Digital Signal Processing

Markus Kuhn

Computer Laboratory, University of Cambridge

http://www.cl.cam.ac.uk/teaching/1314/DSP/

These notes are provided as an aid for following the lectures, and are not a substitute for attending

Michaelmas 2013 – Part II

Signals

- flow of information
- measured quantity that varies with time (or position)
- electrical signal received from a transducer (microphone, thermometer, accelerometer, antenna, etc.)
- electrical signal that controls a process

Continuous-time signals: voltage, current, temperature, speed, ...

Discrete-time signals: daily minimum/maximum temperature, lap intervals in races, sampled continuous signals, ...

Electronics (unlike optics) can only deal easily with time-dependent signals. Spatial signals, such as images, are typically first converted into a time signal with a scanning process (TV, fax, etc.).

Signal processing

Signals may have to be transformed in order to

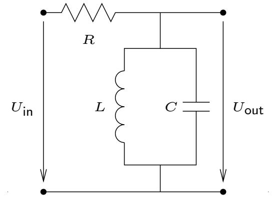
- amplify or filter out embedded information
- detect patterns
- prepare the signal to survive a transmission channel
- prevent interference with other signals sharing a medium
- undo distortions contributed by a transmission channel
- compensate for sensor deficiencies
- find information encoded in a different domain

To do so, we also need

- methods to measure, characterise, model and simulate transmission channels
- mathematical tools that split common channels and transformations into easily manipulated building blocks

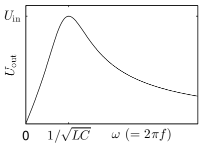
Analog electronics

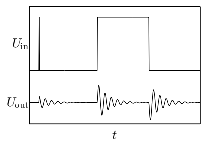
Passive networks (resistors, capacitors, inductances, crystals, SAW filters), non-linear elements (diodes, ...), (roughly) linear operational amplifiers



Advantages:

- passive networks are highly linear over a very large dynamic range and large bandwidths
- analog signal-processing circuits require little or no power
- analog circuits cause little additional interference





$$\frac{U_{\text{in}} - U_{\text{out}}}{R} = \frac{1}{L}\int_{-\infty}^{t} U_{\text{out}}\,\mathrm{d}\tau + C\frac{\mathrm{d}U_{\text{out}}}{\mathrm{d}t}$$

Digital signal processing

Analog/digital and digital/analog converter, CPU, DSP, ASIC, FPGA.

Advantages:

- noise is easy to control after initial quantization
- highly linear (within limited dynamic range)
- complex algorithms fit into a single chip
- flexibility, parameters can easily be varied in software
- digital processing is insensitive to component tolerances, aging, environmental conditions, electromagnetic interference

But:

- discrete-time processing artifacts (aliasing)
- can require significantly more power (battery, cooling)
- digital clock and switching cause interference

Some DSP applications

communication systems
modulation/demodulation, channel equalization, echo cancellation

consumer electronics
perceptual coding of audio and video (DAB, DVB, DVD), speech synthesis, speech recognition

music
synthetic instruments, audio effects, noise reduction

medical diagnostics
magnetic-resonance and ultrasonic imaging, X-ray computed tomography, ECG, EEG, MEG, AED, audiology

geophysics
seismology, oil exploration

astronomy
VLBI, speckle interferometry

transportation
radar, radio navigation

security
steganography, digital watermarking, biometric identification, surveillance systems, signals intelligence, electronic warfare

engineering
control systems, feature extraction for pattern recognition, sensor-data evaluation

Objectives

By the end of the course, you should be able to

- apply basic properties of time-invariant linear systems
- understand sampling, aliasing, convolution, filtering, the pitfalls of spectral estimation
- explain the above in time and frequency domain representations
- use filter-design software
- visualise and discuss digital filters in the z-domain
- use the FFT for convolution, deconvolution, filtering
- implement, apply and evaluate simple DSP applications in MATLAB
- apply transforms that reduce correlation between several signal sources
- understand the basic principles of several widely-used modulation and image-coding techniques.

Textbooks

- R.G. Lyons: *Understanding digital signal processing.* 3rd ed., Prentice-Hall, 2010. (£68)
- A.V. Oppenheim, R.W. Schafer: *Discrete-time signal processing.* 3rd ed., Prentice-Hall, 2007. (£47)
- J. Stein: *Digital signal processing – a computer science perspective.* Wiley, 2000. (£133)
- S.W. Smith: *Digital signal processing – a practical guide for engineers and scientists.* Newness, 2003. (£48)
- K. Steiglitz: *A digital signal processing primer – with applications to digital audio and computer music.* Addison-Wesley, 1996. (£67)
- Sanjit K. Mitra: *Digital signal processing – a computer-based approach.* McGraw-Hill, 2002. (£38)

Units and decibel

Communications engineers often use logarithmic units:

- Quantities often vary over many orders of magnitude $\rightarrow$ difficult to agree on a common SI prefix (nano, micro, milli, kilo, etc.)
- Quotient of quantities (amplification/attenuation) usually more interesting than difference
- Signal strength usefully expressed as field quantity (voltage, current, pressure, etc.) or power, but quadratic relationship between these two ($P = U^2/R = I^2R$) rather inconvenient
- Perception is logarithmic (Weber/Fechner law $\rightarrow$ slide 163)

Plus: Using magic special-purpose units has its own odd attractions ($\rightarrow$ typographers, navigators)

Neper (Np) denotes the natural logarithm of the quotient of a field quantity F and a reference value F_0. (rarely used today)

Bel (B) denotes the base-10 logarithm of the quotient of a power P and a reference power P_0. Common prefix: 10 decibel (dB) = 1 bel.

Decibel

Where P is some power and P_0 a 0 dB reference power, or equally where F is a field quantity and F_0 the corresponding reference level:

$$10\ \text{dB} \cdot \log_{10} \frac{P}{P_0} = 20\ \text{dB} \cdot \log_{10} \frac{F}{F_0}$$

Common reference values are indicated with a suffix after "dB":

$$\begin{aligned}
0\ \text{dBW} &= 1\ \text{W} \\
0\ \text{dBm} &= 1\ \text{mW} = -30\ \text{dBW} \\
0\ \text{dB}\mu\text{V} &= 1\ \mu\text{V} \\
0\ \text{dB}_{\text{SPL}} &= 20\ \mu\text{Pa} \quad \text{(sound pressure level)} \\
0\ \text{dB}_{\text{SL}} &= \text{perception threshold (sensation limit)} \\
0\ \text{dBFS} &= \text{full scale (clipping limit of analog/digital converter)}
\end{aligned}$$

Remember:

$$\begin{aligned}
3\ \text{dB} &= \ \ 2\times \text{ power}, & 6\ \text{dB} &= \ \ 2\times \text{ voltage/pressure/etc.} \\
10\ \text{dB} &= 10\times \text{ power}, & 20\ \text{dB} &= 10\times \text{ voltage/pressure/etc.}
\end{aligned}$$

W.H. Martin: Decibel – the new name for the transmission unit. Bell Syst. Tech. J., Jan. 1929.

Sequences and systems

A *discrete sequence* $\{x_n\}_{n=-\infty}^{\infty}$ is a sequence of numbers

$$\ldots, x_{-2}, x_{-1}, x_0, x_1, x_2, \ldots$$

where x_n denotes the n-th number in the sequence ($n \in \mathbb{Z}$). A discrete sequence maps integer numbers onto real (or complex) numbers.

We normally abbreviate $\{x_n\}_{n=-\infty}^{\infty}$ to $\{x_n\}$, or to $\{x_n\}_n$ if the running index is not obvious. The notation is not well standardized. Some authors write $x[n]$ instead of x_n, others $x(n)$.

Where a discrete sequence $\{x_n\}$ samples a continuous function $x(t)$ as

$$x_n = x(t_\text{s} \cdot n) = x(n/f_\text{s}),$$

we call t_s the *sampling period* and $f_\text{s} = 1/t_\text{s}$ the *sampling frequency*.

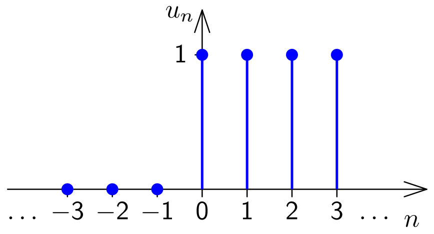
A *discrete system* T receives as input a sequence $\{x_n\}$ and transforms it into an output sequence $\{y_n\} = T\{x_n\}$:

$$\ldots, x_2, x_1, x_0, x_{-1}, \ldots \longrightarrow \boxed{\text{discrete system } T} \longrightarrow \ldots, y_2, y_1, y_0, y_{-1}, \ldots$$

Some simple sequences

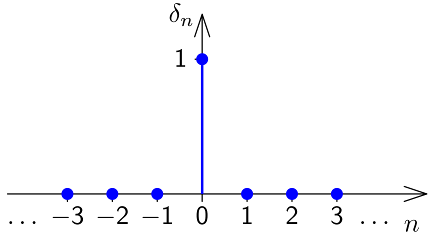
Unit-step sequence:

$$u_n = \begin{cases} 0, & n < 0 \\ 1, & n \geq 0 \end{cases}$$



Impulse sequence:

$$\delta_n = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases}$$
$$= u_n - u_{n-1}$$



Sinusoidial sequences

A cosine wave, frequency f, phase offset θ:

$$x(t) = \cos(2\pi f t + \theta)$$

Sampling it at sampling rate f_s results in the discrete sequence $\{x_n\}$:

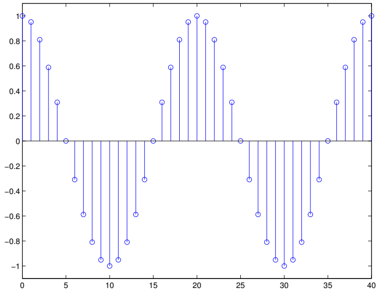
$$x_n = \cos(2\pi f n/f_\mathrm{s} + \theta)$$

MATLAB/Octave example:

```
n=0:40; fs=8000;
f=400; x=cos(2*pi*f*n/fs);
stem(n, x); ylim([-1.1 1.1])
```

This shows 41 samples ($\approx 1/200$ s $= 5$ ms) of an $f = 400$ Hz sine wave, sampled at $f_\mathrm{s} = 8$ kHz.

Exercise: Try $f = 0$, 1000, 2000, 3000, 4000, 5000 Hz. Try negative f. Try sine instead of cosine. Try adding θ phase offsets of $\pm\pi/4$, $\pm\pi/2$, and $\pm\pi$.



Properties of sequences

A sequence $\{x_n\}$ is

$$\textit{periodic} \Leftrightarrow \exists k > 0 : \forall n \in \mathbb{Z} : x_n = x_{n+k}$$

$$\textit{absolutely summable} \Leftrightarrow \sum_{n=-\infty}^{\infty} |x_n| < \infty$$

$$\textit{square summable} \Leftrightarrow \underbrace{\sum_{n=-\infty}^{\infty} |x_n|^2}_{\text{“energy”}} < \infty \quad \Leftrightarrow \quad \text{“energy signal”}$$

$$0 < \underbrace{\lim_{k\to\infty} \frac{1}{1+2k} \sum_{n=-k}^{k} |x_n|^2}_{\text{“average power”}} < \infty \quad \Leftrightarrow \quad \text{“power signal”}$$

This energy/power terminology reflects that if U is a voltage supplied to a load resistor R, then $P = UI = U^2/R$ is the power consumed, and $\int P(t)\,\mathrm{d}t$ the energy. It is used even if we drop physical units (e.g., volts) for simplicity in calculations.

Types of discrete systems

A *causal system* cannot look into the future:

$$y_n = f(x_n, x_{n-1}, x_{n-2}, \ldots)$$

A *memory-less system* depends only on the current input value:

$$y_n = f(x_n)$$

A *delay system* shifts a sequence in time:

$$y_n = x_{n-d}$$

T is a *time-invariant system* if for any d

$$\{y_n\} = T\{x_n\} \quad \Longleftrightarrow \quad \{y_{n-d}\} = T\{x_{n-d}\}.$$

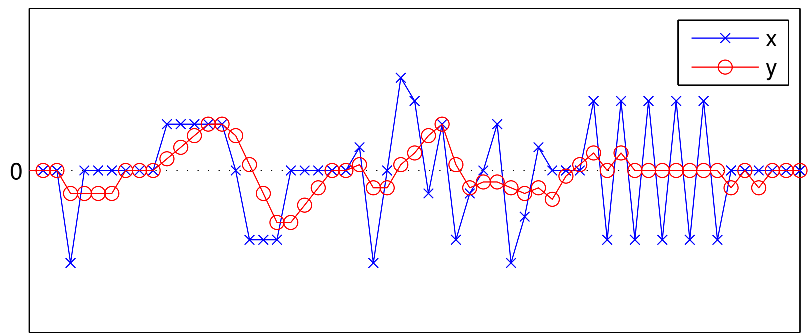
T is a *linear system* if for any pair of sequences $\{x_n\}$ and $\{x'_n\}$

$$T\{a \cdot x_n + b \cdot x'_n\} = a \cdot T\{x_n\} + b \cdot T\{x'_n\}.$$

Example: M-point moving average system

$$y_n = \frac{1}{M} \sum_{k=0}^{M-1} x_{n-k} = \frac{x_{n-M+1} + \cdots + x_{n-1} + x_n}{M}$$

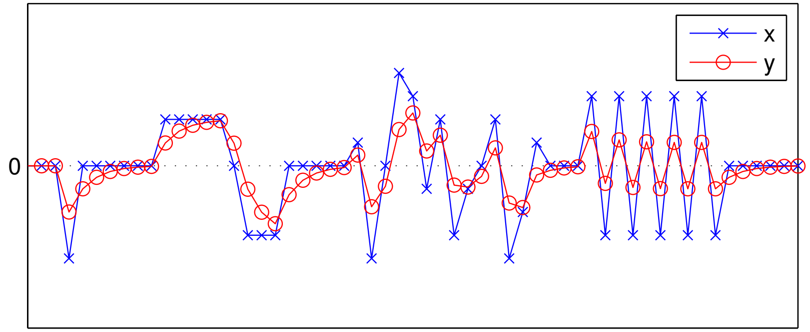
It is causal, linear, time-invariant, with memory. With $M = 4$:



Example: exponential averaging system

$$y_n = \alpha \cdot x_n + (1-\alpha)\cdot y_{n-1} = \alpha \sum_{k=0}^{\infty} (1-\alpha)^k \cdot x_{n-k}$$

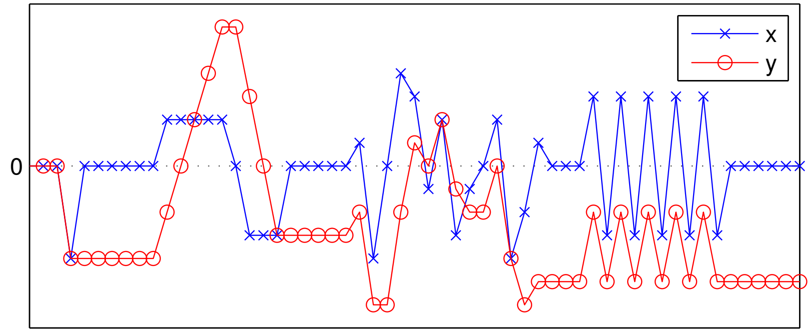
It is causal, linear, time-invariant, with memory. With $\alpha = \frac{1}{2}$:



Example: accumulator system

$$y_n = \sum_{k=-\infty}^{n} x_k$$

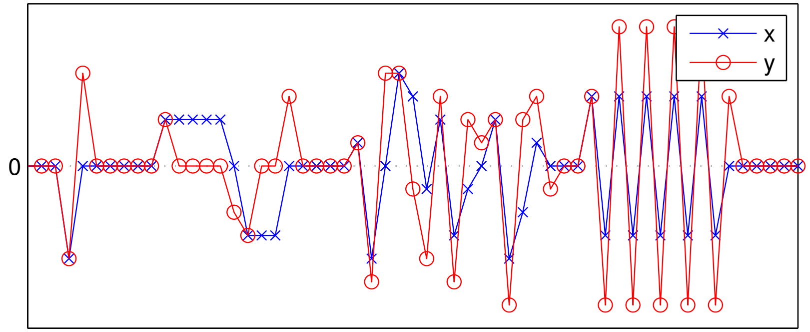
It is causal, linear, time-invariant, with memory.



