# 1. Sinusoids

$x(t) = A \cdot \cos(\omega_0 t + \varphi) = A\cos(2\pi f_0 t + \varphi)$

where:

- $A$ is amplitude
- $\varphi$ is phase shift, is in radians, not seconds, i.e. time shift relative to the frequency.
- $\omega_0$ is the radian frequency (radians per second)
- $f_0 = \omega_0/(2\pi)$ is the cyclic frequency (normal nice intuitive frequency)

## Period vs. Frequency

$$T_0 = \frac{1}{f_0} = \frac{2\pi}{\omega_0}$$

Where $T_0$ is the *period* of the oscillation, i.e. how long it takes for a single cycle of oscillation, start to finish, whereas $f_0$ is how many oscillations per second.

## Time-Shift vs. Phase-Shift

Phase shift is relative to frequency and is in radians, time shift is absolute and is in seconds.

$x(t - t_0) = A \cdot \cos(\omega_0(t - t_0) + \varphi)$

Where:

- $t_0$: time shift
- $\varphi$: phase shift
- Everything else explained above

## Complex Exponential Form

Complex exponentials are basically vectors that spin around the origin. The x and y coordinates are a sine and cosine function, so sinusoids can be re-written as complex exponentials.

$Ae^{j(\omega_0 t + \varphi)} = A\cos(\omega_0 t + \varphi) + jA\sin(\omega_0 t + \varphi)$

$\Re\{Ae^{j(\omega_0 t + \varphi)}\} = A\cos(\omega_0 t + \varphi)$

### Regneregler

TODO: rewrite this section with an in-depth step by step, see exercise 2.11

$$\cos\theta = \frac{e^{j\theta} + e^{-j\theta}}{2}$$

$$\sin\theta = \frac{e^{j\theta} - e^{-j\theta}}{2j}$$

Sum of sinusoids with the same frequency, but different phases or amplitudes, is a new sinusoid with the same frequency but different amplitude and phase.

$\cos(\dots) + \cos(\dots) + \dots$ (where they have the same frequency but not the same amplitude or phase)

$\sum A_k e^{j\phi_k} = Ae^{j\phi}$

$\Re\{Ae^{j\phi}e^{j\omega_0 t}\} = \cos(\dots)$

If you need to sum a bunch of sinusoids that have the same frequency, just transform them to polar form, add them together, multiply the result by $e^{j\omega_0 t}$ and take the real part and that's the sum.

## 2. Spectrum Representation

Sum of sinusoids, where amplitude, phase and frequency can all freely be changed, can theoretically represent any continuous signal (but might require an infinite sum).

$$x(t) = X_0 + \sum_{k=1}^{N} \Re e\left\{X_k e^{j2\pi f_k t}\right\}$$

$$X_k = A_k e^{j\phi_k}$$

---

$$x(t) = X_0 + \sum_{k=1}^{N} \left\{\frac{X_k}{2} e^{j2\pi f_k t} + \frac{X_k^*}{2} e^{-j2\pi f_k t}\right\}$$

Taking the real component of a sum of complex exponentials, is the same as adding the complex conjugates of each term to the terms, which cancels out the imaginary part. This means that the spectrum has a mirrored negative frequency side which is derived from the complex conjugates.

$$\left\{(X_0, 0),\ \left(\tfrac{1}{2}X_1,\ f_1\right),\ \left(\tfrac{1}{2}X_1^*,\ -f_1\right),\ \left(\tfrac{1}{2}X_2,\ f_2\right),\ \left(\tfrac{1}{2}X_2^*,\ -f_2\right), \ldots\right\}$$

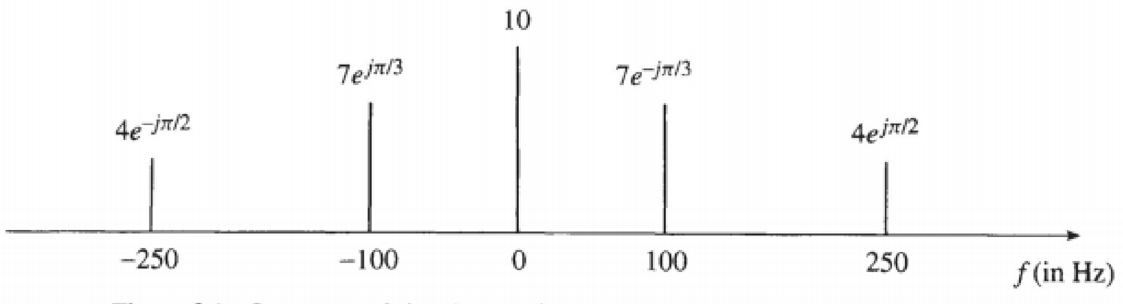


**Figure 3.1** Spectrum of the signal $x(t) = 10 + 14\cos(200\pi t - \pi/3) + 8\cos(500\pi t + \pi/2)$. Positive and negative frequency components must be included even though the negative-frequency ones are the conjugate of the positive-frequency components.

**To solve exercises:** re-write whatever function you have to find the spectrum of into a sum of sinusoids, by first transforming it into complex exponential form, then rearranging the terms and using properties of exponentials to get a sum of complex exponentials, then converting back into a sum of sinusoids.

### Amplitude Modulation

Instead of doing the whole calculations by hand, use these formulas

$$\begin{aligned} x(t) &= \cos(2\pi f_1 t) + \cos(2\pi f_2 t) \\ &= 2\cos(2\pi f_\Delta t)\cos(2\pi f_c t) \end{aligned}$$

$$f_c = \tfrac{1}{2}(f_1 + f_2) \quad f_\Delta = \tfrac{1}{2}(f_2 - f_1)$$

Where $f_\Delta$ and $f_c$ are the frequencies of the sinusoids that are multiplied (modulated) together, whereas $f_1$ and $f_2$ are the frequencies of the equivalent sum of sinusoids.

## Operations on Spectrum

### Addition

Adding a new signal to a spectrum just means adding all the sinusoids from the new signal to the old one, so adding new lines to the plot without changing the existing ones, with on exception. If the frequency is already in the spectrum, adding a sinusoid with the same frequency means you have to calculate the phase and amplitude again, see Chapter 1 regneregler.

### Scaling

Scaling a frequency in the input scales the frequency in the spectrum representation the same amount. The same applies to amplitude and phase.

# 3. Sampling, Aliasing, Shannon, Nyquist

Sampling continuous function into an array of evenly spaced discrete measurements, called *samples.*

$x[n] = x(nT_s)$

Where:

- $n \in (-\infty, \infty)$: is an integer index
- $T_s$: sampling period, i.e. time between each measurement
- $f_s = 1/T_s$: sampling frequency.

Normalized radian frequency:

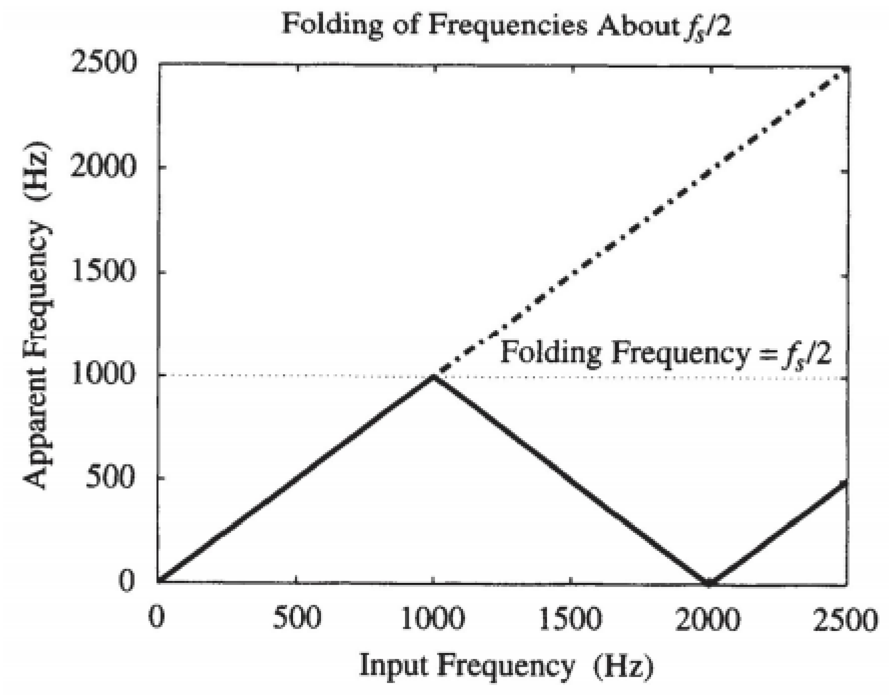
$\hat{\omega} = \omega T_s$

$$x[n] = x(nT_s) = A\cos(\omega n T_s + \phi) = A\cos(\hat{\omega} n + \phi)$$

## Shannon / Nyquist

If a signal is sampled at a frequency $f_s$, the highest frequency signal which can be reconstructed without loss of information is at a frequency of $f_s/2$.

