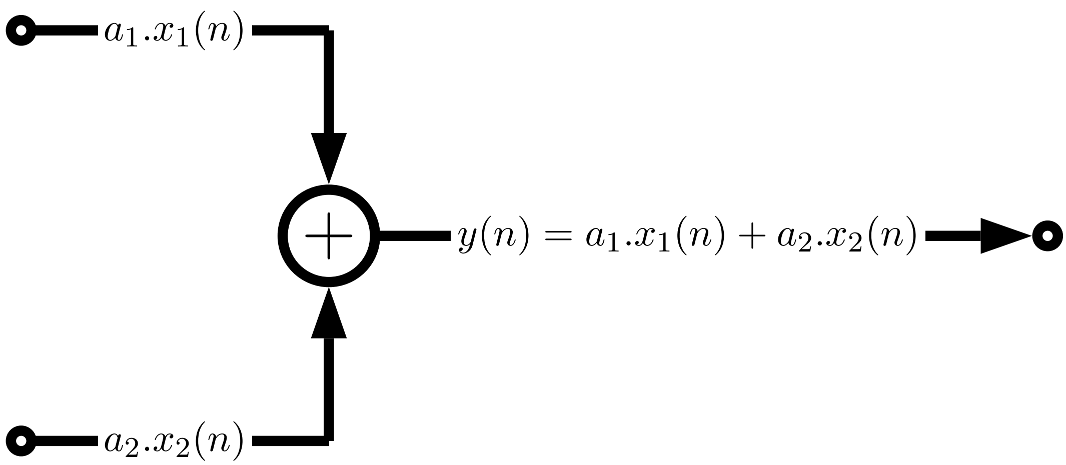


$x(n)$ $\qquad$ $y(n) = 0.5x(n)$

Signal Flow Graphs: Addition

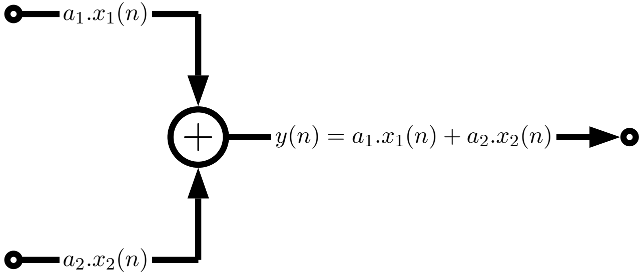
- We represent a addition of two input signal by a circle with a + label.
- We describe the algorithm via the equation:

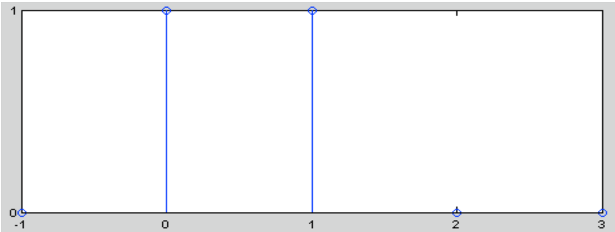
$$y(n) = a_1.x_1(n) + a_2.x_2(n)$$



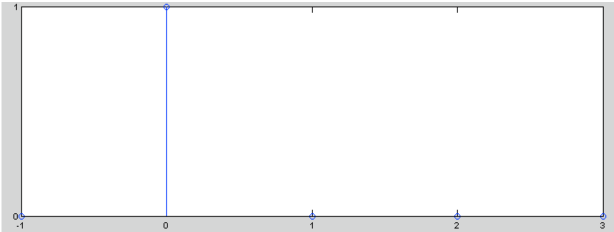
Signal Flow Graphs: Addition Example

In the the example, set $a_1 = a_2 = 1$:

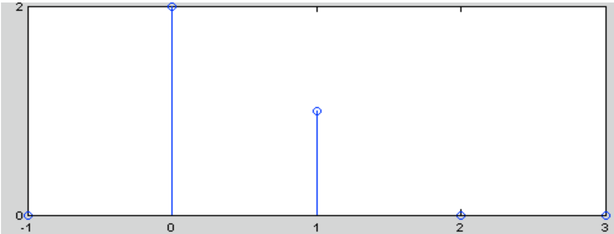




$x_1(n)$



$x_2(n)$



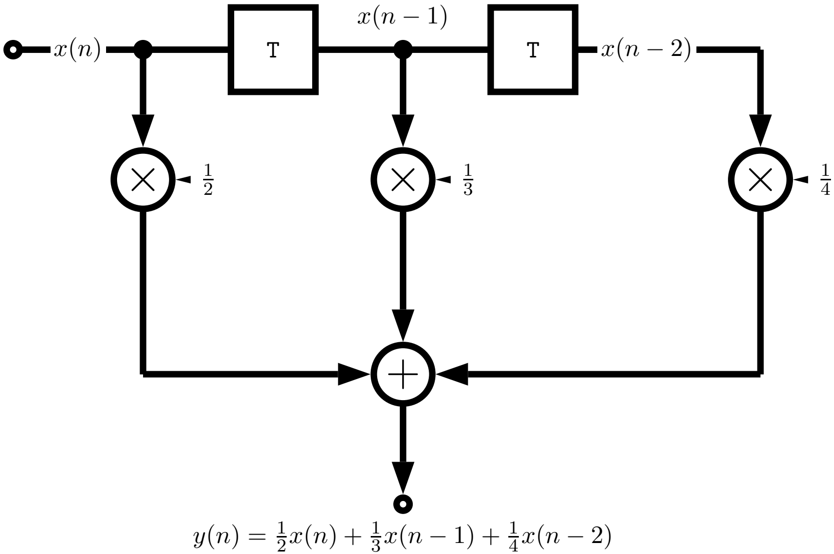
$y(n) = x_1(n) + x_2(n)$

Signal Flow Graphs: Complete Example

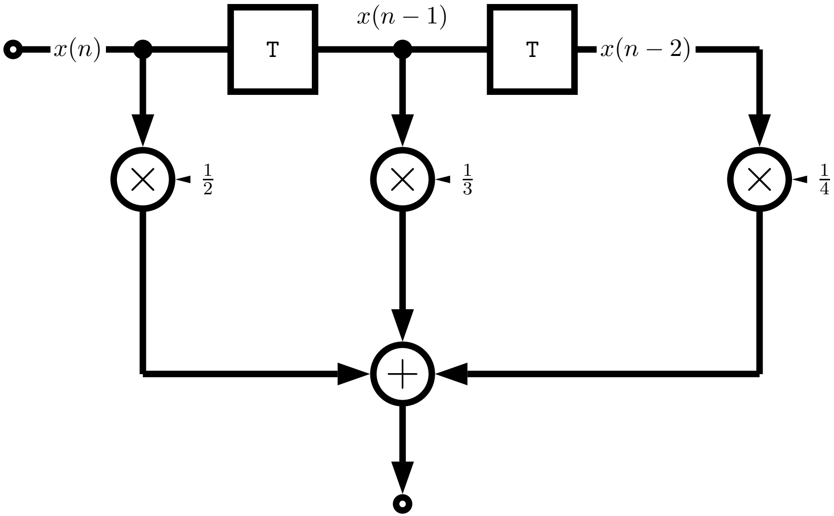
We can combine all above algorithms to build up more complex algorithms:

$$y(n) = \frac{1}{2}x(n) + \frac{1}{3}x(n-1) + \frac{1}{4}x(n-2)$$

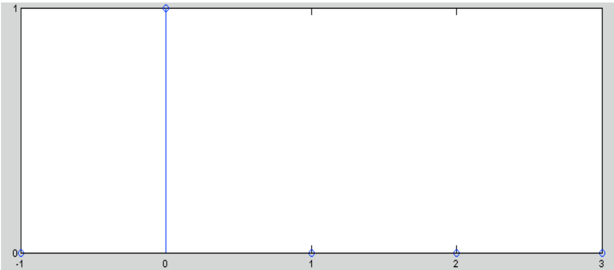
- This has the following signal flow graph:

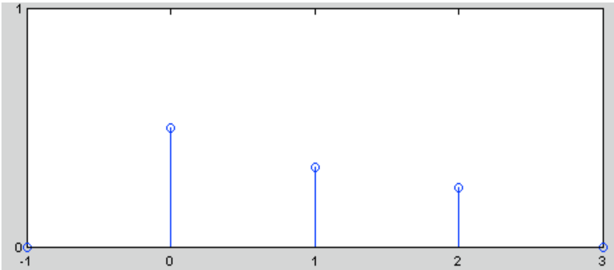


Signal Flow Graphs: Complete Example Impulse Response



$$y(n) = \frac{1}{2}x(n) + \frac{1}{3}x(n-1) + \frac{1}{4}x(n-2)$$





$x(n)$

$$y(n) = \frac{1}{2}x(n) + \frac{1}{3}x(n-1) + \frac{1}{4}x(n-2)$$

Transfer Function and Frequency Response

In a similar way to measuring the **time domain** impulse response $h(n)$ of a digital system we can measure the frequency domain response.