Example: backward difference system

$$y_n = x_n - x_{n-1}$$

It is causal, linear, time-invariant, with memory.



Other examples

Time-invariant non-linear memory-less systems:

$$y_n = x_n^2, \quad y_n = \log_2 x_n, \quad y_n = \max\{\min\{\lfloor 256 x_n \rfloor, 255\}, 0\}$$

Linear but not time-invariant systems:

$$y_n = \begin{cases} x_n, & n \geq 0 \\ 0, & n < 0 \end{cases} = x_n \cdot u_n$$

$$y_n = x_{\lfloor n/4 \rfloor}$$

$$y_n = x_n \cdot \Re(e^{\omega j n})$$

Linear time-invariant non-causal systems:

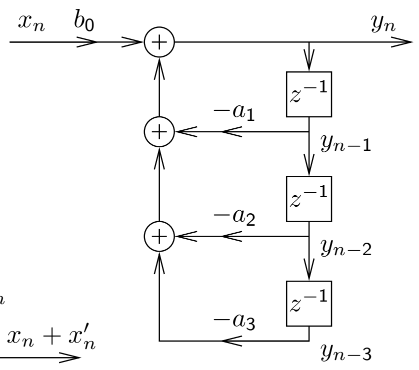
$$y_n = \frac{1}{2}(x_{n-1} + x_{n+1})$$

$$y_n = \sum_{k=-9}^{9} x_{n+k} \cdot \frac{\sin(\pi k \omega)}{\pi k \omega} \cdot [0.5 + 0.5 \cdot \cos(\pi k / 10)]$$

Constant-coefficient difference equations

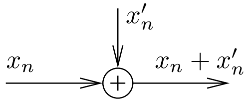
Of particular practical interest are causal linear time-invariant systems of the form

$$y_n = b_0 \cdot x_n - \sum_{k=1}^{N} a_k \cdot y_{n-k}$$

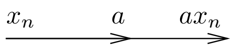


Block diagram representation of sequence operations:

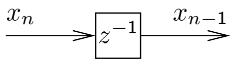
Addition:



Multiplication by constant:



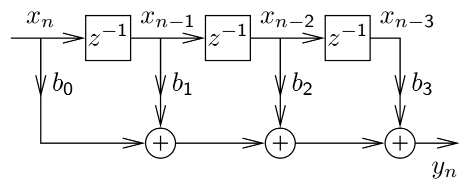
Delay:



The a_k and b_m are constant coefficients.

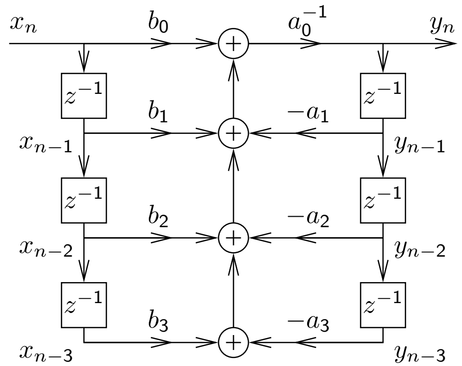
or

$$y_n = \sum_{m=0}^{M} b_m \cdot x_{n-m}$$



or the combination of both:

$$\sum_{k=0}^{N} a_k \cdot y_{n-k} = \sum_{m=0}^{M} b_m \cdot x_{n-m}$$



The MATLAB function `filter` is an efficient implementation of the last variant.

Convolution

Another example of a LTI systems is

$$y_n = \sum_{k=-\infty}^{\infty} a_k \cdot x_{n-k}$$

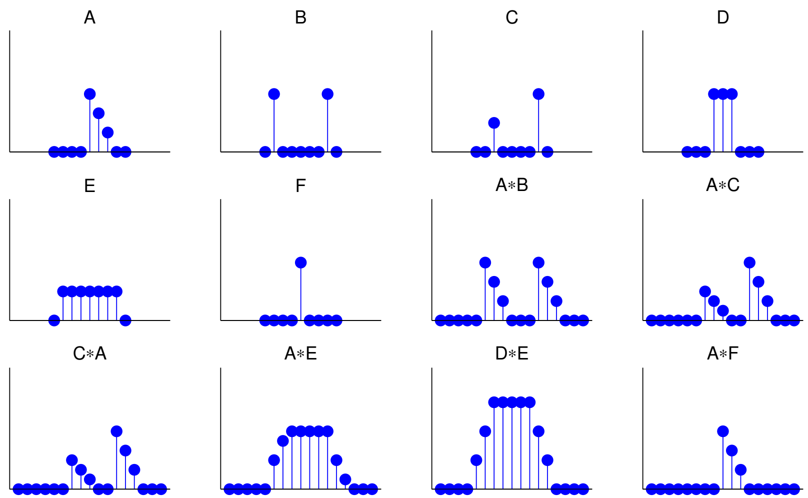
where $\{a_k\}$ is a suitably chosen sequence of coefficients.

This operation over sequences is called *convolution* and is defined as

$$\{p_n\} * \{q_n\} = \{r_n\} \quad \Longleftrightarrow \quad \forall n \in \mathbb{Z} : r_n = \sum_{k=-\infty}^{\infty} p_k \cdot q_{n-k}.$$

If $\{y_n\} = \{a_n\} * \{x_n\}$ is a representation of an LTI system T, with $\{y_n\} = T\{x_n\}$, then we call the sequence $\{a_n\}$ the *impulse response* of T, because $\{a_n\} = T\{\delta_n\}$.

Convolution examples



Properties of convolution

For arbitrary sequences $\{p_n\}$, $\{q_n\}$, $\{r_n\}$ and scalars a, b:

- Convolution is associative

$$(\{p_n\} * \{q_n\}) * \{r_n\} = \{p_n\} * (\{q_n\} * \{r_n\})$$

- Convolution is commutative

$$\{p_n\} * \{q_n\} = \{q_n\} * \{p_n\}$$

- Convolution is linear

$$\{p_n\} * \{a \cdot q_n + b \cdot r_n\} = a \cdot (\{p_n\} * \{q_n\}) + b \cdot (\{p_n\} * \{r_n\})$$

- The impulse sequence (slide 12) is neutral under convolution

$$\{p_n\} * \{\delta_n\} = \{\delta_n\} * \{p_n\} = \{p_n\}$$

- Sequence shifting is equivalent to convolving with a shifted impulse

$$\{p_{n-d}\} = \{p_n\} * \{\delta_{n-d}\}$$

Proof: all LTI systems just apply convolution

Any sequence $\{x_n\}$ can be decomposed into a weighted sum of shifted impulse sequences:

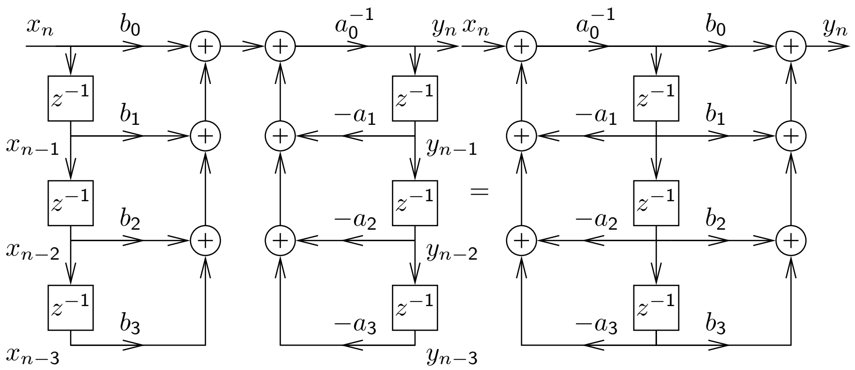
$$\{x_n\} = \sum_{k=-\infty}^{\infty} x_k \cdot \{\delta_{n-k}\}$$

Let's see what happens if we apply a linear$^{(*)}$ time-invariant$^{(**)}$ system T to such a decomposed sequence:

$$\begin{aligned} T\{x_n\} &= T\left(\sum_{k=-\infty}^{\infty} x_k \cdot \{\delta_{n-k}\}\right) \stackrel{(*)}{=} \sum_{k=-\infty}^{\infty} x_k \cdot T\{\delta_{n-k}\} \\ &\stackrel{(**)}{=} \sum_{k=-\infty}^{\infty} x_k \cdot \{\delta_{n-k}\} * T\{\delta_n\} = \left(\sum_{k=-\infty}^{\infty} x_k \cdot \{\delta_{n-k}\}\right) * T\{\delta_n\} \\ &= \{x_n\} * T\{\delta_n\} \qquad \text{q.e.d.} \end{aligned}$$

$\Rightarrow$ The impulse response $T\{\delta_n\}$ fully characterizes an LTI system.

Direct form I and II implementations



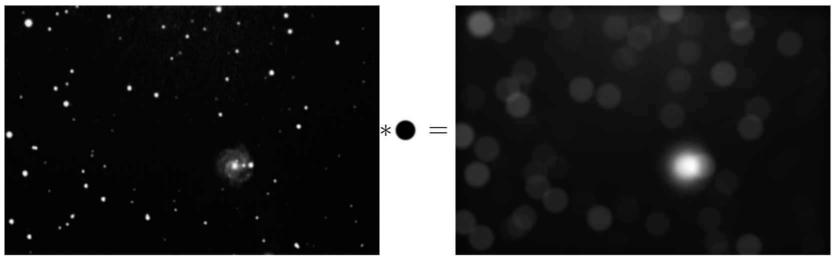
The block diagram representation of the constant-coefficient difference equation on slide 22 is called the *direct form I implementation*.

The number of delay elements can be halved by using the commutativity of convolution to swap the two feedback loops, leading to the *direct form II implementation* of the same LTI system.

These two forms are only equivalent with ideal arithmetic (no rounding errors and range limits).

Convolution: optics example

If a projective lens is out of focus, the blurred image is equal to the original image convolved with the aperture shape (e.g., a filled circle):

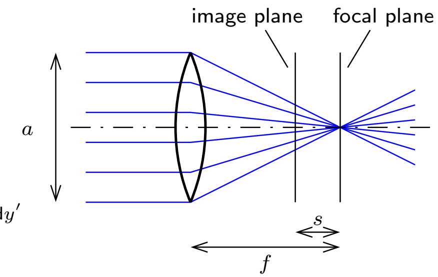


Point-spread function h (disk, $r = \frac{as}{2f}$):

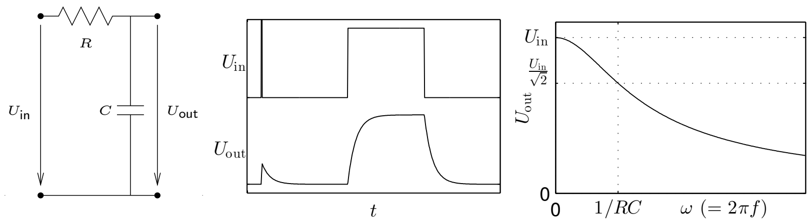
$$h(x, y) = \begin{cases} \frac{1}{r^2\pi}, & x^2 + y^2 \le r^2 \\ 0, & x^2 + y^2 > r^2 \end{cases}$$

Original image I, blurred image $B = I * h$, i.e.

$$B(x, y) = \iint I(x-x', y-y') \cdot h(x', y') \cdot \mathrm{d}x' \mathrm{d}y'$$



Convolution: electronics example



Any passive network (R, L, C) convolves its input voltage U_{in} with an *impulse response function* h, leading to $U_{\text{out}} = U_{\text{in}} * h$, that is

$$U_{\text{out}}(t) = \int_{-\infty}^{\infty} U_{\text{in}}(t-\tau) \cdot h(\tau) \cdot \mathrm{d}\tau$$

In the above example:

$$\frac{U_{\text{in}} - U_{\text{out}}}{R} = C \cdot \frac{\mathrm{d}U_{\text{out}}}{\mathrm{d}t}, \qquad h(t) = \begin{cases} \frac{1}{RC} \cdot \mathrm{e}^{\frac{-t}{RC}}, & t \geq 0 \\ 0, & t < 0 \end{cases}$$

Why are sine waves useful?

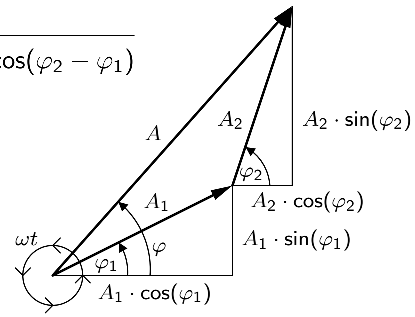
1) Adding together sine waves of equal frequency, but arbitrary amplitude and phase, results in another sine wave of the same frequency:

$$A_1 \cdot \sin(\omega t + \varphi_1) + A_2 \cdot \sin(\omega t + \varphi_2) = A \cdot \sin(\omega t + \varphi)$$

with

$$A = \sqrt{A_1^2 + A_2^2 + 2A_1A_2\cos(\varphi_2 - \varphi_1)}$$

$$\tan\varphi = \frac{A_1 \sin\varphi_1 + A_2 \sin\varphi_2}{A_1 \cos\varphi_1 + A_2 \cos\varphi_2}$$



Sine waves of any phase can be formed from sin and cos alone:

$$A \cdot \sin(\omega t + \varphi) = a \cdot \sin(\omega t) + b \cdot \cos(\omega t)$$

with $a = A \cdot \cos(\varphi)$, $b = A \cdot \sin(\varphi)$ and $A = \sqrt{a^2 + b^2}$, $\tan\varphi = \frac{b}{a}$.

Note: Convolution of a discrete sequence $\{x_n\}$ with another sequence $\{y_n\}$ is nothing but adding together scaled and delayed copies of $\{x_n\}$. (Think of $\{y_n\}$ decomposed into a sum of impulses.)

If $\{x_n\}$ is a sampled sine wave of frequency f, so is $\{x_n\} * \{y_n\}$!