Nyquist frequency/ Nyquist limit: $f_s/2$

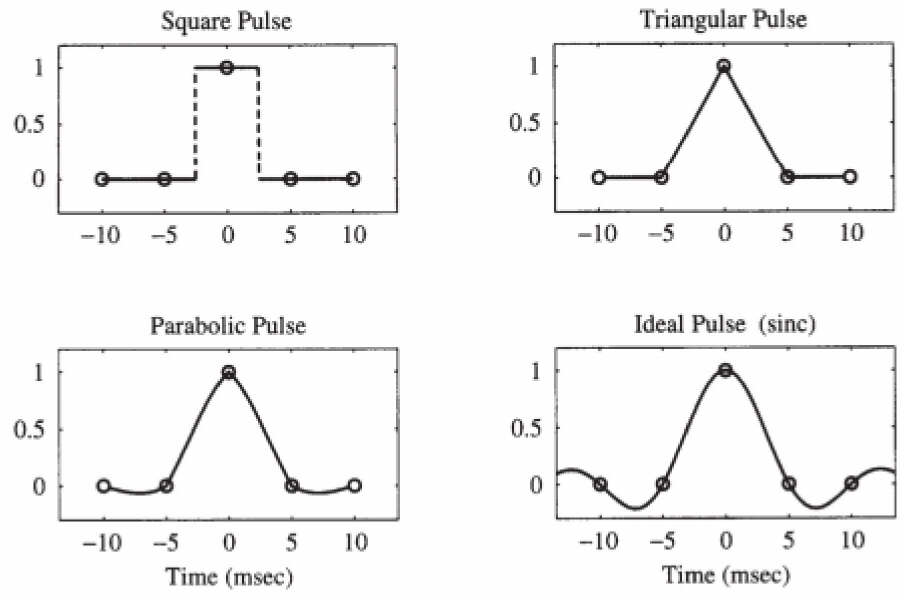


### Magic Formula of Aliasing

$$\begin{aligned} A\cos(\hat{\omega}_0 n + \varphi) &= A\cos((\hat{\omega}_0 + 2\pi\ell)n + \varphi) \\ &= A\cos((2\pi\ell - \hat{\omega}_0)n - \varphi) \end{aligned}$$

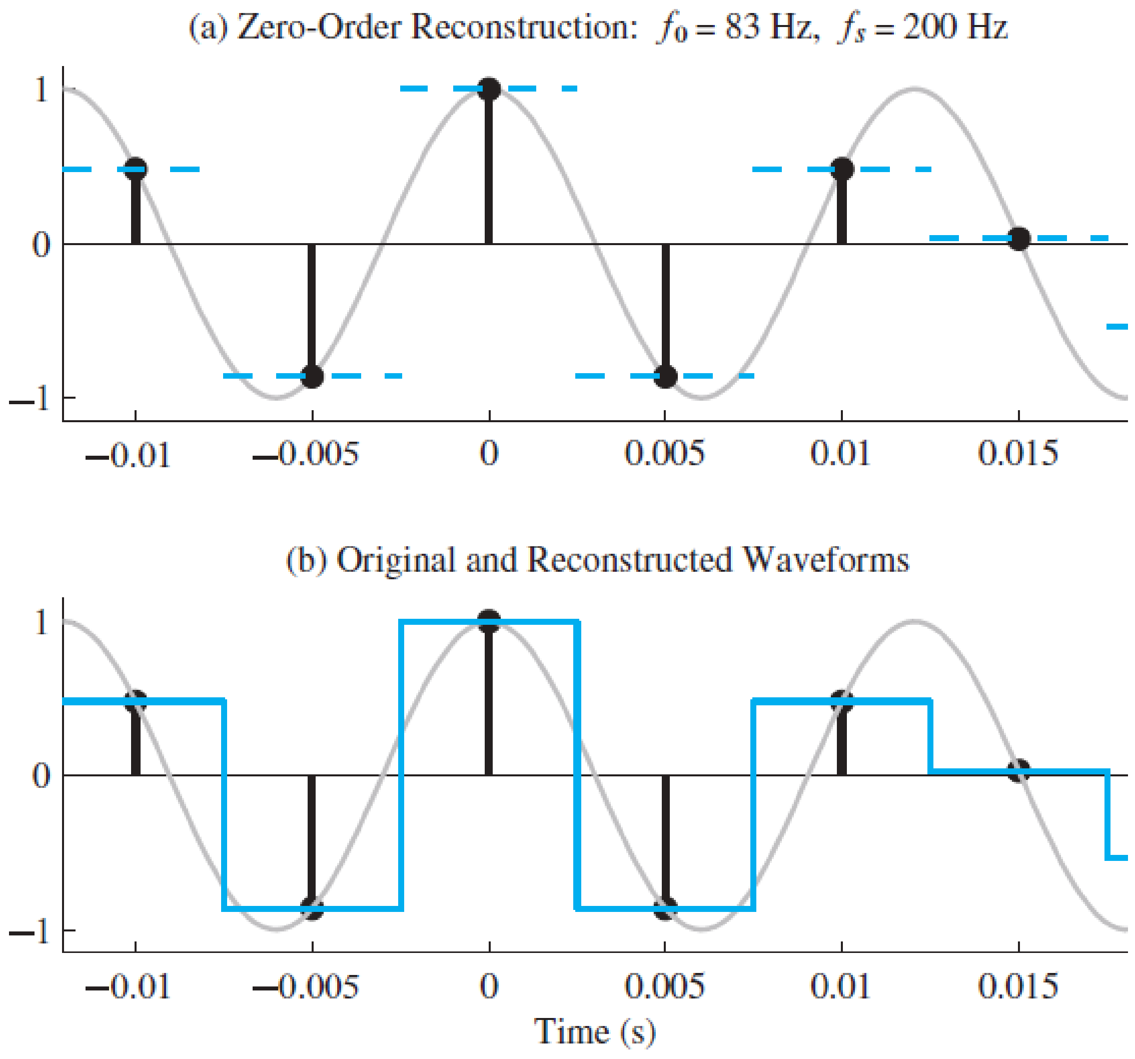
## Digital to Continuous

$$y(t) = \sum_{n=-\infty}^{\infty} y[n]p(t - nT_s)$$



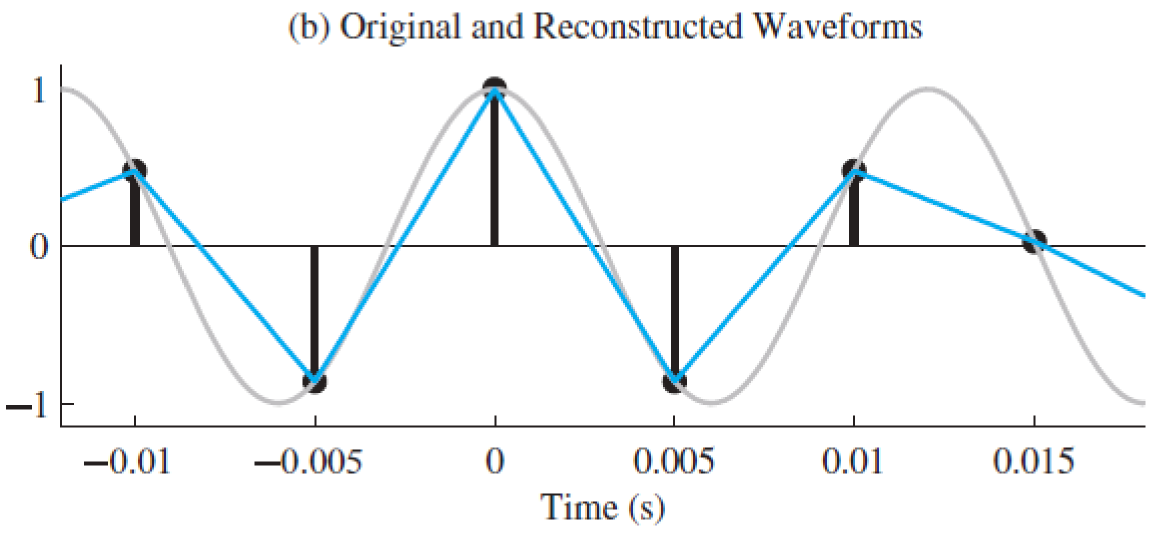
## Square Pulse Interpolation

$$p(t) = \begin{cases} 1 & -\frac{1}{2}T_s < t \le \frac{1}{2}T_s \\ 0 & \text{otherwise} \end{cases}$$



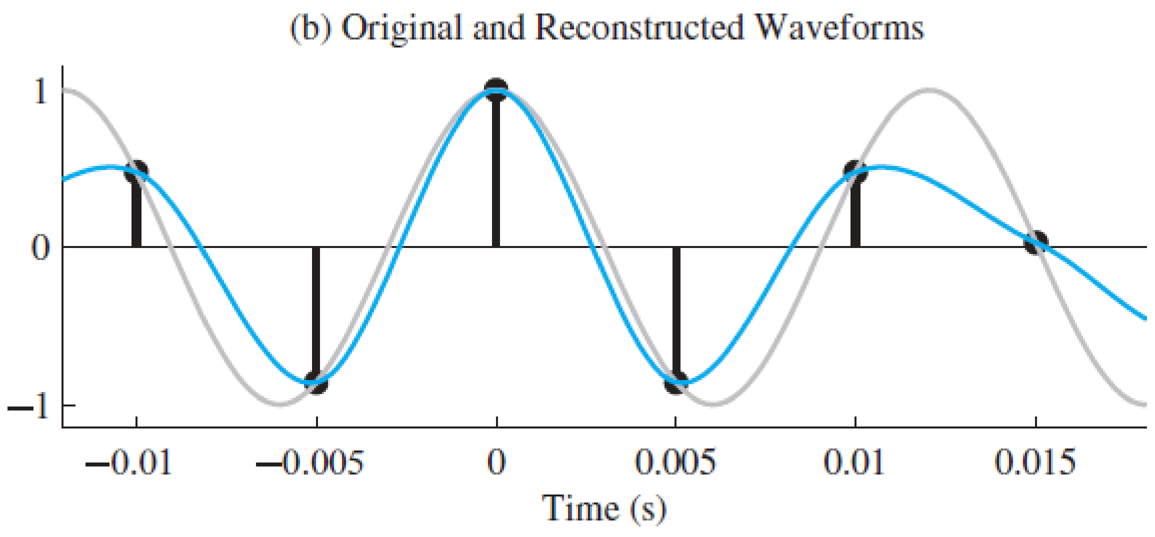
## Linear Interpolation

$$p(t) = \begin{cases} 1 - |t|/T_s & -T_s \le t \le T_s \\ 0 & \text{otherwise} \end{cases}$$



## Cubic Spline Interpolation

Probably not in exam, if it is and you are reading this now during exam then good luck.