- We can measure this by an impulse response

The frequency domain behaviour of digital systems reflects the systems ability to:

- Pass,
- Reject, and
- Enhance

certain frequencies in the input signal frequency spectrum.

We describe such behaviour with a **transfer function** $H(z)$ and the **frequency response** $H(f)$ of the digital system.

The Z-Transform and Fourier Transform

We need some means to obtain, $H(z)$ and $H(f)$.

The **Z-Transform** is defined as:

$$X(z) = \sum_{n=-\infty}^{\infty} x(n).z^{-n}$$

The **Fourier Transform** is defined as:

$$X(e^{i\Omega}) = \sum_{n=-\infty}^{\infty} x(n).e^{-i\Omega n} \text{ where } \Omega = 2\pi f/f_s$$

Clearly **both** transforms are related by substitution of $z \leftrightarrow e^{i\Omega}$

Note: Will study the Fourier Transform in some detail very soon.

Deriving The Transfer Function and Frequency Response

Given an impulse response h(n) simply apply the Z-Transform:

$$H(z) = \sum_{n=-\infty}^{\infty} h(n).z^{-n}$$

to get the **transfer function** $H(z)$.

Similarly apply the Fourier Transform:

$$H(f) = \sum_{n=-\infty}^{\infty} h(n).e^{-i2\pi fn/f_s}$$

to get the **Frequency Response** $H(f)$.

The Z-Transform

The Z-Transform is a general tool that is used to analyse discrete functions (sequences) and related difference equations (recurrence relations).

- As we have seen above the Fourier Transform can be regarded a special case of the Z-Transform ($z \leftrightarrow e^{i\Omega}$).

The **Z-Transform** is defined as:

$$X(z) = \sum_{k=-\infty}^{\infty} x(k).z^{-k}$$

Z-Transform shorthand notation:

$$X(z) = Z\{x_k\}_{-\infty}^{\infty}$$

where $\{x_k\}_{-\infty}^{\infty}$ is the sequence of samples $\ldots, x_{-2}, x_{-1}, x_0, x_1, x_2, \ldots$

The Causal Z-Transform

$\{x_k\}$ and $Z\{x_k\}$ are called the **transform pair**.

We only really deal with sample sequence indexed: $\{x_k\}_0^\infty$.
(For brevity we will often denote this series as $\{x_k\}$)

Such sequences are called **causal**.

So we can develop the **causal Z-transform**:

$$X(z) = Z\{x_k\}_0^\infty = \sum_{k=0}^{\infty} x_k . z^{-k}$$

Note: In practice we deal with **finite** sample sequence so we usually have N samples: $\{x_k\}_0^{N-1}$.

Example:The Z-transform of the Unit Impulse Response

Unit impulse response δ_k has a sequence $\{1, 0, 0, 0 \ldots\}$

So the Z-transform is:

$$
\begin{aligned}
Z\{\delta_k\} &= \sum_{k=0}^{N-1} \delta_k . z^{-k} = \sum_{k=0}^{N-1} \frac{\delta_k}{z^k} \\
&= 1 + \frac{0}{z} + \frac{0}{z^2} + \frac{0}{z^3} + \ldots \\
&= 1
\end{aligned}
$$

Example: Another Z-transform

Consider the sequence $\{1(= a^0), a, a^2, a^3 \ldots\}$
So the Z-transform is:

$$
\begin{aligned}
Z\{a^k\} &= \sum_{k=0}^{N-1} a^k.z^{-k} = \sum_{k=0}^{N-1} \frac{a^k}{z^k} \\
&= 1 + \frac{a}{z} + \left(\frac{a}{z}\right)^2 + \left(\frac{a}{z}\right)^3 + \ldots
\end{aligned}
$$