$\Longrightarrow$ **Sine-wave sequences form a family of discrete sequences that is closed under convolution with arbitrary sequences.**

The same applies for continuous sine waves and convolution.

2) Sine waves are orthogonal to each other:

$$\int_{-\infty}^{\infty} \sin(\omega_1 t + \varphi_1) \cdot \sin(\omega_2 t + \varphi_2)\, \mathrm{d}t \text{ ``$=$'' } 0$$

$$\Longleftrightarrow \quad \omega_1 \neq \omega_2 \quad \vee \quad \varphi_1 - \varphi_2 = (2k+1)\pi/2 \quad (k \in \mathbb{Z})$$

They can be used to form an orthogonal function basis for a transform.

The term "orthogonal" is used here in the context of an (infinitely dimensional) vector space, where the "vectors" are functions of the form $f : \mathbb{R} \to \mathbb{R}$ (or $f : \mathbb{R} \to \mathbb{C}$) and the scalar product is defined as $f \cdot g = \int_{-\infty}^{\infty} f(t) \cdot g(t)\, \mathrm{d}t$.

Why are exponential functions useful?

Adding together two exponential functions with the same base z, but different scale factor and offset, results in another exponential function with the same base:

$$\begin{aligned} A_1 \cdot z^{t+\varphi_1} + A_2 \cdot z^{t+\varphi_2} &= A_1 \cdot z^t \cdot z^{\varphi_1} + A_2 \cdot z^t \cdot z^{\varphi_2} \\ &= (A_1 \cdot z^{\varphi_1} + A_2 \cdot z^{\varphi_2}) \cdot z^t = A \cdot z^t \end{aligned}$$

Likewise, if we convolve a sequence $\{x_n\}$ of values

$$\ldots, z^{-3}, z^{-2}, z^{-1}, 1, z, z^2, z^3, \ldots$$

$x_n = z^n$ with an arbitrary sequence $\{h_n\}$, we get $\{y_n\} = \{z^n\} * \{h_n\}$,

$$y_n = \sum_{k=-\infty}^{\infty} x_{n-k} \cdot h_k = \sum_{k=-\infty}^{\infty} z^{n-k} \cdot h_k = z^n \cdot \sum_{k=-\infty}^{\infty} z^{-k} \cdot h_k = z^n \cdot H(z)$$

where $H(z)$ is independent of n.

Exponential sequences are closed under convolution with arbitrary sequences.

The same applies in the continuous case.

Why are complex numbers so useful?

1) They give us all n solutions ("roots") of equations involving polynomials up to degree n (the " $\sqrt{-1} = \mathrm{j}$ " story).

2) They give us the "great unifying theory" that combines sine and exponential functions:

$$\begin{aligned}\cos(\omega t) &= \frac{1}{2}\left(\mathrm{e}^{\mathrm{j}\omega t} + \mathrm{e}^{-\mathrm{j}\omega t}\right)\\ \sin(\omega t) &= \frac{1}{2\mathrm{j}}\left(\mathrm{e}^{\mathrm{j}\omega t} - \mathrm{e}^{-\mathrm{j}\omega t}\right)\end{aligned}$$

or

$$\cos(\omega t + \varphi) = \frac{1}{2}\left(\mathrm{e}^{\mathrm{j}(\omega t+\varphi)} + \mathrm{e}^{-\mathrm{j}(\omega t+\varphi)}\right)$$

or

$$\begin{aligned}\cos(\omega n + \varphi) &= \Re(\mathrm{e}^{\mathrm{j}(\omega n+\varphi)}) = \Re[(\mathrm{e}^{\mathrm{j}\omega})^n \cdot \mathrm{e}^{\mathrm{j}\varphi}]\\ \sin(\omega n + \varphi) &= \Im(\mathrm{e}^{\mathrm{j}(\omega n+\varphi)}) = \Im[(\mathrm{e}^{\mathrm{j}\omega})^n \cdot \mathrm{e}^{\mathrm{j}\varphi}]\end{aligned}$$

Notation: $\Re(a + \mathrm{j}b) := a$ and $\Im(a + \mathrm{j}b) := b$ where $\mathrm{j}^2 = -1$ and $a, b \in \mathbb{R}$.

We can now represent sine waves as projections of a rotating complex vector. This allows us to represent sine-wave sequences as exponential sequences with basis $\mathrm{e}^{\mathrm{j}\omega}$.

A phase shift in such a sequence corresponds to a rotation of a complex vector.

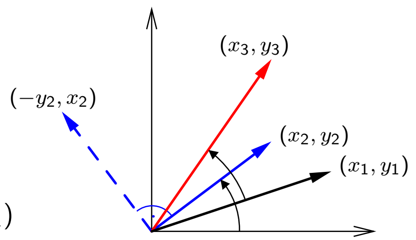
3) Complex multiplication allows us to modify the amplitude and phase of a complex rotating vector using a single operation and value.

Rotation of a 2D vector in (x, y)-form is notationally slightly messy, but fortunately $\mathrm{j}^2 = -1$ does exactly what is required here:

$$\begin{aligned}\begin{pmatrix} x_3 \\ y_3 \end{pmatrix} &= \begin{pmatrix} x_2 & -y_2 \\ y_2 & x_2 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}\\ &= \begin{pmatrix} x_1x_2 - y_1y_2 \\ x_1y_2 + x_2y_1 \end{pmatrix}\end{aligned}$$

$$z_1 = x_1 + \mathrm{j}y_1, \quad z_2 = x_2 + \mathrm{j}y_2$$

$$z_1 \cdot z_2 = x_1x_2 - y_1y_2 + \mathrm{j}(x_1y_2 + x_2y_1)$$



Complex phasors

Amplitude and phase are two distinct characteristics of a sine function that are inconvenient to keep separate notationally.

Complex functions (and discrete sequences) of the form

$$A \cdot \mathrm{e}^{\mathrm{j}(\omega t+\varphi)} = A \cdot [\cos(\omega t + \varphi) + \mathrm{j} \cdot \sin(\omega t + \varphi)]$$

(where $\mathrm{j}^2 = -1$) are able to represent both amplitude and phase in one single algebraic object.

Thanks to complex multiplication, we can also incorporate in one single factor both a multiplicative change of amplitude and an additive change of phase of such a function. This makes discrete sequences of the form

$$x_n = \mathrm{e}^{\mathrm{j}\omega n}$$

eigensequences with respect to an LTI system T, because for each ω, there is a complex number (eigenvalue) $H(\omega)$ such that

$$T\{x_n\} = H(\omega) \cdot \{x_n\}$$

In the notation of slide 32, where the argument of H is the base, we would write $H(\mathrm{e}^{\mathrm{j}\omega})$.

Recall: Fourier transform

We define the Fourier integral transform and its inverse as

$$\begin{aligned}\mathcal{F}\{g(t)\}(f) &= G(f) = \int_{-\infty}^{\infty} g(t) \cdot \mathrm{e}^{-2\pi\mathrm{j}ft}\,\mathrm{d}t\\ \mathcal{F}^{-1}\{G(f)\}(t) &= g(t) = \int_{-\infty}^{\infty} G(f) \cdot \mathrm{e}^{2\pi\mathrm{j}ft}\,\mathrm{d}f\end{aligned}$$

Many equivalent forms of the Fourier transform are used in the literature. There is no strong consensus on whether the forward transform uses $\mathrm{e}^{-2\pi\mathrm{j}ft}$ and the backwards transform $\mathrm{e}^{2\pi\mathrm{j}ft}$, or vice versa. The above form uses the *ordinary frequency* f, whereas some authors prefer the *angular frequency* $\omega = 2\pi f$:

$$\begin{aligned}\mathcal{F}\{h(t)\}(\omega) &= H(\omega) = \alpha \int_{-\infty}^{\infty} h(t) \cdot \mathrm{e}^{\mp\mathrm{j}\omega t}\,\mathrm{d}t\\ \mathcal{F}^{-1}\{H(\omega)\}(t) &= h(t) = \beta \int_{-\infty}^{\infty} H(\omega) \cdot \mathrm{e}^{\pm\mathrm{j}\omega t}\,\mathrm{d}\omega\end{aligned}$$

This substitution introduces factors α and β such that $\alpha\beta = 1/(2\pi)$. Some authors set $\alpha = 1$ and $\beta = 1/(2\pi)$, to keep the convolution theorem free of a constant prefactor; others prefer the unitary form $\alpha = \beta = 1/\sqrt{2\pi}$, in the interest of symmetry.

Properties of the Fourier transform

If

$$x(t) \quad \bullet\!\!-\!\!\circ \quad X(f) \qquad \text{and} \qquad y(t) \quad \bullet\!\!-\!\!\circ \quad Y(f)$$

are pairs of functions that are mapped onto each other by the Fourier transform, then so are the following pairs.

Linearity:

$$ax(t) + by(t) \quad \bullet\!\!-\!\!\circ \quad aX(f) + bY(f)$$

Time scaling:

$$x(at) \quad \bullet\!\!-\!\!\circ \quad \frac{1}{|a|} X\left(\frac{f}{a}\right)$$

Frequency scaling:

$$\frac{1}{|a|} x\left(\frac{t}{a}\right) \quad \bullet\!\!-\!\!\circ \quad X(af)$$

Time shifting:

$$x(t - \Delta t) \quad \bullet\!\!-\!\!\circ \quad X(f) \cdot \mathrm{e}^{-2\pi \mathrm{j} f \Delta t}$$

Frequency shifting:

$$x(t) \cdot \mathrm{e}^{2\pi \mathrm{j} \Delta f t} \quad \bullet\!\!-\!\!\circ \quad X(f - \Delta f)$$

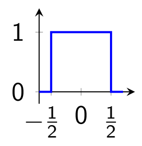
Parseval's theorem (total energy):

$$\int_{-\infty}^{\infty} |x(t)|^2 \mathrm{d}t \quad = \quad \int_{-\infty}^{\infty} |X(f)|^2 \mathrm{d}f$$

Fourier transform example: rect and sinc

The Fourier transform of the "rectangular function"

$$\mathrm{rect}(t) = \begin{cases} 1 & \text{if } |t| < \frac{1}{2} \\ \frac{1}{2} & \text{if } |t| = \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}$$



is the "(normalized) sinc function"

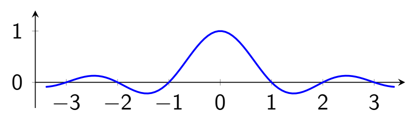
$$\mathcal{F}\{\mathrm{rect}(t)\}(f) = \int_{-\frac{1}{2}}^{\frac{1}{2}} \mathrm{e}^{-2\pi \mathrm{j} f t} \mathrm{d}t = \frac{\sin \pi f}{\pi f} = \mathrm{sinc}(f)$$

and vice versa

$$\mathcal{F}\{\mathrm{sinc}(t)\}(f) = \mathrm{rect}(f).$$

Some noteworthy properties of these functions:

- $\int_{-\infty}^{\infty} \mathrm{sinc}(t)\,\mathrm{d}t = 1 = \int_{-\infty}^{\infty} \mathrm{rect}(t)\,\mathrm{d}t$
- $\mathrm{sinc}(0) = 1 = \mathrm{rect}(0)$
- $\forall n \in \mathbb{Z} \setminus \{0\} : \mathrm{sinc}(n) = 0$



Convolution theorem

Continuous form:

$$\mathcal{F}\{(f * g)(t)\} \quad = \quad \mathcal{F}\{f(t)\} \cdot \mathcal{F}\{g(t)\}$$

$$\mathcal{F}\{f(t) \cdot g(t)\} \quad = \quad \mathcal{F}\{f(t)\} * \mathcal{F}\{g(t)\}$$

Discrete form:

$$\{x_n\} * \{y_n\} \;=\; \{z_n\} \quad \Longleftrightarrow \quad X(\mathrm{e}^{\mathrm{j}\omega}) \cdot Y(\mathrm{e}^{\mathrm{j}\omega}) \;=\; Z(\mathrm{e}^{\mathrm{j}\omega})$$

Convolution in the time domain is equivalent to (complex) scalar multiplication in the frequency domain.

Convolution in the frequency domain corresponds to scalar multiplication in the time domain.

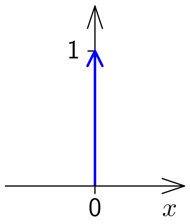
Proof: $z(r) = \int_s x(s) y(r-s) \mathrm{d}s \quad \Longleftrightarrow \quad \int_r z(r) \mathrm{e}^{-\mathrm{j}\omega r} \mathrm{d}r = \int_r \int_s x(s) y(r-s) \mathrm{e}^{-\mathrm{j}\omega r} \mathrm{d}s \mathrm{d}r =$ $\int_s x(s) \int_r y(r-s) \mathrm{e}^{-\mathrm{j}\omega r} \mathrm{d}r \mathrm{d}s = \int_s x(s) \mathrm{e}^{-\mathrm{j}\omega s} \int_r y(r-s) \mathrm{e}^{-\mathrm{j}\omega(r-s)} \mathrm{d}r \mathrm{d}s \overset{t:=r-s}{=}$ $\int_s x(s) \mathrm{e}^{-\mathrm{j}\omega s} \int_t y(t) \mathrm{e}^{-\mathrm{j}\omega t} \mathrm{d}t \mathrm{d}s = \int_s x(s) \mathrm{e}^{-\mathrm{j}\omega s} \mathrm{d}s \cdot \int_t y(t) \mathrm{e}^{-\mathrm{j}\omega t} \mathrm{d}t$. (Same for $\sum$ instead of $\int$.)

Dirac delta function

The continuous equivalent of the impulse sequence $\{\delta_n\}$ is known as Dirac delta function $\delta(x)$. It is a generalized function, defined such that

$$\delta(x) = \begin{cases} 0, & x \neq 0 \\ \infty, & x = 0 \end{cases}$$

$$\int_{-\infty}^{\infty} \delta(x)\,\mathrm{d}x = 1$$



and can be thought of as the limit of function sequences such as

$$\delta(x) = \lim_{n\to\infty} \begin{cases} 0, & |x| \geq 1/n \\ n/2, & |x| < 1/n \end{cases}$$

or

$$\delta(x) = \lim_{n\to\infty} \frac{n}{\sqrt{\pi}}\,\mathrm{e}^{-n^2x^2}$$

The delta function is mathematically speaking not a function, but a *distribution*, that is an expression that is only defined when integrated.