## Sinc Interpolation

Sinc is the ideal interpolation, but it requires infinite samples so it's not possible in reality.

$$p(t) = \text{sinc}(t/T_s) = \frac{\sin(\pi t/T_s)}{(\pi t/T_s)} \qquad \text{for } -\infty < t < \infty$$

# 4 & 5. Finite Impulse Response (FIR) Filters

A FIR filter is basically:

- A rolling average where we can freely choose the weights of the inputs.
- A convolution of the input signal with the filter impulse response.

$$y[n] = \sum_{k}^{M} b_k x[n-k]$$

Where :

- $M$ is the size of the filter window.
- $b$ is an array of weights.

## Impulse Response

The impulse response of a system is:

$\delta[n] \Rightarrow \text{system} \Rightarrow h[n]$

Where $\delta$ is the unit impulse function: $\begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases}$

So the impulse response of a FIR filter is:

$$h[n] = \sum_{k}^{M} b_k \delta[n-k]$$

## Convolution

If you have the impulse response of a system, you can filter a signal by taking the convolution:

$y[n] = x[n] * h[n]$

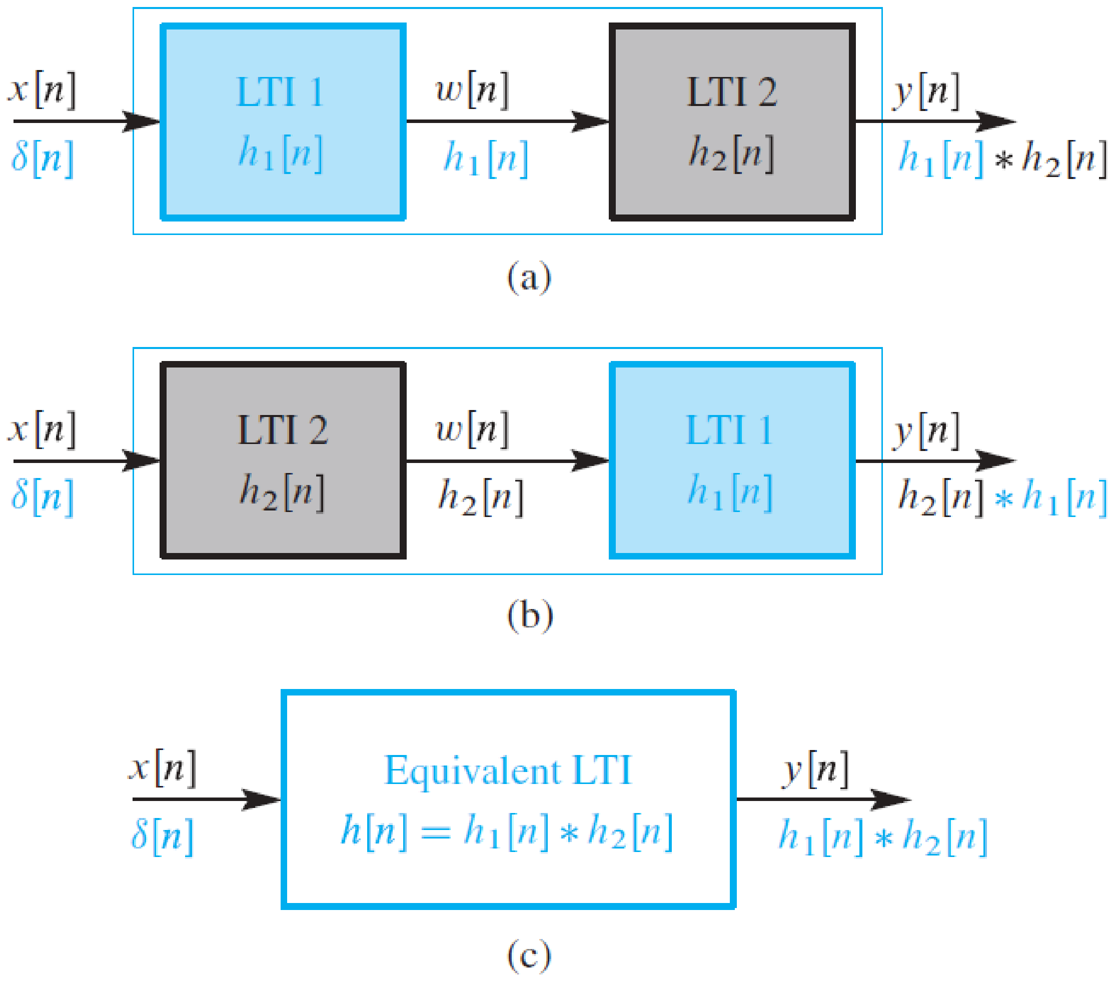
*The Convolution Sum Formula*

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

For a finite impulse response of size $M$, the convolution is the same as the definition of FIR filters. I.e.:

$$y[n] = x[n] * h[n] = \sum_{k}^{M} x[n]h[n-k] = \sum_{k}^{M} b_k x[n-k]$$

If the window or the signal are not infinite, the sum will also not be infinite. So it can in general be done.7



## Frequency Response

For an arbitrary discrete sinusoid input $x$, the response of the filter is:

$$x[n] = Ae^{j\phi}e^{j\hat{\omega}n}$$

$$\begin{aligned} y[n] &= \sum_{k}^{M} b_k Ae^{j\phi}e^{j\hat{\omega}(n-k)} \\ &= \left(\sum_{k}^{M} b_k e^{j\hat{\omega}k}\right) Ae^{j\phi}e^{j\hat{\omega}n} \\ &= \mathcal{H}(\hat{\omega})Ae^{j\phi}e^{j\hat{\omega}n} \end{aligned}$$

This gives a formula for the response of the filter to frequencies:

$$\mathcal{H}(\hat{\omega}) = \sum_{k}^{M} b_k e^{j\hat{\omega}k}$$

## Regneregler

Superposition:

$$\mathcal{H}_3 = a\mathcal{H}_1 + b\mathcal{H}_2$$

Adding or scaling two frequency responses together gives a new frequency response, i.e. they are a linear combination.

## FIR Filter Difference Equations

This is a difference equation:

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

A difference equation is the sum of all the samples multiplied by coefficients that make up the FIR filter, so in general:

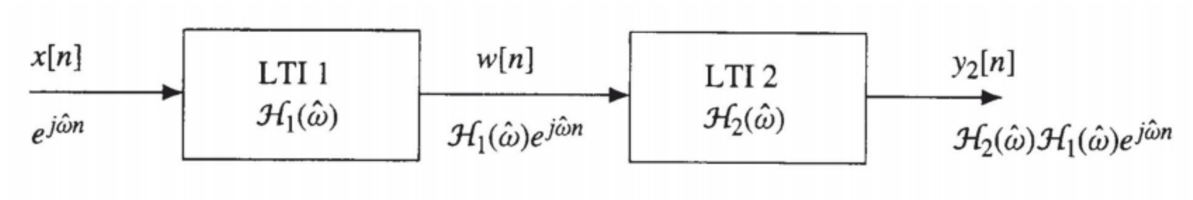
$y[n] = \sum b_k x[n-k]$

So a FIR filter with weights: $b = (1, 0.5, 0.33, 0.25)$

$y[n] = b_0 x[n-3] + b_1 x[n-2] + b_2 x[n-1] + b_3 x[n]$

$y[n] = x[n-3] + 0.5x[n-2] + 0.33x[n-1] + 0.25x[n]$

## Cascaded FIR Filters



$x[n]$
$e^{j\hat{\omega}n}$
LTI 1
$\mathcal{H}_1(\hat{\omega})$
$w[n]$
$\mathcal{H}_1(\hat{\omega})e^{j\hat{\omega}n}$
LTI 2
$\mathcal{H}_2(\hat{\omega})$
$y_2[n]$
$\mathcal{H}_2(\hat{\omega})\mathcal{H}_1(\hat{\omega})e^{j\hat{\omega}n}$

$$\begin{array}{ccc} \textit{Convolution} & \Longleftrightarrow & \textit{Multiplication} \\ h_1[n] * h_2[n] & \Longleftrightarrow & \mathcal{H}_1(\hat{\omega})\mathcal{H}_2(\hat{\omega}) \end{array}$$