Now

$$
\frac{1}{1-x} = 1 + x + x^2 + x^3 \ldots, \text{ for } |x| < 1
$$

So

$$
\begin{aligned}
Z\{a^k\} &= 1 + \frac{a}{z} + \left(\frac{a}{z}\right)^2 + \left(\frac{a}{z}\right)^3 + \dots \\
&= \frac{1}{1 - \frac{a}{z}}, \text{ provided } |\frac{a}{z}| < 1 \\
&= \frac{z}{z-a}, \text{ provided } |z| > |a|
\end{aligned}
$$

Table of Z-transforms

Sequence	Z-transform	Constraints on z
$\{\delta_k\} = \{1, 0, 0, \ldots\}$	1	All values of z
$\{u_k\} = \{1, 1, 1, \ldots\}$	$\frac{z}{z-1}$	$\|z\| > 1$
$\{k\} = \{0, 1, 2, 3, \ldots\}$	$\frac{z}{(z-1)^2}$	$\|z\| > 1$
$\{k^2\} = \{0, 1, 4, 9, \ldots\}$	$\frac{z(z+1)}{(z-1)^3}$	$\|z\| > 1$
$\{k^3\} = \{0, 1, 8, 27, \ldots\}$	$\frac{z(z^2+4z+1)}{(z-1)^4}$	$\|z\| > 1$
$\{a^k\} = \{1, a, a^2, a^3 \ldots\}$	$\frac{z}{z-a}$	$\|z\| > \|a\|$
$\{ka^k\} = \{1, a, 2a^2, 3a^3 \ldots\}$	$\frac{az}{(z-a)^2}$	$\|z\| > \|a\|$

Using MATLAB to compute Z-transforms

If you have access to MATLAB's symbolic toolbox then you can use the `ztrans` function to compute the Z-transform equations, ztransform_demo.m:

```
%  Compute Z transform of n
syms n;  % make n symbolic variable
f = n;
ztrans(f)

ans =

z/(z - 1)^2

%  Compute Z transform of n^2
f = n^2;
ztrans(f)

ans =

(z^2 + z)/(z - 1)^3

%  Compute Z transform of n^3
f = n^3;
ztrans(f)

ans =

(z^3 + 4*z^2 + z)/(z - 1)^4
```

```
%  Compute Z transform of n^4
f = n^4;
ztrans(f)

ans =

(z^4 + 11*z^3 + 11*z^2 + z)/(z - 1)^5

%  Compute Z transform of a^z
syms a z;
g = a^z;
ztrans(g)

ans =

-w/(a - w)


%  Compute Z transform of sin(an)
syms  w;
f = sin(a*n);
ztrans(f, w)

ans =

(w*sin(a))/(w^2 - 2*cos(a)*w + 1)
```

Properties of the Z-transform

Linearity : If a and b are constants then

$$Z(a\{x_k\} + by\{y_k\}) = aZ\{x_k\} + bZ\{y_k\}$$

First Shift Theorem (Left Shift) :

$$Z\{x_{k+m}\} = z^m Z\{x_k\} - [z^m x_0 + z^{m-1}x_1 + \ldots + zx_{m-1}]$$

Second Shift Theorem (Right Shift) :

$$Z\{x_{k-m}\} = z^{-m} Z\{x_k\}$$

Translation : $Z\{a^k x_k\} = X(a^{-1}z)$

Final Value Theorem :

$x(\infty) = \lim_{k\to\infty} x_k = \lim_{z\to 1}\left\{\left(\frac{z-1}{z}\right) X(z)\right\}$ provided $\lim_{k\to\infty} x_k$ exists.

Initial Value Theorem :

$$x(0) = x_0 = \lim_{z\to\infty}\{X(z)\}$$

Derivative of the Z-transform :

If $Z\{x_k\} = X(z)$ then $-zX'(z) = Z\{kx_k\}$

Examples given in lecture.

Inverse Z-transform

If sequence $\{x_k\}$ had the Z-transform $Z\{x_k\} = X(z)$, then the **inverse Z-transform** is defined as:

$$Z^{-1}X(z) = \{x_k\}$$

For more information, see `doc ztrans`, `doc sym/ztrans`, `doc iztrans`, `doc sym/iztrans`