Some properties of the Dirac delta function:

$$\int_{-\infty}^{\infty} f(x)\delta(x-a)\,\mathrm{d}x = f(a)$$

$$\int_{-\infty}^{\infty} \mathrm{e}^{\pm 2\pi \mathrm{j} x a}\mathrm{d}x = \delta(a)$$

$$\sum_{n=-\infty}^{\infty} \mathrm{e}^{\pm 2\pi \mathrm{j} n x a} = \frac{1}{|a|}\sum_{n=-\infty}^{\infty} \delta(x - n/a)$$

$$\delta(ax) = \frac{1}{|a|}\delta(x)$$

Fourier transform:

$$\mathcal{F}\{\delta(t)\}(f) = \int_{-\infty}^{\infty} \delta(t)\cdot \mathrm{e}^{-2\pi \mathrm{j} f t}\,\mathrm{d}t = \mathrm{e}^0 = 1$$

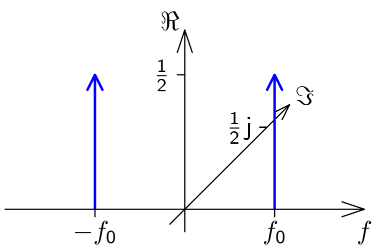
$$\mathcal{F}^{-1}\{1\}(t) = \int_{-\infty}^{\infty} 1 \cdot \mathrm{e}^{2\pi \mathrm{j} f t}\,\mathrm{d}f = \delta(t)$$

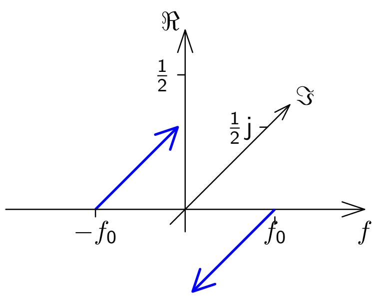
Sine and cosine in the frequency domain

$$\cos(2\pi f_0 t) = \frac{1}{2}\,\mathrm{e}^{2\pi \mathrm{j} f_0 t} + \frac{1}{2}\,\mathrm{e}^{-2\pi \mathrm{j} f_0 t} \qquad \sin(2\pi f_0 t) = \frac{1}{2\mathrm{j}}\,\mathrm{e}^{2\pi \mathrm{j} f_0 t} - \frac{1}{2\mathrm{j}}\,\mathrm{e}^{-2\pi \mathrm{j} f_0 t}$$

$$\mathcal{F}\{\cos(2\pi f_0 t)\}(f) = \frac{1}{2}\delta(f - f_0) + \frac{1}{2}\delta(f + f_0)$$

$$\mathcal{F}\{\sin(2\pi f_0 t)\}(f) = -\frac{\mathrm{j}}{2}\delta(f - f_0) + \frac{\mathrm{j}}{2}\delta(f + f_0)$$





As any $x(t) \in \mathbb{R}$ can be decomposed into sine and cosine functions, the spectrum of any real-valued signal will show the symmetry $X(\mathrm{e}^{\mathrm{j}\omega}) = [X(\mathrm{e}^{-\mathrm{j}\omega})]^*$, where * denotes the complex conjugate (i.e., negated imaginary part).

Fourier transform symmetries

We call a function $x(t)$

$$\text{odd if } x(-t) = -x(t)$$
$$\text{even if } x(-t) = x(t)$$

and $\cdot^*$ is the complex conjugate, such that $(a + \mathrm{j}b)^* = (a - \mathrm{j}b)$.

Then

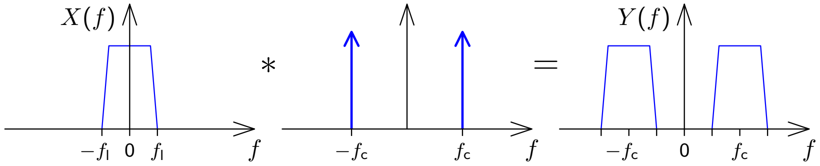
$x(t)$ is real	$\Leftrightarrow$	$X(-f) = [X(f)]^*$
$x(t)$ is imaginary	$\Leftrightarrow$	$X(-f) = -[X(f)]^*$
$x(t)$ is even	$\Leftrightarrow$	$X(f)$ is even
$x(t)$ is odd	$\Leftrightarrow$	$X(f)$ is odd
$x(t)$ is real and even	$\Leftrightarrow$	$X(f)$ is real and even
$x(t)$ is real and odd	$\Leftrightarrow$	$X(f)$ is imaginary and odd
$x(t)$ is imaginary and even	$\Leftrightarrow$	$X(f)$ is imaginary and even
$x(t)$ is imaginary and odd	$\Leftrightarrow$	$X(f)$ is real and odd

Example: amplitude modulation

Communication channels usually permit only the use of a given frequency interval, such as 300–3400 Hz for the analog phone network or 590–598 MHz for TV channel 36. Modulation with a carrier frequency f_c shifts the spectrum of a signal $x(t)$ into the desired band.

Amplitude modulation (AM):

$$y(t) = A \cdot \cos(2\pi t f_\mathrm{c}) \cdot x(t)$$



The spectrum of the baseband signal in the interval $-f_\mathrm{l} < f < f_\mathrm{l}$ is shifted by the modulation to the intervals $\pm f_\mathrm{c} - f_\mathrm{l} < f < \pm f_\mathrm{c} + f_\mathrm{l}$.

How can such a signal be demodulated?

Sampling using a Dirac comb

The loss of information in the sampling process that converts a continuous function $x(t)$ into a discrete sequence $\{x_n\}$ defined by

$$x_n = x(t_\mathrm{s} \cdot n) = x(n/f_\mathrm{s})$$

can be modelled through multiplying $x(t)$ by a comb of Dirac impulses

$$s(t) = t_\mathrm{s} \cdot \sum_{n=-\infty}^{\infty} \delta(t - t_\mathrm{s} \cdot n)$$

to obtain the sampled function

$$\hat{x}(t) = x(t) \cdot s(t)$$

The function $\hat{x}(t)$ now contains exactly the same information as the discrete sequence $\{x_n\}$, but is still in a form that can be analysed using the Fourier transform on continuous functions.

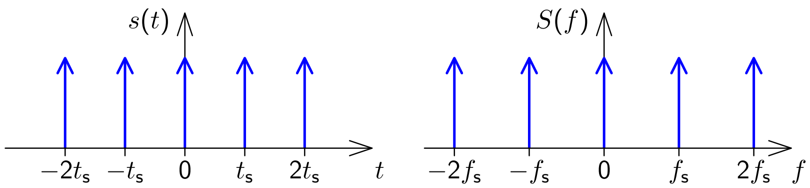
The Fourier transform of a Dirac comb

$$s(t) = t_\mathrm{s} \cdot \sum_{n=-\infty}^{\infty} \delta(t - t_\mathrm{s} \cdot n) \quad = \sum_{n=-\infty}^{\infty} \mathrm{e}^{2\pi \mathrm{j} n t / t_\mathrm{s}}$$

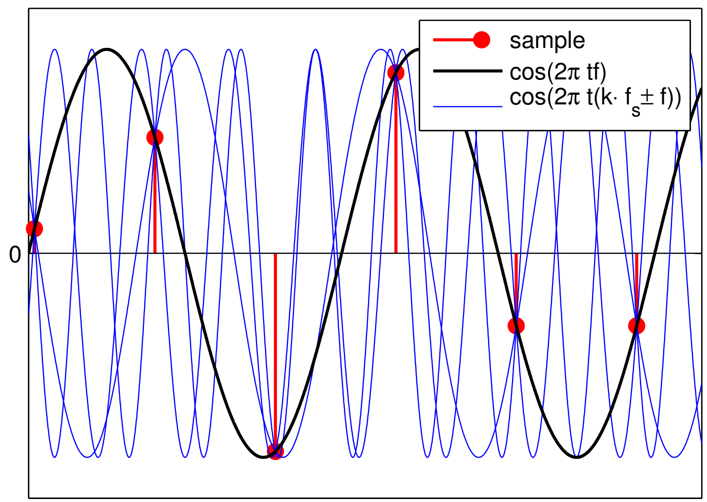
is another Dirac comb

$$S(f) = \mathcal{F}\left\{ t_\mathrm{s} \cdot \sum_{n=-\infty}^{\infty} \delta(t - t_\mathrm{s} n) \right\}(f) =$$

$$t_\mathrm{s} \cdot \int_{-\infty}^{\infty} \sum_{n=-\infty}^{\infty} \delta(t - t_\mathrm{s} n)\, \mathrm{e}^{-2\pi \mathrm{j} f t} \mathrm{d}t = \sum_{n=-\infty}^{\infty} \delta\left(f - \frac{n}{t_\mathrm{s}}\right).$$

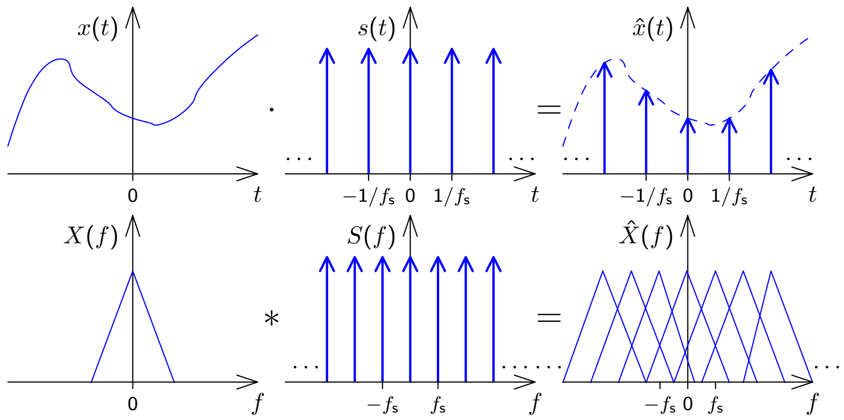


Sampling and aliasing



Sampled at frequency f_s, the function $\cos(2\pi t f)$ cannot be distinguished from $\cos[2\pi t(k f_\mathrm{s} \pm f)]$ for any $k \in \mathbb{Z}$.

Frequency-domain view of sampling



Sampling a signal in the time domain corresponds in the frequency domain to convolving its spectrum with a Dirac comb. The resulting copies of the original signal spectrum in the spectrum of the sampled signal are called "images".

Discrete-time Fourier transform

The Fourier transform of a sampled signal

$$\hat{x}(t) = t_s \cdot \sum_{n=-\infty}^{\infty} x_n \cdot \delta(t - t_s \cdot n)$$

is

$$\mathcal{F}\{\hat{x}(t)\}(f) = \hat{X}(f) = \int_{-\infty}^{\infty} \hat{x}(t) \cdot \mathrm{e}^{-2\pi \mathrm{j} f t} \mathrm{d}t = t_s \cdot \sum_{n=-\infty}^{\infty} x_n \cdot \mathrm{e}^{-2\pi \mathrm{j} \frac{f}{f_s} n}$$

Some authors prefer the notation $\hat{X}(\mathrm{e}^{\mathrm{j}\omega}) = \sum_n x_n \cdot \mathrm{e}^{-\mathrm{j}\omega n}$ to highlight the periodicity of $\hat{X}$ and its relationship with the z-transform (slide 99), where $\omega = 2\pi \frac{f}{f_s}$.

The inverse transform is

$$\hat{x}(t) = \int_{-\infty}^{\infty} \hat{X}(f) \cdot \mathrm{e}^{2\pi \mathrm{j} f t} \mathrm{d}f \quad \text{or} \quad x_m = \int_{-f_s/2}^{f_s/2} \hat{X}(f) \cdot \mathrm{e}^{2\pi \mathrm{j} \frac{f}{f_s} m} \mathrm{d}f.$$

Nyquist limit and anti-aliasing filters

If the (double-sided) bandwidth of a signal to be sampled is larger than the sampling frequency f_s, the images of the signal that emerge during sampling may overlap with the original spectrum.

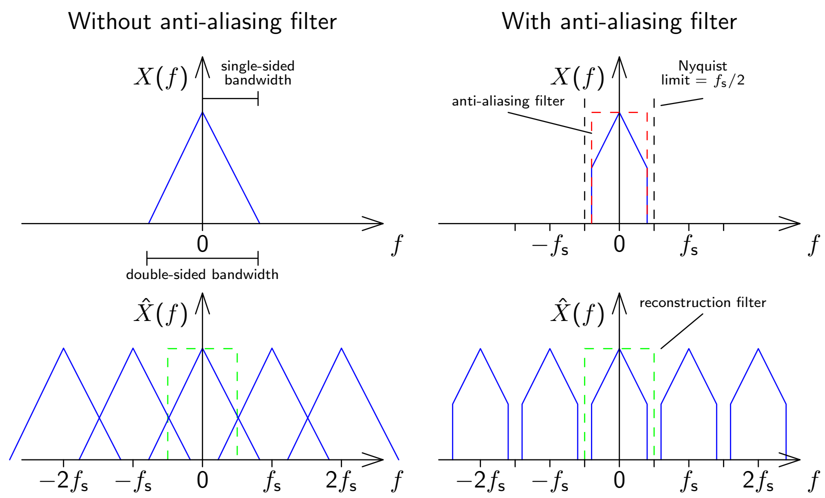
Such an overlap will hinder reconstruction of the original continuous signal by removing the aliasing frequencies with a *reconstruction filter*.

Therefore, it is advisable to limit the bandwidth of the input signal to the sampling frequency f_s before sampling, using an *anti-aliasing filter*.

In the common case of a real-valued base-band signal (with frequency content down to 0 Hz), all frequencies f that occur in the signal with non-zero power should be limited to the interval $-f_s/2 < f < f_s/2$.

The upper limit $f_s/2$ for the single-sided bandwidth of a baseband signal is known as the "Nyquist limit".

Nyquist limit and anti-aliasing filters



Anti-aliasing and reconstruction filters both suppress frequencies outside $|f| < f_s/2$.