# 6. Discrete-Time Fourier Transform (DTFT)

Generalizing the concept of frequency response, we get the Discrete Time Fourier Transform (DTFT) which allows us to take the frequency spectrum of any signal, with the constraint that the signal needs to be infinite, so the DTFT is only computable analytically.

$$\mathcal{H}(\hat{\omega}) = frequency\_response\{h[n]\} = \sum_{k}^{M} h[n]e^{-j\hat{\omega}k}$$

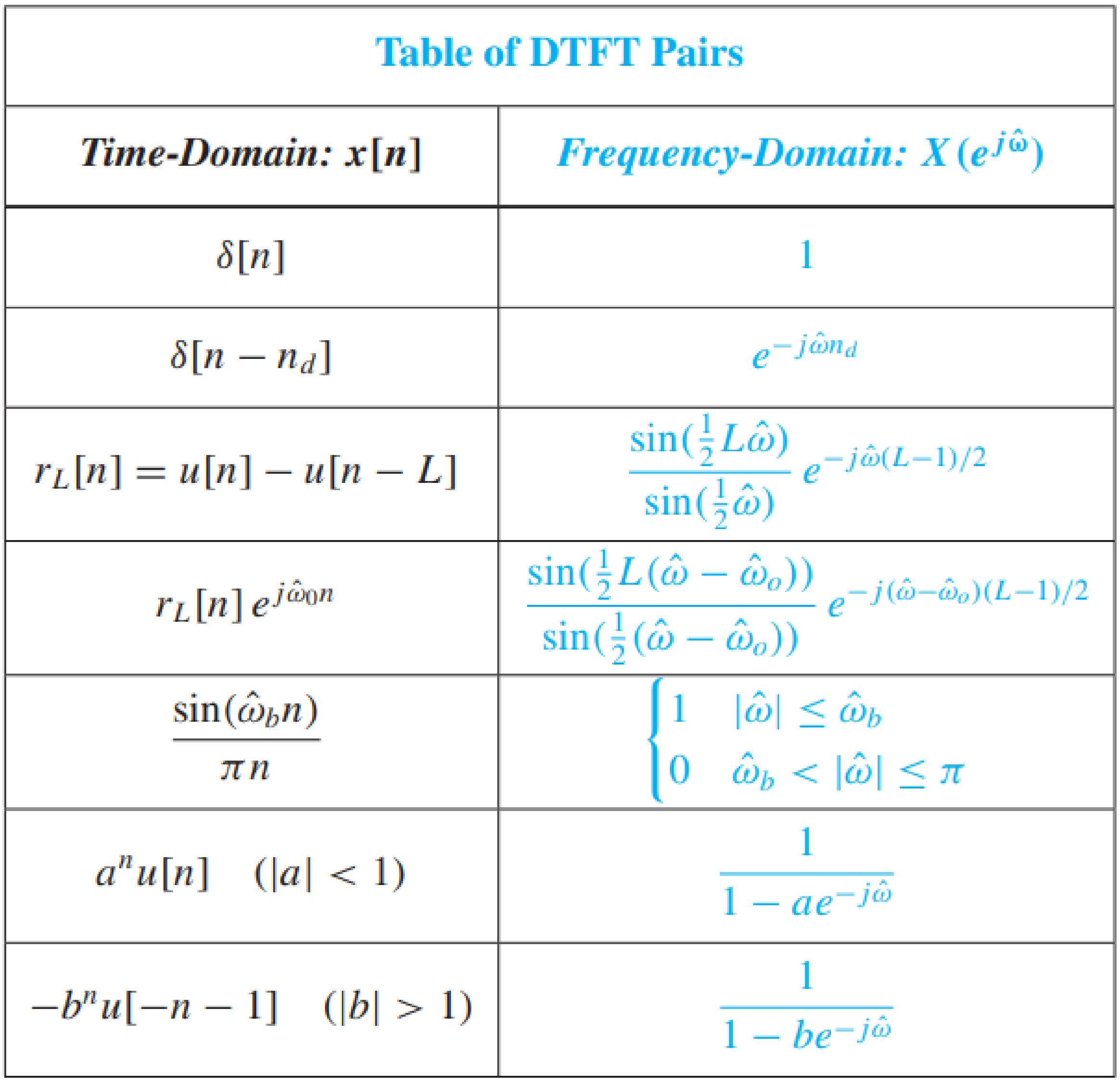
$\Downarrow$ generalizes to

$$X(\hat{\omega}) = DTFT\{x[n]\} = \sum_{n=-\infty}^{\infty} x[n]e^{-j\hat{\omega}n}$$

## Inverse DTFT

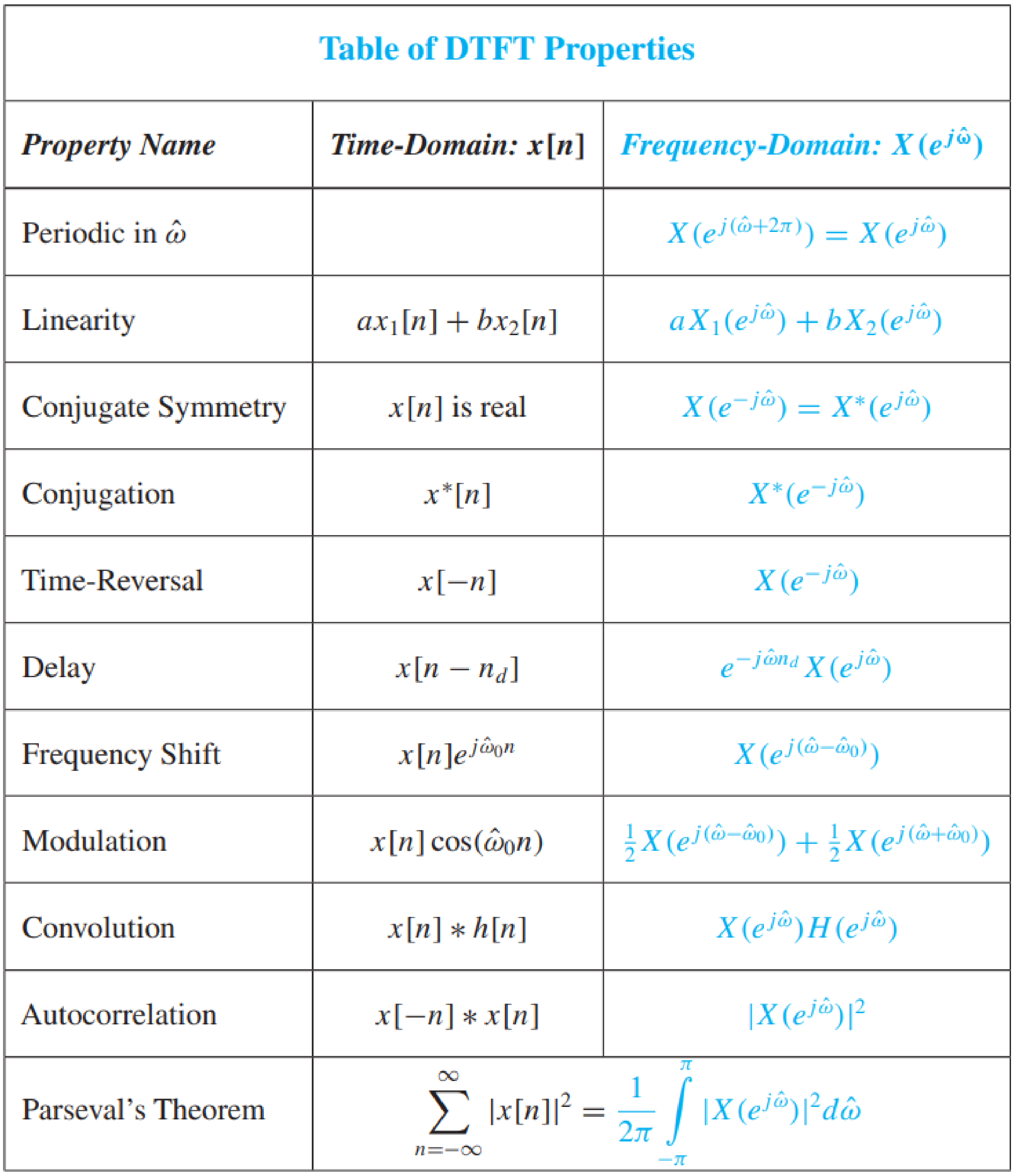
$$x[n] = IDTFT\{X(\hat{\omega})\} = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(\hat{\omega})e^{j\hat{\omega}n}\,d\hat{\omega}$$

## Common DTFT Table



| Table of DTFT Pairs | |
|---|---|
| ***Time-Domain:*** $x[n]$ | ***Frequency-Domain:*** $X(e^{j\hat{\omega}})$ |
| $\delta[n]$ | $1$ |
| $\delta[n-n_d]$ | $e^{-j\hat{\omega}n_d}$ |
| $r_L[n] = u[n] - u[n-L]$ | $\dfrac{\sin(\frac{1}{2}L\hat{\omega})}{\sin(\frac{1}{2}\hat{\omega})} e^{-j\hat{\omega}(L-1)/2}$ |
| $r_L[n]\, e^{j\hat{\omega}_0 n}$ | $\dfrac{\sin(\frac{1}{2}L(\hat{\omega}-\hat{\omega}_o))}{\sin(\frac{1}{2}(\hat{\omega}-\hat{\omega}_o))} e^{-j(\hat{\omega}-\hat{\omega}_o)(L-1)/2}$ |
| $\dfrac{\sin(\hat{\omega}_b n)}{\pi n}$ | $\begin{cases} 1 & \lvert\hat{\omega}\rvert \le \hat{\omega}_b \\ 0 & \hat{\omega}_b < \lvert\hat{\omega}\rvert \le \pi \end{cases}$ |
| $a^n u[n] \quad (\lvert a\rvert < 1)$ | $\dfrac{1}{1 - ae^{-j\hat{\omega}}}$ |
| $-b^n u[-n-1] \quad (\lvert b\rvert > 1)$ | $\dfrac{1}{1 - be^{-j\hat{\omega}}}$ |

## Properties of DTFT Table



| Table of DTFT Properties | | |
|---|---|---|
| ***Property Name*** | ***Time-Domain:*** $x[n]$ | ***Frequency-Domain:*** $X(e^{j\hat{\omega}})$ |
| Periodic in $\hat{\omega}$ | | $X(e^{j(\hat{\omega}+2\pi)}) = X(e^{j\hat{\omega}})$ |
| Linearity | $ax_1[n] + bx_2[n]$ | $aX_1(e^{j\hat{\omega}}) + bX_2(e^{j\hat{\omega}})$ |
| Conjugate Symmetry | $x[n]$ is real | $X(e^{-j\hat{\omega}}) = X^*(e^{j\hat{\omega}})$ |
| Conjugation | $x^*[n]$ | $X^*(e^{-j\hat{\omega}})$ |
| Time-Reversal | $x[-n]$ | $X(e^{-j\hat{\omega}})$ |
| Delay | $x[n-n_d]$ | $e^{-j\hat{\omega}n_d}X(e^{j\hat{\omega}})$ |
| Frequency Shift | $x[n]e^{j\hat{\omega}_0 n}$ | $X(e^{j(\hat{\omega}-\hat{\omega}_0)})$ |
| Modulation | $x[n]\cos(\hat{\omega}_0 n)$ | $\frac{1}{2}X(e^{j(\hat{\omega}-\hat{\omega}_0)}) + \frac{1}{2}X(e^{j(\hat{\omega}+\hat{\omega}_0)})$ |
| Convolution | $x[n] * h[n]$ | $X(e^{j\hat{\omega}})H(e^{j\hat{\omega}})$ |
| Autocorrelation | $x[-n] * x[n]$ | $\lvert X(e^{j\hat{\omega}})\rvert^2$ |
| Parseval's Theorem | $\sum_{n=-\infty}^{\infty} \lvert x[n]\rvert^2 = \frac{1}{2\pi}\int_{-\pi}^{\pi} \lvert X(e^{j\hat{\omega}})\rvert^2 d\hat{\omega}$ | |

## Energy of a Signal

$$energy = \sum_{-\infty}^{\infty} x[n]^2 = \frac{1}{2\pi}\int_{-\pi}^{\pi} |X(\hat{\omega})|^2 d\hat{\omega}$$

## Ideal Filters

Ideal low-pass:

$$\mathcal{H}_{lp}(\hat{\omega}) = \begin{cases} 1 & |\hat{\omega}| \le \hat{\omega}_{cut} \\ 0 & \hat{\omega}_{cut} < |\hat{\omega}_{cut}| \le \pi \end{cases}$$

$$h_{lp}[n] = IDTFT\{\mathcal{H}_{lp}(\hat{\omega})\} = \frac{\sin(\hat{\omega}_{cut} n)}{\pi n} \qquad -\infty < n < \infty$$

This is very similar to the sinc function.

Ideal high-pass:

$$\mathcal{H}_{hp}(\hat{\omega}) = \begin{cases} 0 & |\hat{\omega}| \leq \hat{\omega}_{cut} \\ 1 & \hat{\omega}_{cut} < |\hat{\omega}_{cut}| \leq \pi \end{cases}$$

$$h_{hp}[n] = IDTFT\{\mathcal{H}_{hp}(\hat{\omega})\} = \delta[n] - h_{lp}[n] = \delta[n] - \frac{\sin(\hat{\omega}_{cut} n)}{\pi n} \qquad -\infty < n < \infty$$

# 7. Discrete Fourier Transform (DFT) & Fast Fourier Transform (FFT)

We can take the Fourier transform of a finite signal by discretizing the frequency spectrum. This gives us the Discrete Fourier Transform (DFT).

$$X[k] = DFT\{x[n]\} = \sum_{n=0}^{N-1} x[n] e^{-j(2\pi/N)kn}$$

Where:

$$k \in [0, N-1]$$

I.e. for a finite signal of size $N$, the DFT produces a spectrum of size $N$. The entries in the spectrum are called "bins". They are complex numbers, where the magnitude of the number is the amplitude of that frequency, and the angle is the phase.

The bigger the signal, the more fine-grained the frequency spectrum becomes, this is a practical example of the *Heisenberg uncertainty principle.*

## Inverse DFT

$$x[n] = IDFT\{X[k]\} = \frac{1}{N} \sum_{n=0}^{N-1} x[n] e^{j(2\pi/N)kn}$$

Notice how the formula for the inverse DFT is almost identical to the normal DFT.

## Table of Common DFTs

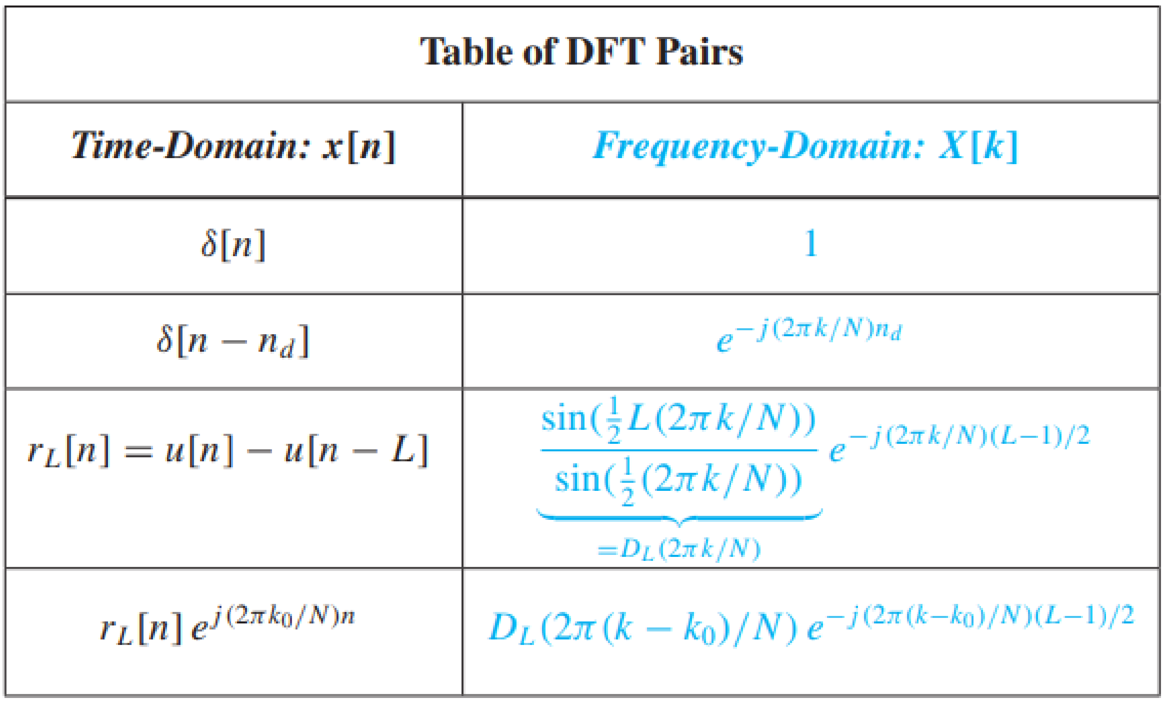
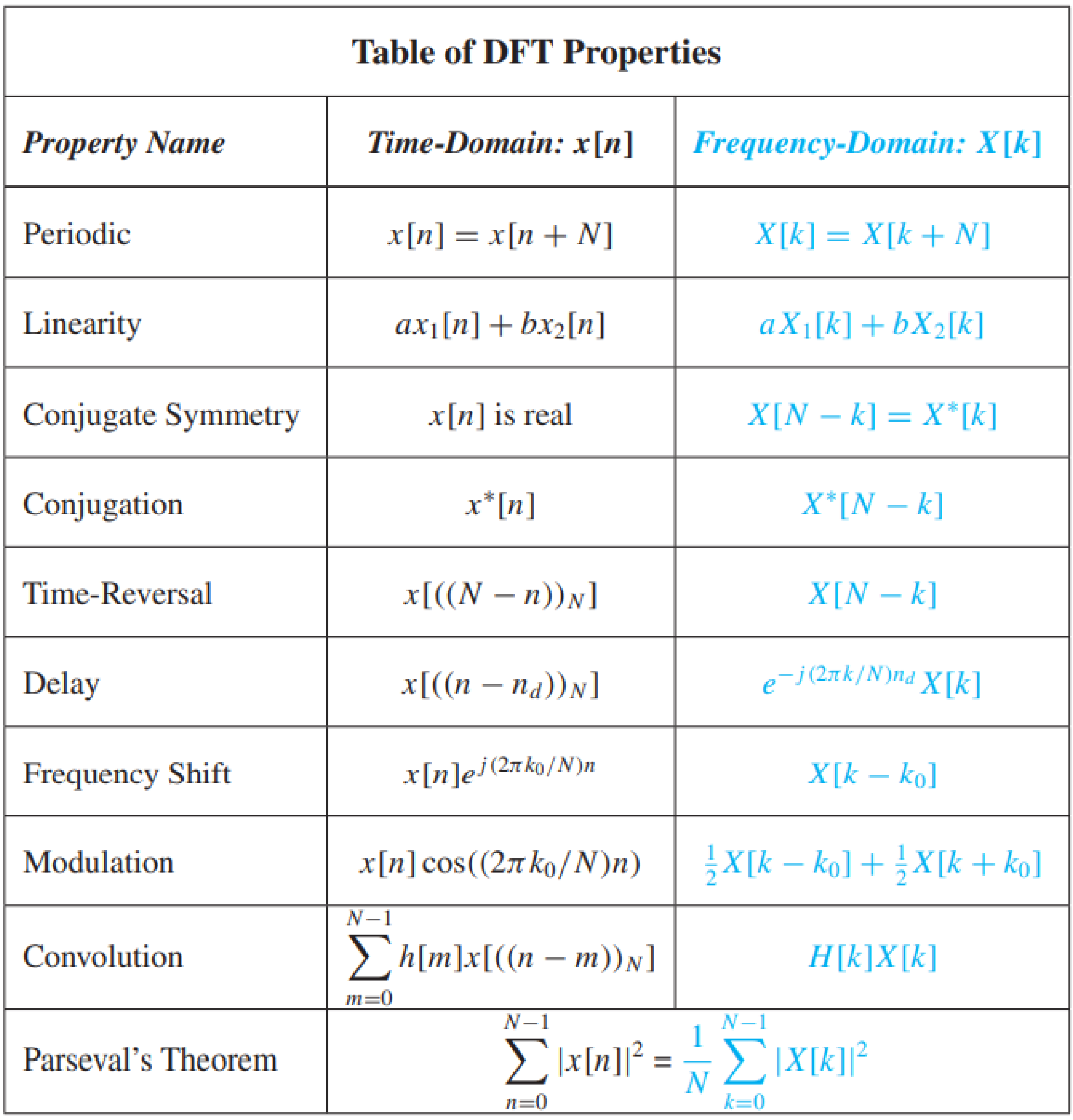