Reconstruction of a continuous band-limited waveform

The ideal anti-aliasing filter for eliminating any frequency content above $f_s/2$ before sampling with a frequency of f_s has the Fourier transform

$$H(f) = \begin{cases} 1 & \text{if } |f| < \frac{f_s}{2} \\ 0 & \text{if } |f| > \frac{f_s}{2} \end{cases} = \mathrm{rect}(t_s f).$$

This leads, after an inverse Fourier transform, to the impulse response

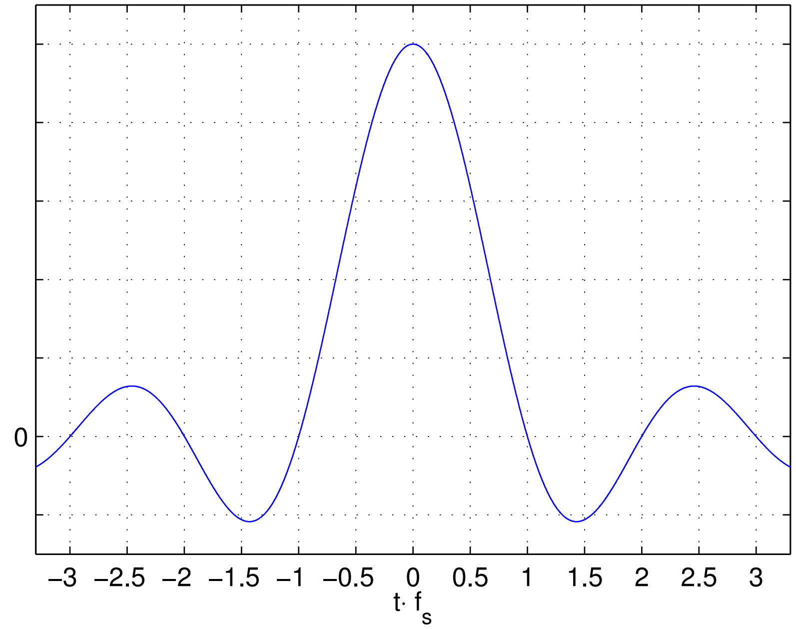
$$h(t) = f_s \cdot \frac{\sin \pi t f_s}{\pi t f_s} = \frac{1}{t_s} \cdot \mathrm{sinc}\left(\frac{t}{t_s}\right).$$

The original band-limited signal can be reconstructed by convolving this with the sampled signal $\hat{x}(t)$, which eliminates the periodicity of the frequency domain introduced by the sampling process:

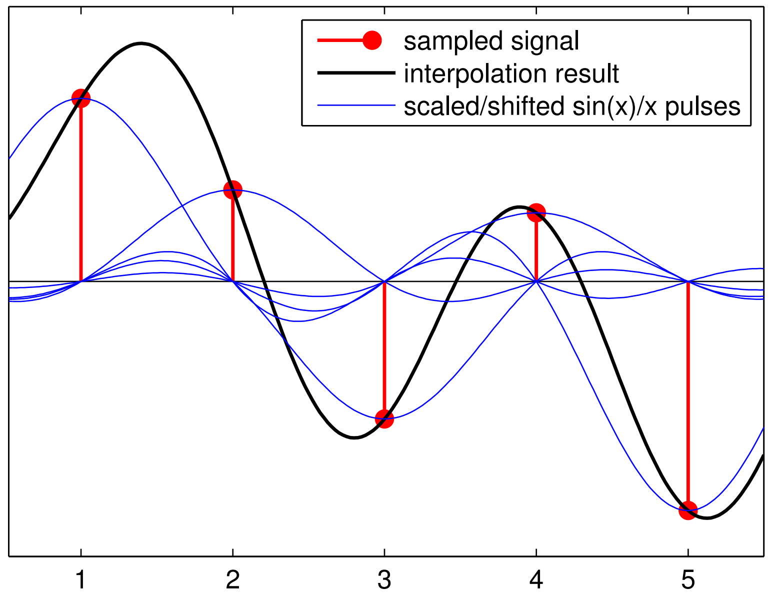
$$x(t) = h(t) * \hat{x}(t)$$

Note that sampling $h(t)$ gives the impulse function: $h(t) \cdot s(t) = \delta(t)$.

Impulse response of ideal low-pass filter with cut-off frequency $f_s/2$:



Reconstruction filter example



Reconstruction filters

The mathematically ideal form of a reconstruction filter for suppressing aliasing frequencies interpolates the sampled signal $x_n = x(t_s \cdot n)$ back into the continuous waveform

$$x(t) = \sum_{n=-\infty}^{\infty} x_n \cdot \frac{\sin \pi(t - t_s \cdot n)}{\pi(t - t_s \cdot n)}.$$

Choice of sampling frequency

Due to causality and economic constraints, practical analog filters can only approximate such an ideal low-pass filter. Instead of a sharp transition between the "pass band" ($< f_s/2$) and the "stop band" ($> f_s/2$), they feature a "transition band" in which their signal attenuation gradually increases.

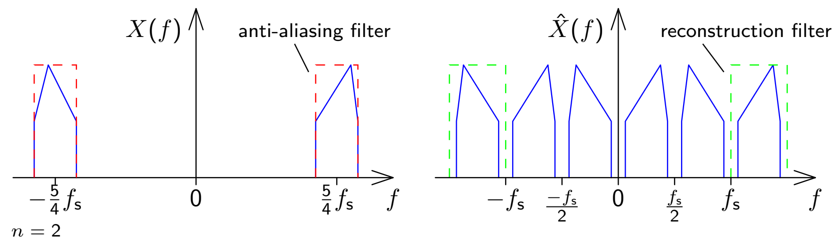
The sampling frequency is therefore usually chosen somewhat higher than twice the highest frequency of interest in the continuous signal (e.g., 4×). On the other hand, the higher the sampling frequency, the higher are CPU, power and memory requirements. Therefore, the choice of sampling frequency is a tradeoff between signal quality, analog filter cost and digital subsystem expenses.

Band-pass signal sampling

Sampled signals can also be reconstructed if their spectral components remain entirely within the interval $n \cdot f_s/2 < |f| < (n+1) \cdot f_s/2$ for some $n \in \mathbb{N}$. (The baseband case discussed so far is just $n = 0$.)

In this case, the aliasing copies of the positive and the negative frequencies will interleave instead of overlap, and can therefore be removed again with a reconstruction filter with the impulse response

$$h(t) = f_s \frac{\sin \pi t f_s/2}{\pi t f_s/2} \cdot \cos\left(2\pi t f_s \frac{2n+1}{4}\right) = (n+1) f_s \frac{\sin \pi t (n+1) f_s}{\pi t (n+1) f_s} - n f_s \frac{\sin \pi t n f_s}{\pi t n f_s}.$$



IQ sampling / complex baseband signal

Consider signal $x(t) \in \mathbb{R}$ in which only frequencies $f_l < |f| < f_h$ are of interest. This band has a centre frequency of $f_c = (f_l + f_h)/2$ and a bandwidth $B = f_h - f_l$. It can be sampled efficiently (at the lowest possible sampling frequency) by *downconversion*:

- Shift its spectrum by $-f_c$:

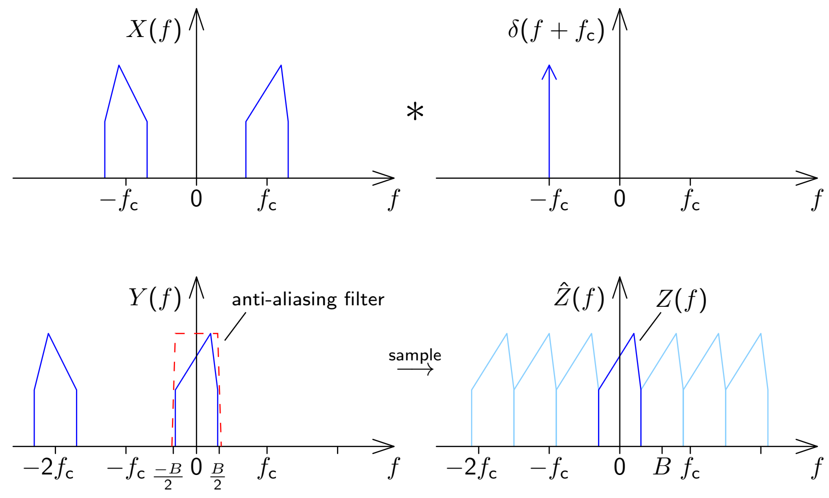
$$y(t) = x(t) \cdot e^{-2\pi j f_c t}$$

- Low-pass filter it with a cut-off frequency of $B/2$:

$$z(t) = B \int_{-\infty}^{\infty} y(\tau) \cdot \text{sinc}((t-\tau)B) \cdot d\tau \;\bullet\!\!-\!\!\circ\; Z(f) = Y(f) \cdot \text{rect}(f/B)$$

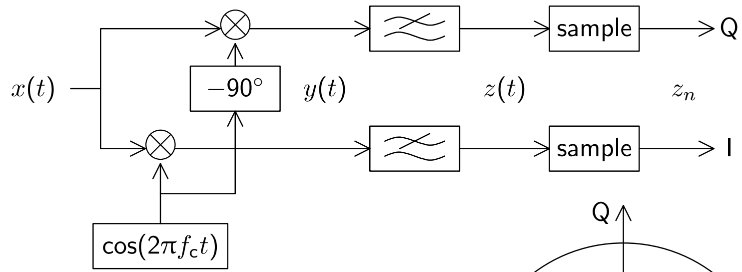
- Sample the result at sampling frequency B (or higher):

$$z_n = z(n/B)$$



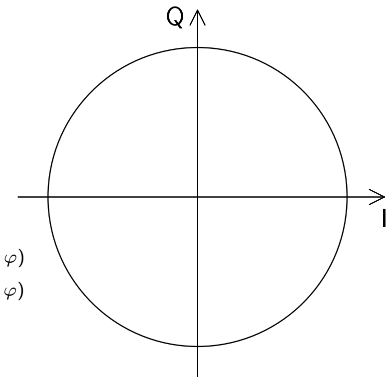
Shifting the center frequency f_c of the interval of interest to 0 Hz (DC) makes the spectrum asymmetric. This leads to a complex-valued time-domain representation
$(\exists f : Z(f) \neq [Z(-f)]^* \Longrightarrow \exists t : z(t) \in \mathbb{C} \setminus \mathbb{R})$.

The real part $\Re(z(t))$ is also known as "in-phase" signal (I) and the imaginary part $\Im(z(t))$ as "quadrature" signal (Q).



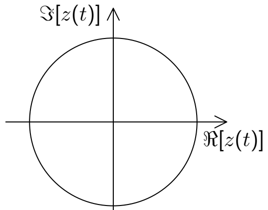
Consider:

- $\sin(x) = \cos(x - \frac{1}{2}\pi)$
- $\cos(x) \cdot \cos(x) = \frac{1}{2} + \frac{1}{2}\cos 2x$
- $\sin(x) \cdot \sin(x) = \frac{1}{2} - \frac{1}{2}\cos 2x$
- $\sin(x) \cdot \cos(x) = 0 + \frac{1}{2}\sin 2x$
- $\cos(x) \cdot \cos(x - \varphi) = \frac{1}{2}\cos(\varphi) + \frac{1}{2}\cos(2x - \varphi)$
- $\sin(x) \cdot \cos(x - \varphi) = \frac{1}{2}\sin(\varphi) + \frac{1}{2}\sin(2x - \varphi)$



Recall products of sine and cosine:

- $\cos(x) \cdot \cos(y) = \frac{1}{2}\cos(x-y) + \frac{1}{2}\cos(x+y)$
- $\sin(x) \cdot \sin(y) = \frac{1}{2}\cos(x-y) - \frac{1}{2}\cos(x+y)$
- $\sin(x) \cdot \cos(y) = \frac{1}{2}\sin(x-y) + \frac{1}{2}\sin(x+y)$



Examples:

Amplitude-modulated signal:

$$x(t) = s(t) \cdot \cos(2\pi t f_c + \varphi) \quad \longrightarrow \quad z(t) = \frac{1}{2} \cdot s(t) \cdot e^{j\varphi}$$

Noncoherent demodulation: $s(t) = 2|z(t)|$ $(s(t) > 0$ required)

Coherent demodulation: $s(t) = 2\Re[z(t) \cdot e^{-j\varphi}]$ (φ required)

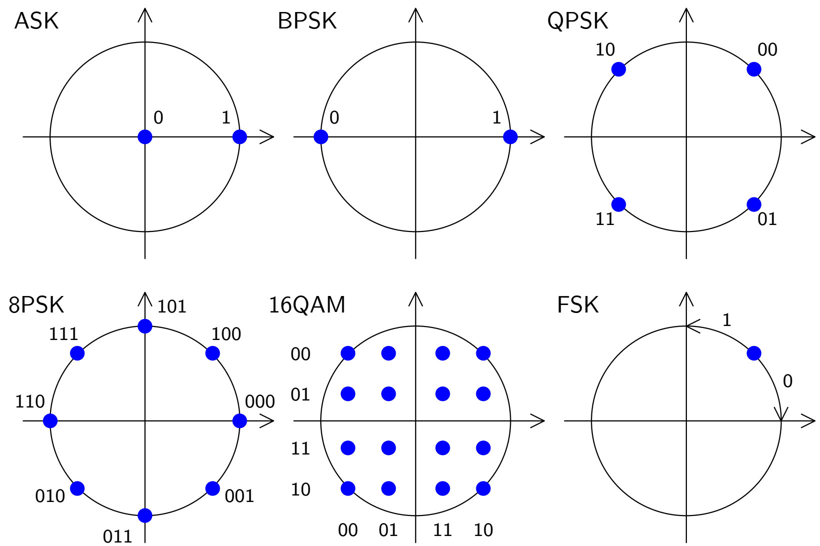
Frequency-modulated signal:

$$x(t) = \cos\left[2\pi t f_c + \int_0^t s(\tau)\mathrm{d}\tau + \varphi\right] \quad \longrightarrow \quad z(t) = \frac{1}{2} \cdot e^{j\int_0^t s(\tau)\mathrm{d}\tau + j\varphi}$$

Demodulation: idea is $s(t) = \frac{\mathrm{d}}{\mathrm{d}t}\angle z(t)$, where $a \cdot \angle e^{j\phi} = \phi$ $(a, \phi \in \mathbb{R})$, but only for $-\pi \leq \phi < \pi$.

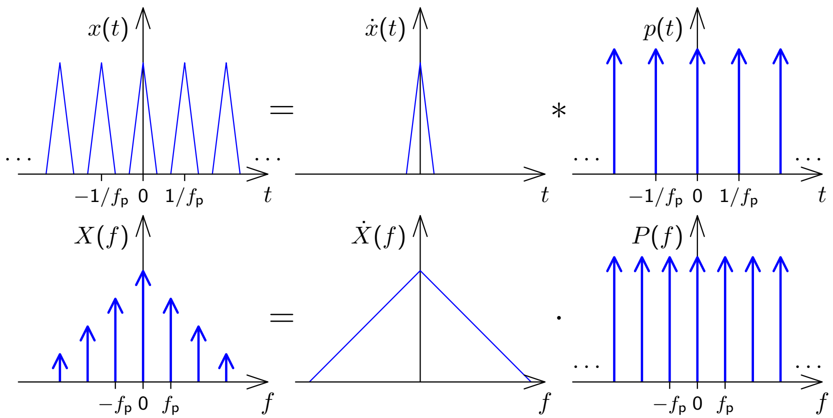
In practice: $s(t) \approx \Im\left[\frac{\mathrm{d}z(t)}{\mathrm{d}t} \cdot z^*(t)\right] / |z(t)|^2$ or $s(t) \approx \angle\frac{z(t)}{z(t-\Delta t)} / \Delta t$

Digital modulation schemes



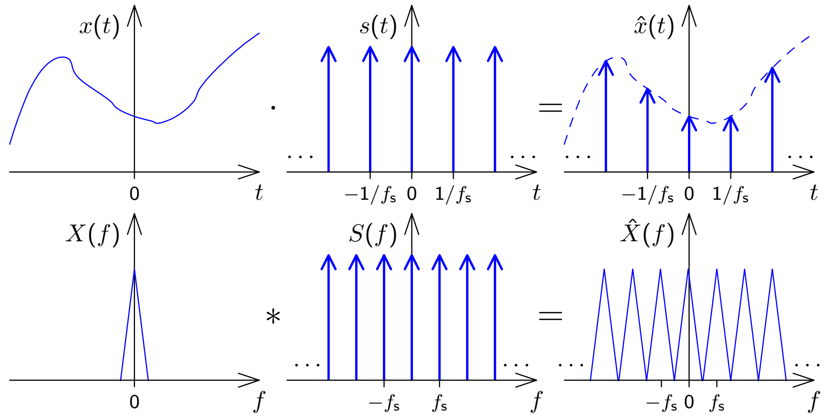
Spectrum of a periodic signal

A signal $x(t)$ that is periodic with frequency f_p can be factored into a single period $\dot{x}(t)$ convolved with an impulse comb $p(t)$. This corresponds in the frequency domain to the multiplication of the spectrum of the single period with a comb of impulses spaced f_p apart.



Spectrum of a sampled signal

A signal $x(t)$ that is sampled with frequency f_s has a spectrum that is periodic with a period of f_s.

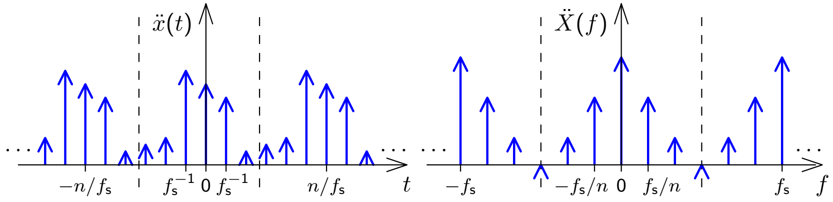


Continuous vs discrete Fourier transform

- Sampling a **continuous** signal makes its spectrum **periodic**
- A **periodic** signal has a **sampled** spectrum

We **sample** a signal $x(t)$ with f_s, getting $\hat{x}(t)$. We take n consecutive samples of $\hat{x}(t)$ and **repeat** these periodically, getting a new signal $\ddot{x}(t)$ with period n/f_s. Its spectrum $\ddot{X}(f)$ is **sampled** (i.e., has non-zero value) at frequency intervals f_s/n and **repeats** itself with a period f_s.

Now both $\ddot{x}(t)$ and its spectrum $\ddot{X}(f)$ are **finite** vectors of length n.



Discrete Fourier Transform (DFT)

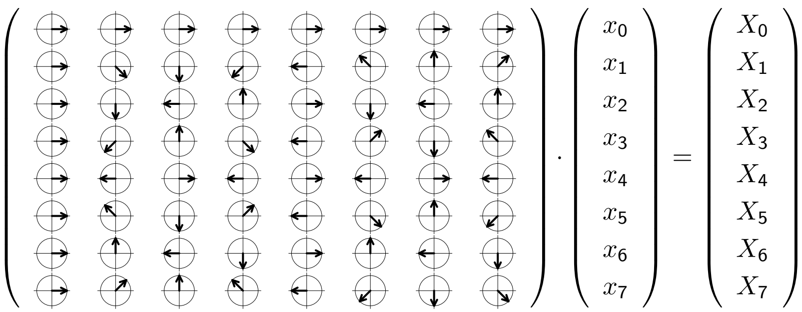
$$X_k = \sum_{i=0}^{n-1} x_i \cdot \mathrm{e}^{-2\pi \mathrm{j}\frac{ik}{n}} \qquad x_k = \frac{1}{n} \cdot \sum_{i=0}^{n-1} X_i \cdot \mathrm{e}^{2\pi \mathrm{j}\frac{ik}{n}}$$

The n-point DFT multiplies a vector with an $n \times n$ matrix

$$F_n = \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \mathrm{e}^{-2\pi\mathrm{j}\frac{1}{n}} & \mathrm{e}^{-2\pi\mathrm{j}\frac{2}{n}} & \mathrm{e}^{-2\pi\mathrm{j}\frac{3}{n}} & \cdots & \mathrm{e}^{-2\pi\mathrm{j}\frac{n-1}{n}} \\ 1 & \mathrm{e}^{-2\pi\mathrm{j}\frac{2}{n}} & \mathrm{e}^{-2\pi\mathrm{j}\frac{4}{n}} & \mathrm{e}^{-2\pi\mathrm{j}\frac{6}{n}} & \cdots & \mathrm{e}^{-2\pi\mathrm{j}\frac{2(n-1)}{n}} \\ 1 & \mathrm{e}^{-2\pi\mathrm{j}\frac{3}{n}} & \mathrm{e}^{-2\pi\mathrm{j}\frac{6}{n}} & \mathrm{e}^{-2\pi\mathrm{j}\frac{9}{n}} & \cdots & \mathrm{e}^{-2\pi\mathrm{j}\frac{3(n-1)}{n}} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \mathrm{e}^{-2\pi\mathrm{j}\frac{n-1}{n}} & \mathrm{e}^{-2\pi\mathrm{j}\frac{2(n-1)}{n}} & \mathrm{e}^{-2\pi\mathrm{j}\frac{3(n-1)}{n}} & \cdots & \mathrm{e}^{-2\pi\mathrm{j}\frac{(n-1)(n-1)}{n}} \end{pmatrix}$$

$$F_n \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{pmatrix} = \begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ \vdots \\ X_{n-1} \end{pmatrix}, \qquad \frac{1}{n} \cdot F_n^* \cdot \begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ \vdots \\ X_{n-1} \end{pmatrix} = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{pmatrix}$$

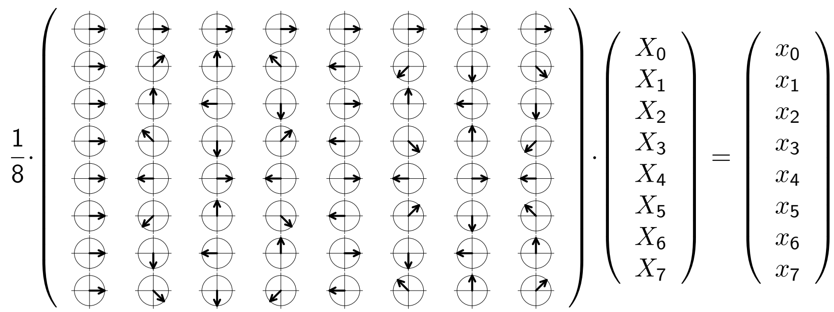
Discrete Fourier Transform visualized



The n-point DFT of a signal $\{x_i\}$ sampled at frequency f_s contains in the elements X_0 to $X_{n/2}$ of the resulting frequency-domain vector the frequency components 0, f_s/n, $2f_\mathsf{s}/n$, $3f_\mathsf{s}/n$, $\dots$, $f_\mathsf{s}/2$, and contains in X_{n-1} downto $X_{n/2}$ the corresponding negative frequencies. Note that for a real-valued input vector, both X_0 and $X_{n/2}$ will be real, too.

Why is there no phase information recovered at $f_\mathsf{s}/2$?

Inverse DFT visualized



Fast Fourier Transform (FFT)

$$\left(\mathcal{F}_n\{x_i\}_{i=0}^{n-1}\right)_k = \sum_{i=0}^{n-1} x_i \cdot \mathrm{e}^{-2\pi \mathrm{j}\frac{ik}{n}}$$

$$= \sum_{i=0}^{\frac{n}{2}-1} x_{2i} \cdot \mathrm{e}^{-2\pi \mathrm{j}\frac{ik}{n/2}} + \mathrm{e}^{-2\pi \mathrm{j}\frac{k}{n}} \sum_{i=0}^{\frac{n}{2}-1} x_{2i+1} \cdot \mathrm{e}^{-2\pi \mathrm{j}\frac{ik}{n/2}}$$

$$= \begin{cases} \left(\mathcal{F}_{\frac{n}{2}}\{x_{2i}\}_{i=0}^{\frac{n}{2}-1}\right)_k + \mathrm{e}^{-2\pi \mathrm{j}\frac{k}{n}} \cdot \left(\mathcal{F}_{\frac{n}{2}}\{x_{2i+1}\}_{i=0}^{\frac{n}{2}-1}\right)_k, & k < \frac{n}{2} \\ \left(\mathcal{F}_{\frac{n}{2}}\{x_{2i}\}_{i=0}^{\frac{n}{2}-1}\right)_{k-\frac{n}{2}} + \mathrm{e}^{-2\pi \mathrm{j}\frac{k}{n}} \cdot \left(\mathcal{F}_{\frac{n}{2}}\{x_{2i+1}\}_{i=0}^{\frac{n}{2}-1}\right)_{k-\frac{n}{2}}, & k \geq \frac{n}{2} \end{cases}$$

The DFT over n-element vectors can be reduced to two DFTs over $n/2$-element vectors plus n multiplications and n additions, leading to $\log_2 n$ rounds and $n \log_2 n$ additions and multiplications overall, compared to n^2 for the equivalent matrix multiplication.

A high-performance FFT implementation in C with many processor-specific optimizations and support for non-power-of-2 sizes is available at `http://www.fftw.org/`.

Efficient real-valued FFT

The symmetry properties of the Fourier transform applied to the discrete Fourier transform $\{X_i\}_{i=0}^{n-1} = \mathcal{F}_n\{x_i\}_{i=0}^{n-1}$ have the form

$$\begin{aligned} \forall i: x_i &= \Re(x_i) &\iff& \forall i: X_{n-i} = X_i^* \\ \forall i: x_i &= \mathrm{j} \cdot \Im(x_i) &\iff& \forall i: X_{n-i} = -X_i^* \end{aligned}$$

These two symmetries, combined with the linearity of the DFT, allows us to calculate two real-valued n-point DFTs

$$\{X_i'\}_{i=0}^{n-1} = \mathcal{F}_n\{x_i'\}_{i=0}^{n-1} \qquad \{X_i''\}_{i=0}^{n-1} = \mathcal{F}_n\{x_i''\}_{i=0}^{n-1}$$

simultaneously in a single complex-valued n-point DFT, by composing its input as

$$x_i = x_i' + \mathrm{j} \cdot x_i''$$

and decomposing its output as

$$X_i' = \frac{1}{2}(X_i + X_{n-i}^*) \qquad X_i'' = \frac{1}{2\mathrm{j}}(X_i - X_{n-i}^*)$$

where $X_n = X_0$.

To optimize the calculation of a single real-valued FFT, use this trick to calculate the two half-size real-value FFTs that occur in the first round.

Fast complex multiplication

Calculating the product of two complex numbers as

$$(a + \mathrm{j}b) \cdot (c + \mathrm{j}d) = (ac - bd) + \mathrm{j}(ad + bc)$$

involves four (real-valued) multiplications and two additions.

The alternative calculation

$$(a + \mathrm{j}b) \cdot (c + \mathrm{j}d) = (\alpha - \beta) + \mathrm{j}(\alpha + \gamma) \quad \text{with} \quad \begin{aligned} \alpha &= a(c+d) \\ \beta &= d(a+b) \\ \gamma &= c(b-a) \end{aligned}$$

provides the same result with three multiplications and five additions.

The latter may perform faster on CPUs where multiplications take three or more times longer than additions.

This "Karatsuba multiplication" is most helpful on simpler microcontrollers. Specialized signal-processing CPUs (DSPs) feature 1-clock-cycle multipliers. High-end desktop processors use pipelined multipliers that stall where operations depend on each other.

FFT-based convolution

Calculating the convolution of two finite sequences $\{x_i\}_{i=0}^{m-1}$ and $\{y_i\}_{i=0}^{n-1}$ of lengths m and n via

$$z_i = \sum_{j=\max\{0, i-(n-1)\}}^{\min\{m-1, i\}} x_j \cdot y_{i-j}, \qquad 0 \leq i < m + n - 1$$

takes mn multiplications.

Can we apply the FFT and the convolution theorem to calculate the convolution faster, in just $O(m \log m + n \log n)$ multiplications?

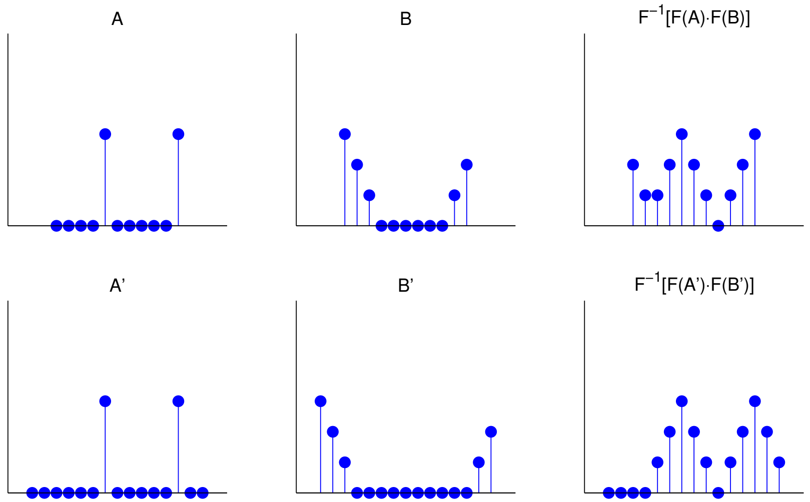
$$\{z_i\} = \mathcal{F}^{-1}\left(\mathcal{F}\{x_i\} \cdot \mathcal{F}\{y_i\}\right)$$

There is obviously no problem if this condition is fulfilled:

$\{x_i\}$ and $\{y_i\}$ are periodic, with equal period lengths

In this case, the fact that the DFT interprets its input as a single period of a periodic signal will do exactly what is needed, and the FFT and inverse FFT can be applied directly as above.

In the general case, measures have to be taken to prevent a wrap-over:



Both sequences are padded with zero values to a length of at least $m + n - 1$. This ensures that the start and end of the resulting sequence do not overlap.

Zero padding is usually applied to extend both sequence lengths to the next higher power of two ($2^{\lceil \log_2(m+n-1) \rceil}$), which facilitates the FFT.

With a causal sequence, simply append the padding zeros at the end.

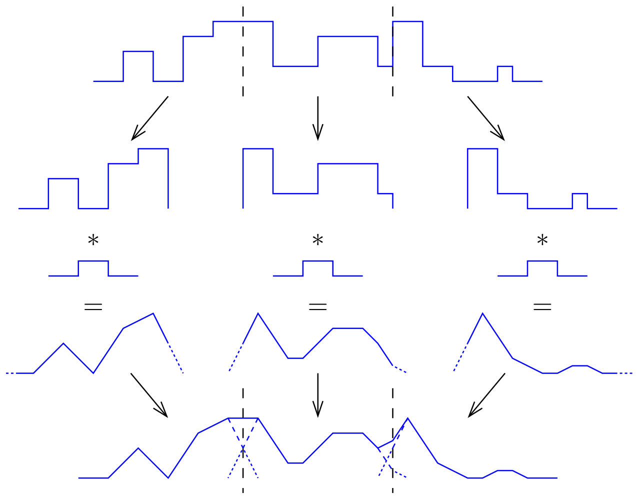
With a non-causal sequence, values with a negative index number are wrapped around the DFT block boundaries and appear at the right end. In this case, zero-padding is applied in the center of the block, between the last and first element of the sequence.

Thanks to the periodic nature of the DFT, zero padding at both ends has the same effect as padding only at one end.

If both sequences can be loaded entirely into RAM, the FFT can be applied to them in one step. However, one of the sequences might be too large for that. It could also be a realtime waveform (e.g., a telephone signal) that cannot be delayed until the end of the transmission.

In such cases, the sequence has to be split into shorter blocks that are separately convolved and then added together with a suitable overlap.