**Table of DFT Pairs**

| *Time-Domain:* $x[n]$ | *Frequency-Domain:* $X[k]$ |
|---|---|
| $\delta[n]$ | $1$ |
| $\delta[n - n_d]$ | $e^{-j(2\pi k/N)n_d}$ |
| $r_L[n] = u[n] - u[n-L]$ | $\underbrace{\frac{\sin(\frac{1}{2}L(2\pi k/N))}{\sin(\frac{1}{2}(2\pi k/N))}}_{=D_L(2\pi k/N)} e^{-j(2\pi k/N)(L-1)/2}$ |
| $r_L[n]\, e^{j(2\pi k_0/N)n}$ | $D_L(2\pi(k-k_0)/N)\, e^{-j(2\pi(k-k_0)/N)(L-1)/2}$ |

## Properties of DFTs



| Table of DFT Properties | | |
|---|---|---|
| ***Property Name*** | ***Time-Domain:*** $x[n]$ | ***Frequency-Domain:*** $X[k]$ |
| Periodic | $x[n] = x[n+N]$ | $X[k] = X[k+N]$ |
| Linearity | $ax_1[n] + bx_2[n]$ | $aX_1[k] + bX_2[k]$ |
| Conjugate Symmetry | $x[n]$ is real | $X[N-k] = X^*[k]$ |
| Conjugation | $x^*[n]$ | $X^*[N-k]$ |
| Time-Reversal | $x[((N-n))_N]$ | $X[N-k]$ |
| Delay | $x[((n-n_d))_N]$ | $e^{-j(2\pi k/N)n_d} X[k]$ |
| Frequency Shift | $x[n]e^{j(2\pi k_0/N)n}$ | $X[k-k_0]$ |
| Modulation | $x[n]\cos((2\pi k_0/N)n)$ | $\frac{1}{2}X[k-k_0] + \frac{1}{2}X[k+k_0]$ |
| Convolution | $\sum_{m=0}^{N-1} h[m]x[((n-m))_N]$ | $H[k]X[k]$ |
| Parseval's Theorem | $\sum_{n=0}^{N-1} \|x[n]\|^2 = \frac{1}{N}\sum_{k=0}^{N-1} \|X[k]\|^2$ | |

## Computing DFT by Sampling DTFT

If you need to calculate the N-point DFT of a signal with a well known DTFT, for example the unit impulse $\delta$, it is easier to sample the DTFT rather than compute the entire DFT sum. To do so, assume we have the DTFT, then you can turn it into a DFT by substituting $\hat{\omega} = (2\pi k/N)$, where $k \in [0, N-1]$ is the index of the DFT, so:

$$DFT\{x\} = DTFT\{x\}(2\pi k/N)$$
$$X[k] = X(2\pi k/N)$$

## Fast Fourier Transform (FFT)

**If you just want to check your answers, here's Matlab for that**

```
clear
a = [1 0 0 0 0 0 0 0 0 0 0]
A = fft(a)
% 1 1 1 1 1 1 1 1 1 1 1
stem(A) % plots FFT spectrum
```

The naïve DFT has a time complexity of $\mathcal{O}(N^2)$, a better algorithm called the Fast Fourier Transform has time complexity $\mathcal{O}(N \log N)$.

Notably, the FFT only works if $N$ is a power of 2, i.e. $log_2 N \in \mathbb{N}$

The algorithm is recursive:

- Base case: size of the input array is 2, DFT is trivial.

$$X_2[0] = x_2[0] + x_2[1]$$

$$X_2[1] = x_2[0] + e^{-j2\pi/2}\, x_2[1] = x_2[0] - x_2[1]$$

- Recursive case: perform the DFT of the vector formed by all the even entries and the vector formed by all the odd entries (recursively), then combine them (see pseudocode)

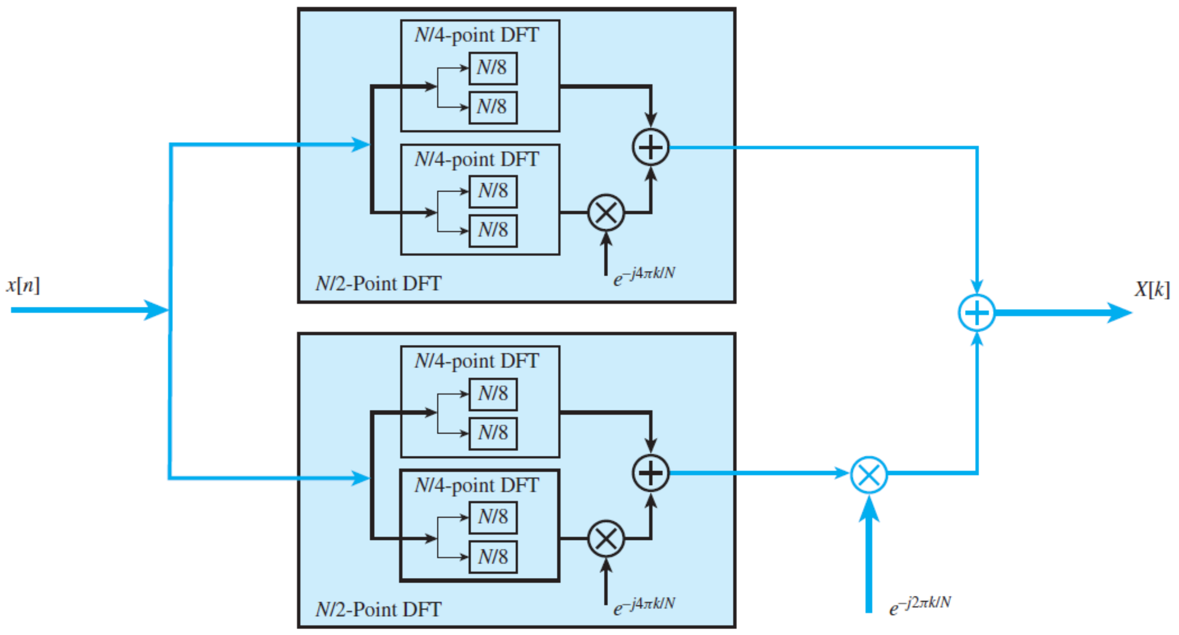
```
# x is an array of floating point numbers, with a size that is a power of 2
def fft(x):
    # instantiate the output array, with the same size as the input
    X = [0.0] * len(x)

    # base case: the array is of size 2, then the DFT is trivial
    if len(x) == 2:
        X[0] = x[0] + x[1]
        X[1] = x[0] - x[1]
        return X

    # recursive case: take the DFT of the arrays formed by the odd terms and the
even
    # terms, and combine them together
    else:
        fft_even = fft(x[0::2])  # the [0::2] slice operator takes every other
entry
                                 # starting from 0, so all the even entries: 0,
2, 4 ...
        fft_odd  = fft(x[1::2])  # the [1::2] slice operator takes every other
entry
                                 # starting from 1, so all the odd entries: 1,
3, 5 ...

        # iterate over output array
        for k in range(len(x)//2):
            # first half of output:
            X[k]             = fft_even[k] + fft_odd[k] * exp(-2j*pi*k/N)
            # second half of the output
            X[k + len(x)//2] = fft_even[k] - fft_odd[k] * exp(-2j*pi*k/N)
        return X
```

For an 8-point DFT, the algorithm can be summarized by this block diagram:



The inverse FFT (IFFT) is almost identical, the only differences are: that the sign of `j` in the the `exp(2j*pi*k/N)` term is flipped (so `exp(2j*pi*k/N)` instead `exp(-2j*pi*k/N)`); and that each entry of the output vector is divided by $N$ at the end.