Each block is zero-padded at both ends and then convolved as before:

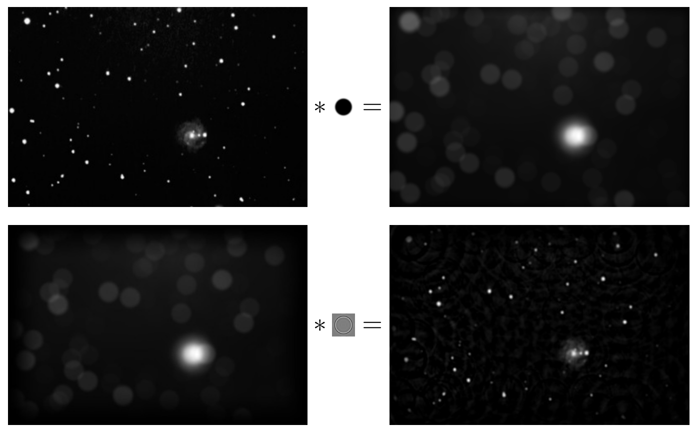


The regions originally added as zero padding are, after convolution, aligned to overlap with the unpadded ends of their respective neighbour blocks. The overlapping parts of the blocks are then added together.

Deconvolution

A signal $u(t)$ was distorted by convolution with a known impulse response $h(t)$ (e.g., through a transmission channel or a sensor problem). The "smeared" result $s(t)$ was recorded.

Can we undo the damage and restore (or at least estimate) $u(t)$?



The convolution theorem turns the problem into one of multiplication:

$$\begin{aligned} s(t) &= \int u(t-\tau)\cdot h(\tau)\cdot \mathsf{d}\tau \\ s &= u * h \\ \mathcal{F}\{s\} &= \mathcal{F}\{u\}\cdot\mathcal{F}\{h\} \\ \mathcal{F}\{u\} &= \mathcal{F}\{s\}/\mathcal{F}\{h\} \\ u &= \mathcal{F}^{-1}\{\mathcal{F}\{s\}/\mathcal{F}\{h\}\} \end{aligned}$$

In practice, we also record some noise $n(t)$ (quantization, etc.):

$$c(t) = s(t) + n(t) = \int u(t-\tau)\cdot h(\tau)\cdot \mathsf{d}\tau + n(t)$$

Problem – At frequencies f where $\mathcal{F}\{h\}(f)$ approaches zero, the noise will be amplified (potentially enormously) during deconvolution:

$$\tilde{u} = \mathcal{F}^{-1}\{\mathcal{F}\{c\}/\mathcal{F}\{h\}\} = u + \mathcal{F}^{-1}\{\mathcal{F}\{n\}/\mathcal{F}\{h\}\}$$

Typical workarounds:

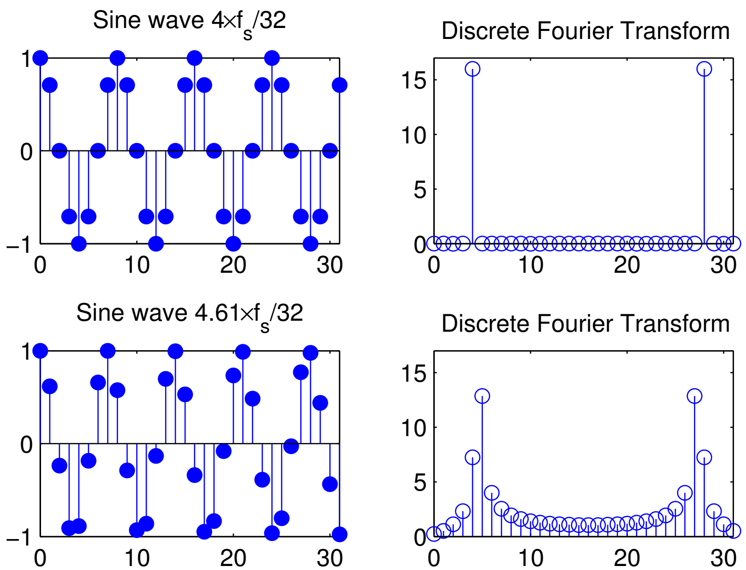
- Modify the Fourier transform of the impulse response, such that $|\mathcal{F}\{h\}(f)| > \epsilon$ for some experimentally chosen threshold ϵ.
- If estimates of the signal spectrum $|\mathcal{F}\{s\}(f)|$ and the noise spectrum $|\mathcal{F}\{n\}(f)|$ can be obtained, then we can apply the "Wiener filter" ("optimal filter")

$$W(f) = \frac{|\mathcal{F}\{s\}(f)|^2}{|\mathcal{F}\{s\}(f)|^2 + |\mathcal{F}\{n\}(f)|^2}$$

before deconvolution:

$$\tilde{u} = \mathcal{F}^{-1}\{W\cdot\mathcal{F}\{c\}/\mathcal{F}\{h\}\}$$

Spectral estimation



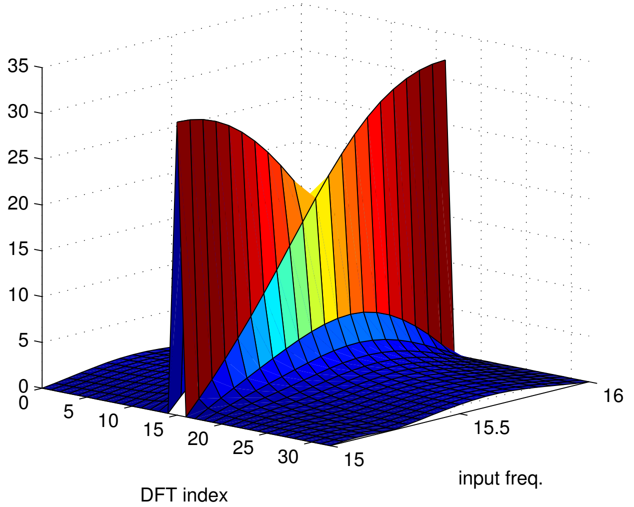
We introduced the DFT as a special case of the continuous Fourier transform, where the input is sampled *and periodic*.

If the input is sampled, but not periodic, the DFT can still be used to calculate an approximation of the Fourier transform of the original continuous signal. However, there are two effects to consider. They are particularly visible when analysing pure sine waves.

Sine waves whose frequency is a multiple of the base frequency (f_s/n) of the DFT are identical to their periodic extension beyond the size of the DFT. They are, therefore, represented exactly by a single sharp peak in the DFT. All their energy falls into one single frequency "bin" in the DFT result.

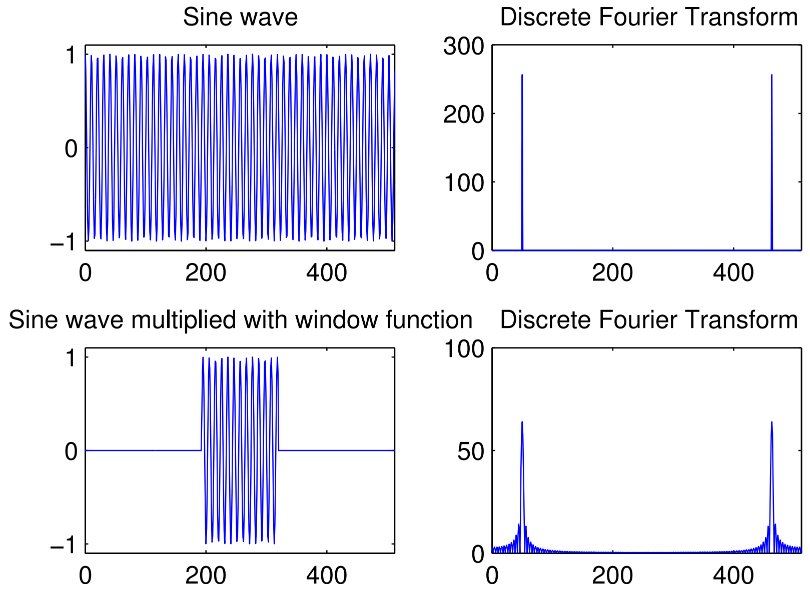
Sine waves with other frequencies, which do not match exactly one of the output frequency bins of the DFT, are still represented by a peak at the output bin that represents the nearest integer multiple of the DFT's base frequency. However, such a peak is distorted in two ways:

- Its amplitude is lower (down to 63.7%).
- Much signal energy has "leaked" to other frequencies.



The *leakage* of energy to other frequency bins not only blurs the estimated spectrum. The peak amplitude also changes significantly as the frequency of a tone changes from that associated with one output bin to the next, a phenomenon known as *scalloping*. In the above graphic, an input sine wave gradually changes from the frequency of bin 15 to that of bin 16 (only positive frequencies shown).

Windowing



The reason for the leakage and scalloping losses is easy to visualize with the help of the convolution theorem:

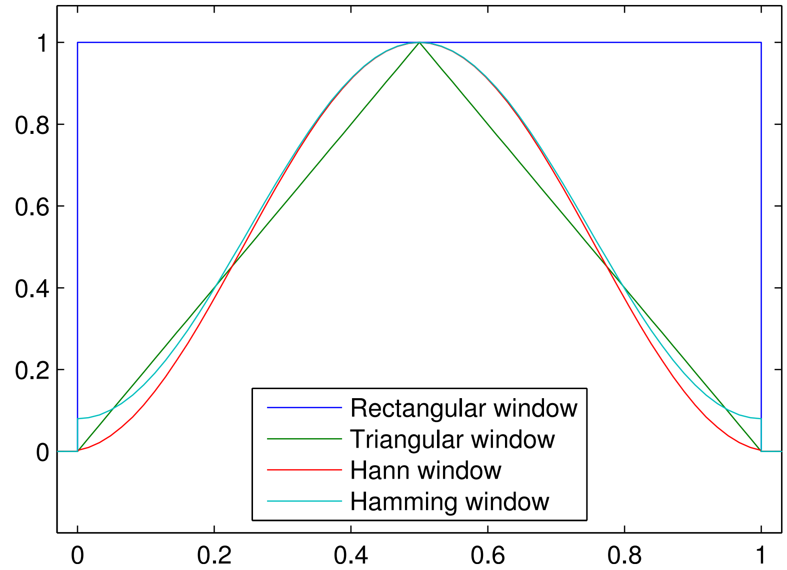
The operation of cutting a sequence of the size of the DFT input vector out of a longer original signal (the one whose continuous Fourier spectrum we try to estimate) is equivalent to multiplying this signal with a rectangular function. This destroys all information and continuity outside the "window" that is fed into the DFT.

Multiplication with a rectangular window of length T in the time domain is equivalent to convolution with $\sin(\pi f T)/(\pi f T)$ in the frequency domain.

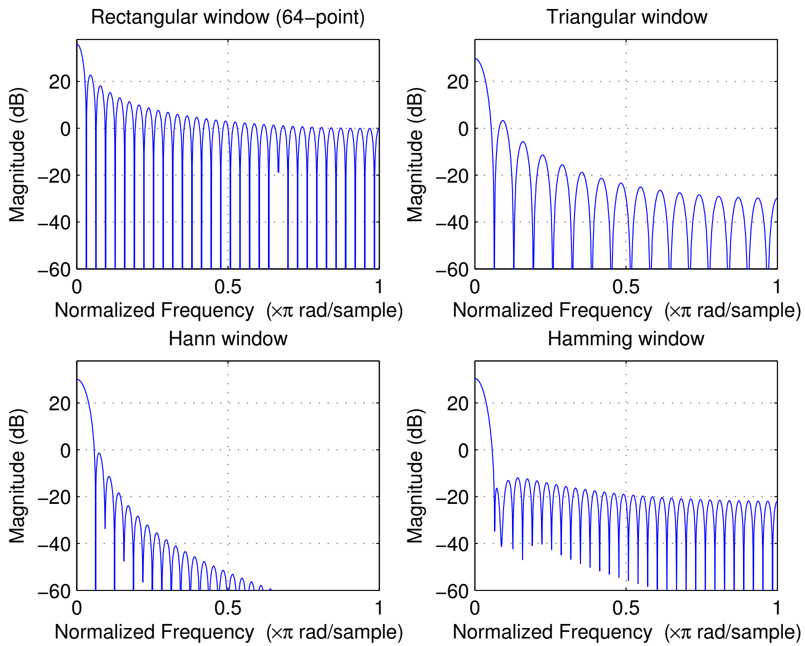
The subsequent interpretation of this window as a periodic sequence by the DFT leads to sampling of this convolution result (sampling meaning multiplication with a Dirac comb whose impulses are spaced f_s/n apart).

Where the window length was an exact multiple of the original signal period, sampling of the $\sin(\pi f T)/(\pi f T)$ curve leads to a single Dirac pulse, and the windowing causes no distortion. In all other cases, the effects of the convolution become visible in the frequency domain as leakage and scalloping losses.

Some better window functions



All these functions are 0 outside the interval [0,1].



Numerous alternatives to the rectangular window have been proposed that reduce leakage and scalloping in spectral estimation. These are vectors multiplied element-wise with the input vector before applying the DFT to it. They all force the signal amplitude smoothly down to zero at the edge of the window, thereby avoiding the introduction of sharp jumps in the signal when it is extended periodically by the DFT.

Three examples of such window vectors $\{w_i\}_{i=0}^{n-1}$ are:

Triangular window (Bartlett window):

$$w_i = 1 - \left|1 - \frac{i}{n/2}\right|$$

Hann window (raised-cosine window, Hanning window):

$$w_i = 0.5 - 0.5 \times \cos\left(2\pi \frac{i}{n-1}\right)$$

Hamming window:

$$w_i = 0.54 - 0.46 \times \cos\left(2\pi \frac{i}{n-1}\right)$$

Zero padding increases DFT resolution

The two figures below show two spectra of the 16-element sequence

$$s_i = \cos(2\pi \cdot 3i/16) + \cos(2\pi \cdot 4i/16), \qquad i \in \{0, \dots, 15\}.$$

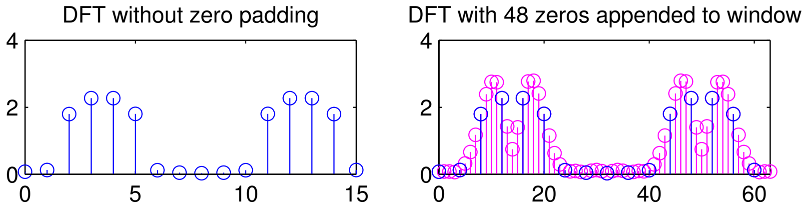
The left plot shows the DFT of the windowed sequence

$$x_i = s_i \cdot w_i, \qquad i \in \{0, \dots, 15\}$$

and the right plot shows the DFT of the zero-padded windowed sequence

$$x'_i = \begin{cases} s_i \cdot w_i, & i \in \{0, \dots, 15\} \\ 0, & i \in \{16, \dots, 63\} \end{cases}$$

where $w_i = 0.54 - 0.46 \times \cos(2\pi i/15)$ is the Hamming window.



Applying the discrete Fourier transform to an n-element long real-valued sequence leads to a spectrum consisting of only $n/2 + 1$ discrete frequencies.

Since the resulting spectrum has already been distorted by multiplying the (hypothetically longer) signal with a windowing function that limits its length to n non-zero values and forces the waveform smoothly down to zero at the window boundaries, appending further zeros outside the window will not distort the signal further.