# 8. Z-Transform

The z-transform transforms a vector $x$ into a complex polynomial $p(z)$. Every sample in $x$ becomes a coefficient of a term of the polynomial. The exponent of the term is a negative number, which is the negative of the index of the input vector its coefficient is taken from:

$$X(z) = \sum_{n=0}^{N-1} x[n] z^{-n}$$

Or the generalized form, for an infinite signal $x$:

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

Example:

$$\begin{matrix} ( & 1 & 2 & 3 & 5 & 8 & ) \\ & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\ & 1 & +2z^{-1} & +3z^{-2} & +5z^{-3} & +8z^{-4} & \end{matrix}$$

## Z-Transform of Time Delay

z-transforms are useful because they describe time delays very easily.

This is a time delay by $d$ samples in time-domain:

$x[n] \rightarrow x[n-d]$

The same in z-domain:

$X(z) \rightarrow X(z)z^{-d}$

## Z-Transform of FIR Filters

$$y[n] = \sum_{k=0}^{M} b_k x[n-k]$$

$\Downarrow$

$$Y(z) = \left( \sum_{k=0}^{M} b_k z^{-k} \right) X(z)$$

The part in parentheses is written as $H(z)$ and is called the system-function.

## Z-Transform of IIR Filters

$$y[n] = \sum_{l=1}^{N} a_l y[n-l] + \sum_{k=0}^{M} b_k x[n-k]$$

$$H(z) = \frac{\displaystyle\sum_{k=0}^{M} b_k z^{-k}}{1 - \displaystyle\sum_{\ell=1}^{N} a_\ell z^{-\ell}}$$

Notice that the $a$ coefficients start from 1 rather than 0, this is because you can't refer to $y[n]$ in the definition of $y[n]$. Also the sign is inverted for some reason.

## Common z-transforms Table

| | Signal | Transform |
|---|---|---|
| Unit impulse signal | $\delta[n]$ | $1$ |
| Unit step signal | $u[n]$ | $\frac{1}{1-z^{-1}}$ |
| | $-u[-n-1]$ | $\frac{1}{1-z^{-1}}$ |
| Shifted unit impulse signal | $\delta[n-m]$ | $z^{-m}$ |
| | | |
| | $\alpha^n u[n]$ | $\frac{1}{1-\alpha z^{-1}}$ |
| | $-\alpha^n u[-n-1]$ | $\frac{1}{1-\alpha z^{-1}}$ |
| | $n\alpha^n u[n]$ | $\frac{\alpha z^{-1}}{(1-\alpha z^{-1})^2}$ |
| | $-n\alpha^n u[-n-1]$ | $\frac{\alpha z^{-1}}{(1-\alpha z^{-1})^2}$ |

## Properties Table

| Property | n-domain | z-domain | $\hat{\omega}$-domain |
|---|---|---|---|
| Linearity (superposition) | $ax_1[n] + bx_2[n]$ | $aX_1(z) + bX_2(z)$ | $aX_1(\hat{\omega}) + bX_2(\hat{\omega})$ |
| Time delay | $x[n - n_0]$ | $z^{-n_0}X(z)$ | $e^{-j\hat{\omega}n_0}X(\hat{\omega})$ |
| Convolution | $h[n] * x[n]$ | $H(z)X(z)$ | $H(\hat{\omega})X(\hat{\omega})$ |

# 9 & 10. Infinite Impulse Response (IIR) Filters

IIR filters have feedback, i.e. their difference equation contains references to the output of the function, i.e. it's a form of discrete differential equation:

$$y[n] = \sum_{l=1}^{N} a_l y[n-l] + \sum_{k=0}^{M} b_k x[n-k]$$

This is the **general differential equation** you have two sums: previous outputs and previous inputs, multiplied with some coefficients. In this case our signal $x$ and the output $y$ are assumed to be 0 for negative times, i.e. we start the computation at index 0, and whenever we go out of bounds, i.e. have a negative index, we assume the value is 0.

If we take the z-transform we get:

$$Y(z) = X(z)H(z) = X(z)\frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \dots}{1 - a_1 z^{-1} - a_2 z^{-2} - \dots}$$

The roots of the **numerator** are called **zeros**

The roots of the **denominator** are called **poles**

Because IIR filters have a feedback path in them, they decay exponentially and their impulse response approaches 0 asymptotically, but only reaches 0 after infinite time, therefore they are called infinite impulse response filters (IIR)

## Stability

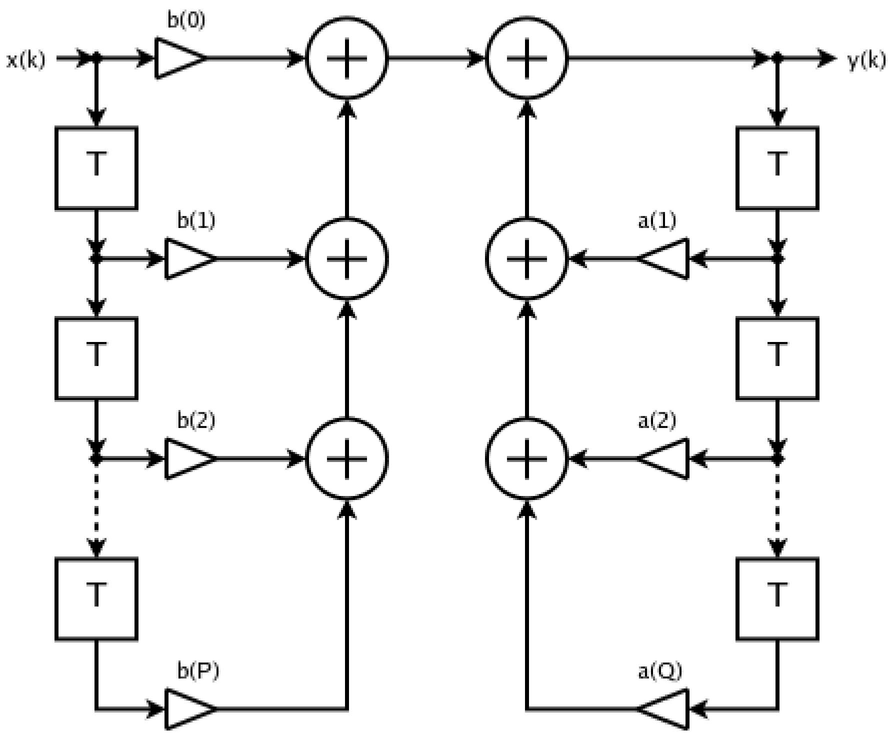
In order to be stable the following condition must be satisfied:

$$\left|\sum_{0}^{\infty} h[n]\right| < \infty$$

Same condition in z-domain is that all poles must be contained within the disc described by the unit circle.

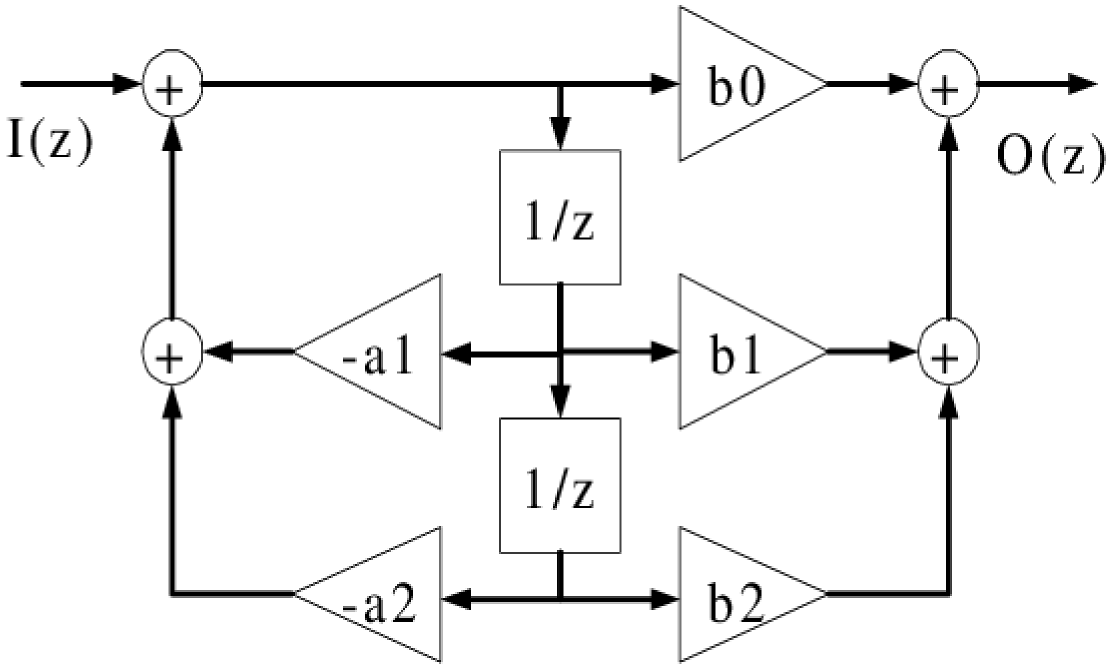
## Block Diagrams

This is a block diagram for a difference equation:



**Imagine that the blocks that have a T in them, have a $z^{-1}$ in them instead. I couldn't find a good picture**

Sometimes it is represented in this more compact form, they are equivalent. This form cuts down on the number of delay operations, but is harder to read.



## Frequency Response From z-domain

From z-domain system function:

$$H(z) = \frac{b_0 + b_1 z^{-1} + \dots}{1 - a_1 z^{-1} - \dots}$$

Then insert $z = e^{j\hat{\omega}}$ to get the frequency and phase response, i.e. the complex frequency response:

$$H(\hat{\omega}) = \frac{b_0 + b_1 e^{-j\hat{\omega}} + \dots}{1 - a_1 e^{-j\hat{\omega}} - \dots}$$

Then you have to find the following:

- (real) frequency response: $|H(\hat{\omega})|$
- phase response: $\angle H(\hat{\omega})$

### Frequency and Phase Response of 1st Order Filters

$$|H(\hat{\omega})|^2 = \frac{|b_0|^2 + |b_1|^2 + 2b_0 b_1 \cos(\hat{\omega})}{1 + |a_1|^2 - 2a_1 \cos(\hat{\omega})}$$

$$\angle H(\hat{\omega}) = atan\left(\frac{-b_1 \sin(\hat{\omega})}{b_0 + b_1 \cos(\hat{\omega})}\right) - atan\left(\frac{a_1 \sin(\hat{\omega})}{1 - a_1 \cos(\hat{\omega})}\right)$$

### Frequency and Phase Response of 2nd Order Filters

It's so unintuitive that you should just use Matlab (TODO)

## Design by Poles And Zeros

- *zero*: placing a zero **on** the unit circle, forces the frequencies at $\hat{\omega} = \angle zero$ to be 0, and frequencies nearby to become quieter. This can be used for wide band-stop filters.
- *pole*: placing a pole **near** but not on the unit circle, boosts the frequencies at $\hat{\omega} = \angle pole$ and the nearby frequencies to a lesser degree. This can be used for wide band-pass or peaking filters.

Then you take the inverse z-transform:

$$\begin{aligned} H(z) &= \frac{(s_1 - z^{-1})(s_2 - z^{-1})}{(1 - p_1 z^{-1})(1 - p_2 z^{-1})} \\ &= \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - a_1 z^{-1} - a_2 z^{-2}} \\ y[n] &= b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] - a_1 y[n-1] - a_2 y[n-2] \\ h[n] &= b_0 \delta[n] + b_1 \delta[n-1] + b_2 \delta[n-2] - a_1 h[n-1] - a_2 h[n-2] \end{aligned}$$

Notice that the book really likes to put negative signs on the $a$ coefficients, but like depends if you prefer to make it explicit that the $a$ coefficients are negative. Doesn't really matter.

### Closed-Form of IIR Impulse Response in n-domain

Do the inverse z-transform, this time using the tables of basic transforms and partial fraction expansion. This allows you to define the impulse response as a function without recursion:

First you find the zeros and poles, if you don't know them already:

$$\begin{aligned} H(z) &= \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}}{1 - a_1 z^{-1} - a_2 z^{-2}} \\ &= \frac{(s_1 - z^{-1})(s_2 - z^{-1})(s_3 - z^{-1})}{(1 - p_1 z^{-1})(1 - p_2 z^{-1})} \end{aligned}$$

Then you need to do partial fraction expansion, by using an illuminati operation:

$$H(z) = \frac{A}{1 - p_1 z^{-1}} + \frac{B}{1 - p_2 z^{-1}} + C$$

Now because of the superposition property of z-transforms, you can do the inverse z-transform on each by consulting a table:

$$h[n] = A(p_1^n)u[n] + B(p_2^n)u[n] + C\delta[n]$$

# Appendix A - Mat C,B,A Regneregler

Factoring 2nd degree polynomials:

$$ax^2 + bx + c = (r_1 - x)(r_2 - x)$$

$$r_1, r_2 = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Exponentials regneregler:

$$(e^a)^b = e^{ab}$$

$$e^a e^b = e^{a+b}$$

$$1 = e^0 = e^{j0} = e^{j0\hat{\omega}} = e^{0 \text{whatever the fuck you want lol}}$$

$$re^{j\varphi} = r\cos(\varphi) + jr\sin(\varphi) = a + jb$$

$$a + jb = \sqrt{a^2 + b^2}e^{j\,\text{atan}(b/a)}$$

Partial fraction decomposition:

The easy way: use Matlab's `residue` function:

```
b = [b2 b1 b0]
a = [a2 a1 1]
[r p k] = residue(b, a)
% r will contain the numerators of each fraction
% p will contain the roots of the denominator polynomials
```

The hard way: manually

$$\frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - a_1 z^{-1} - a_2 z^{-2}} = \frac{(s_1 - z^{-1})(s_2 - z^{-1})}{(1 - p_1 z^{-1})(1 - p_2 z^{-1})} = \frac{A}{1 - p_1 z^{-1}} + \frac{B}{1 - p_2 z^{-1}}$$

$$(s_1 - z^{-1})(s_2 - z^{-1}) = A(1 - p_2 z^{-1}) + B(1 - p_1 z^{-1})$$

Solve for A and B, usually by splitting the equation into a system of equations, if you can get the left hand side to look like (something + something) and the right hand side to be (something + something), then you split the equation containing just the left of the plus sign and just the right of the plus sign.

In the general case:

$$\frac{P}{(factor_1)(factor_2)\dots(factor_n)} = \frac{A}{factor_1} + \frac{B}{factor_2} + \dots + \frac{Z}{factor_n}$$

$$P = A(factor_2)\dots(factor_n) + B(factor_1)\dots(factor_n) + \dots + Z(factor_1)(factor_2)\dots$$

Essentially every letter is multiplied with every factor except the one below the letter.

Then solve for $A, B, \dots, Z$

# Appendix B - Matlab Reference

For loops

```
for i = min:step:max
    % code goes here
end
```

Functions

```
function return_variable = function_name(parameters)
    % code goes here
    return_variable = result
end
```

Unit step function

```
% unit step function
function x = unit(n)
if (n < 0)
    x = 0;
else
    x = 1;
end
end
```

Cartesian to polar

```
[theta rho] = cart2pol(real(n), imag(n))
```

Polar to cartesian

```
[x y] = pol2cart(angle(n), abs(n))
```

Partial fraction expansion / decomposition

```
[r p k] = residue([b2 b1 b0], [a2 a1 1])
% where r are the numerator values
% and p are the roots of each fraction's denominator
```

DFT

```
a = [1 0 0 0 0 0 0 0 0 0]
A = fft(a)
% 1 1 1 1 1 1 1 1 1 1
stem(A) % plots FFT spectrum
```

Equations manipulation

```
syms x y z     % defines symbols, necessary to define symbolic equations
eq = x == something   % this is how you assign an equation to a variable
solve(eq, var)    % solves symbolically
vpasolve(eq, var, [guess_min guess_max]) % solves numerically, guess is optional
isolate(eq, var)  % isolates variable (solve but sometimes works different)
subs(eq, old, new) % substitutes into equation
subs(eq)   % updates all variables in equation
simplify(eq) % simplify equation
collect(eq)  % factorizes as much as possible
expand(eq)   % multiplies everything out
```

Sampling

```
n = min:step:max
x = f(n)
```

Visualization / plotting

```
plot(x, y)  % where x and y are vectors
plotf(x, y) % where y is a function of x

% multiple plots on the same figure
plot(x, y)  % this and the next 2 are on the same figure
hold on
plot(x, y)
plot(x, y)
hold off
plotf(x, y) % here a new figure starts
hold on
plotf(x, y)
plotf(x, y)

% plot circle
function h = circle(x,y,r)
th = 0:pi/50:2*pi;
xunit = r * cos(th) + x;
yunit = r * sin(th) + y;
h = plot(xunit, yunit);
end

% pole-zero plot, requires function above to function
circle(0,0,1)
hold on
plot([x y], 'x', 'MarkerSize', 10, 'LineWidth', 2) % poles
plot([x y], 'o', 'MarkerSize', 10, 'LineWidth', 2) % zeros
axis equal
grid on
hold off
```

Array / vector initialization, concatentation

```
zeros(10) % array of ten zeros
ones(10)  % array of ten ones
[ones(5), zeros(5)] % concatenating arrays
```

symbolic sum / numeric sum

```
% symbolic sum
symsum(expr, index_variable, min, max)

% numeric sum
sum(array)
```