The frequency resolution of the DFT is the sampling frequency divided by the block size of the DFT. Zero padding can therefore be used to increase the frequency resolution of the DFT.

Note that zero padding does *not* add any additional information to the signal. The spectrum has already been "low-pass filtered" by being convolved with the spectrum of the windowing function. Zero padding in the time domain merely samples this spectrum blurred by the windowing step at a higher resolution, thereby making it easier to visually distinguish spectral lines and to locate their peak more precisely.

Digital filters

Filter: supresses (removes, attenuates) unwanted signal components.

- **low-pass filter** – suppress all frequencies above a cut-off frequency
- **high-pass filter** – suppress all frequencies below a cut-off frequency, including DC (direct current = 0 Hz)
- **band-pass filter** – suppress signals outside a frequency interval (= passband)
- **band-stop filter** (aka: band-reject filter) – suppress signals inside a single frequency interval (= stopband)
- **notch filter** – narrow band-stop filter, ideally suppressing only a single frequency

For digital filters, we also distinguish

- **finite impulse response (FIR)** filters
- **infinite impulse response (IIR)** filters

depending on how far their memory reaches back in time.

Window-based design of FIR filters

Recall that the ideal continuous low-pass filter with cut-off frequency f_c has the frequency characteristic

$$H(f) = \begin{cases} 1 & \text{if } |f| < f_c \\ 0 & \text{if } |f| > f_c \end{cases} = \text{rect}\left(\frac{f}{2f_c}\right)$$

and the impulse response

$$h(t) = 2f_c \frac{\sin 2\pi t f_c}{2\pi t f_c} = 2f_c \cdot \text{sinc}(2f_c \cdot t).$$

Sampling this impulse response with the sampling frequency f_s of the signal to be processed will lead to a periodic frequency characteristic, that matches the periodic spectrum of the sampled signal.

There are two problems though:

- the impulse response is infinitely long
- this filter is not causal, that is $h(t) \neq 0$ for $t < 0$

Solutions:

- Make the impulse response finite by multiplying the sampled $h(t)$ with a windowing function
- Make the impulse response causal by adding a delay of half the window size

The impulse response of an n-th order low-pass filter is then chosen as

$$h_i = 2f_c/f_s \cdot \frac{\sin[2\pi(i - n/2)f_c/f_s]}{2\pi(i - n/2)f_c/f_s} \cdot w_i$$

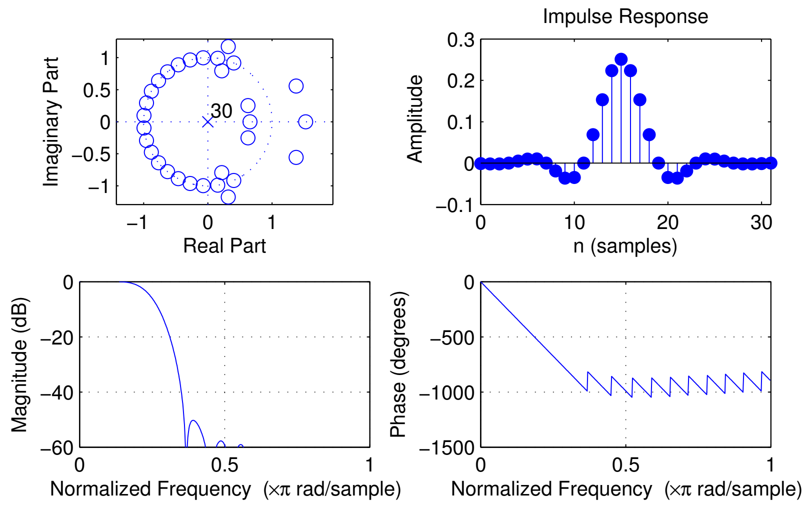
where $\{w_i\}$ is a windowing sequence, such as the Hamming window

$$w_i = 0.54 - 0.46 \times \cos(2\pi i/n)$$

with $w_i = 0$ for $i < 0$ and $i > n$.

Note that for $f_c = f_s/4$, we have $h_i = 0$ for all even values of i. Therefore, this special case requires only half the number of multiplications during the convolution. Such "half-band" FIR filters are used, for example, as anti-aliasing filters wherever a sampling rate needs to be halved.

FIR low-pass filter design example



order: $n = 30$, cutoff frequency (-6 dB): $f_c = 0.25 \times f_s/2$, window: Hamming

Filter performance

An ideal filter has a gain of 1 in the pass-band and a gain of 0 in the stop band, and nothing in between.

A practical filter will have

- frequency-dependent gain near 1 in the passband
- frequency-dependent gain below a threshold in the stopband
- a transition band between the pass and stop bands

We truncate the ideal, infinitely-long impulse response by multiplication with a window sequence.

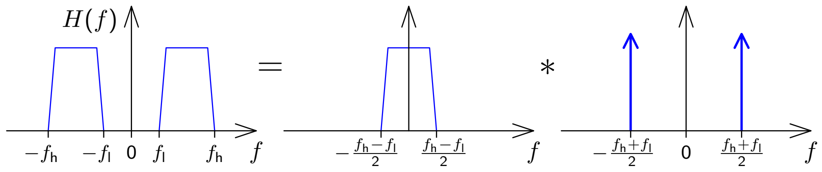
In the frequency domain, this will convolve the rectangular frequency response of the ideal low-pass filter with the frequency characteristic of the window.

The width of the main lobe determines the width of the transition band, and the side lobes cause ripples in the passband and stopband.

Low-pass to band-pass filter conversion (modulation)

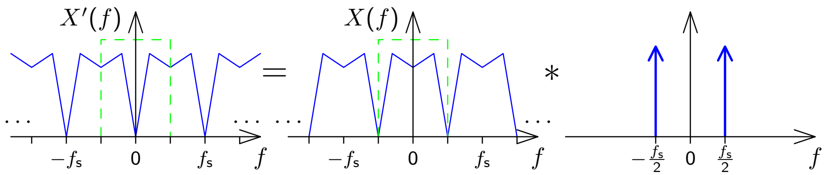
To obtain a band-pass filter that attenuates all frequencies f outside the range $f_\mathrm{l} < f < f_\mathrm{h}$, we first design a low-pass filter with a cut-off frequency $(f_\mathrm{h} - f_\mathrm{l})/2$. We then multiply its impulse response with a sine wave of frequency $(f_\mathrm{h} + f_\mathrm{l})/2$, effectively amplitude modulating it, to shift its centre frequency. Finally, we apply a window function:

$$h_i = (f_\mathrm{h} - f_\mathrm{l})/f_\mathrm{s} \cdot \frac{\sin[\pi(i - n/2)(f_\mathrm{h} - f_\mathrm{l})/f_\mathrm{s}]}{\pi(i - n/2)(f_\mathrm{h} - f_\mathrm{l})/f_\mathrm{s}} \cdot \cos[\pi i(f_\mathrm{h} + f_\mathrm{l})/f_\mathrm{s}] \cdot w_i$$



Low-pass to high-pass filter conversion (freq. inversion)

In order to turn the spectrum $X(f)$ of a real-valued signal x_i sampled at f_s into an inverted spectrum $X'(f) = X(f_\mathrm{s}/2 - f)$, we merely have to shift the periodic spectrum by $f_\mathrm{s}/2$:



This can be accomplished by multiplying the sampled sequence x_i with $y_i = \cos \pi f_\mathrm{s} t = \cos \pi i$, which is nothing but multiplication with the sequence

$$\ldots, 1, -1, 1, -1, 1, -1, 1, -1, \ldots$$

So in order to design a discrete high-pass filter that attenuates all frequencies f outside the range $f_\mathrm{c} < |f| < f_\mathrm{s}/2$, we merely have to design a low-pass filter that attenuates all frequencies outside the range $-f_\mathrm{c} < f < f_\mathrm{c}$, and then multiply every second value of its impulse response with -1.

Polynomial representation of sequences

We can represent sequences $\{x_n\}$ as polynomials:

$$X(v) = \sum_{n=-\infty}^{\infty} x_n v^n$$

Example of polynomial multiplication:

$$\begin{array}{ccccccccc}
 & (1 & + & 2v & + & 3v^2) & \cdot & (2 + 1v) \\
\hline
 & 2 & + & 4v & + & 6v^2 & & \\
+ & & & 1v & + & 2v^2 & + & 3v^3 \\
\hline
= & 2 & + & 5v & + & 8v^2 & + & 3v^3
\end{array}$$

Compare this with the convolution of two sequences (in MATLAB):

`conv([1 2 3], [2 1])` equals `[2 5 8 3]`

Convolution of sequences is equivalent to polynomial multiplication:

$$\begin{aligned}
\{h_n\} * \{x_n\} &= \{y_n\} \quad \Rightarrow \quad y_n = \sum_{k=-\infty}^{\infty} h_k \cdot x_{n-k} \\
&\quad\ \downarrow \qquad \downarrow \\
H(v) \cdot X(v) &= \left(\sum_{n=-\infty}^{\infty} h_n v^n \right) \cdot \left(\sum_{n=-\infty}^{\infty} x_n v^n \right) \\
&= \sum_{n=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} h_k \cdot x_{n-k} \cdot v^n
\end{aligned}$$

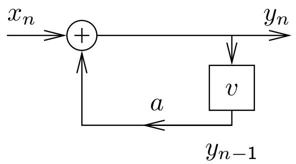
Note how the Fourier transform of a sequence can be accessed easily from its polynomial form:

$$X(\mathrm{e}^{-\mathrm{j}\omega}) = \sum_{n=-\infty}^{\infty} x_n \mathrm{e}^{-\mathrm{j}\omega n}$$

Example of polynomial division:

$$\frac{1}{1-av} = 1 + av + a^2v^2 + a^3v^3 + \cdots = \sum_{n=0}^{\infty} a^n v^n$$

$$\begin{array}{r|l}
 & 1 + av + a^2v^2 + \cdots \\ \hline
1 - av & 1 \\
 & 1 - av \\ \hline
 & \quad av \\
 & \quad av - a^2v^2 \\ \hline
 & \qquad a^2v^2 \\
 & \qquad a^2v^2 - a^3v^3 \\ \hline
 & \qquad\qquad \cdots
\end{array}$$



Rational functions (quotients of two polynomials) can provide a convenient closed-form representations for infinitely-long exponential sequences, in particular the impulse responses of IIR filters.

The z-transform

The z-transform of a sequence $\{x_n\}$ is defined as:

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

Note that this differs only in the sign of the exponent from the polynomial representation discussed on the preceeding slides.

Recall that the above $X(z)$ is exactly the factor with which an exponential sequence $\{z^n\}$ is multiplied, if it is convolved with $\{x_n\}$:

$$\{z^n\} * \{x_n\} = \{y_n\}$$

$$\Rightarrow \quad y_n = \sum_{k=-\infty}^{\infty} z^{n-k} x_k = z^n \cdot \sum_{k=-\infty}^{\infty} z^{-k} x_k = z^n \cdot X(z)$$

The z-transform defines for each sequence a continuous complex-valued surface over the complex plane $\mathbb{C}$. For finite sequences, its value is always defined across the entire complex plane.

For infinite sequences, it can be shown that the z-transform converges only for the region

$$\lim_{n\to\infty} \left| \frac{x_{n+1}}{x_n} \right| < |z| < \lim_{n\to-\infty} \left| \frac{x_{n+1}}{x_n} \right|$$

The z-transform identifies a sequence unambiguously only in conjunction with a given *region of convergence*. In other words, there exist different sequences, that have the same expression as their z-transform, but that converge for different amplitudes of z.

The z-transform is a generalization of the discrete-time Fourier transform, which it contains on the complex unit circle ($|z| = 1$):

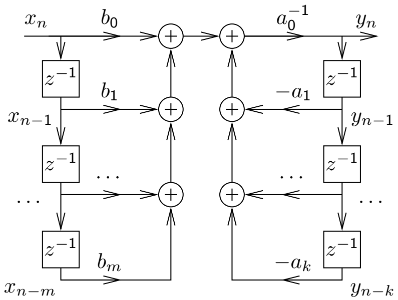
$$t_{\mathrm{s}}^{-1} \cdot \mathcal{F}\{\hat{x}(t)\}(f) = X(\mathrm{e}^{\mathrm{j}\omega}) = \sum_{n=-\infty}^{\infty} x_n \mathrm{e}^{-\mathrm{j}\omega n}$$

where $\omega = 2\pi \frac{f}{f_{\mathrm{s}}}$.

The z-transform of the impulse response $\{h_n\}$ of the causal LTI system defined by

$$\sum_{l=0}^{k} a_l \cdot y_{n-l} = \sum_{l=0}^{m} b_l \cdot x_{n-l}$$

with $\{y_n\} = \{h_n\} * \{x_n\}$ is the rational function



$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_m z^{-m}}{a_0 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_k z^{-k}}$$

($b_m \neq 0$, $a_k \neq 0$) which can also be written as

$$H(z) = \frac{z^k \sum_{l=0}^{m} b_l z^{m-l}}{z^m \sum_{l=0}^{k} a_l z^{k-l}}.$$

$H(z)$ has m zeros and k poles at non-zero locations in the z plane, plus $k-m$ zeros (if $k>m$) or $m-k$ poles (if $m>k$) at $z=0$.

This function can be converted into the form

$$H(z) = \frac{b_0}{a_0} \cdot \frac{\prod_{l=1}^{m}(1 - c_l \cdot z^{-1})}{\prod_{l=1}^{k}(1 - d_l \cdot z^{-1})} = \frac{b_0}{a_0} \cdot z^{k-m} \cdot \frac{\prod_{l=1}^{m}(z - c_l)}{\prod_{l=1}^{k}(z - d_l)}$$

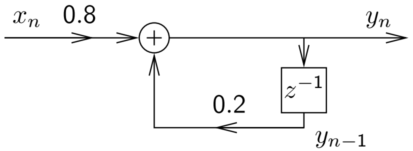
where the c_l are the non-zero positions of zeros ($H(c_l)=0$) and the d_l are the non-zero positions of the poles (i.e., $z \to d_l \Rightarrow |H(z)| \to \infty$) of $H(z)$. Except for a constant factor, $H(z)$ is entirely characterized by the position of these zeros and poles.

On the unit circle $z = \mathrm{e}^{\mathrm{j}\omega}$, $H(\mathrm{e}^{\mathrm{j}\omega})$ is the discrete-time Fourier transform of $\{h_n\}$ ($\omega = \pi f / \frac{f_s}{2}$). The DTFT amplitude can also be expressed in terms of the relative position of $\mathrm{e}^{\mathrm{j}\omega}$ to the zeros and poles:

$$|H(\mathrm{e}^{\mathrm{j}\omega})| = \left|\frac{b_0}{a_0}\right| \cdot \frac{\prod_{l=1}^{m} |\mathrm{e}^{\mathrm{j}\omega} - c_l|}{\prod_{l=1}^{k} |\mathrm{e}^{\mathrm{j}\omega} - d_l|}$$

Example: z-transform of a simple filter

Consider this IIR filter:



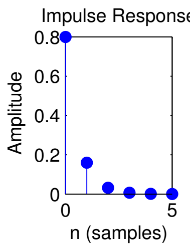
Its z-transform

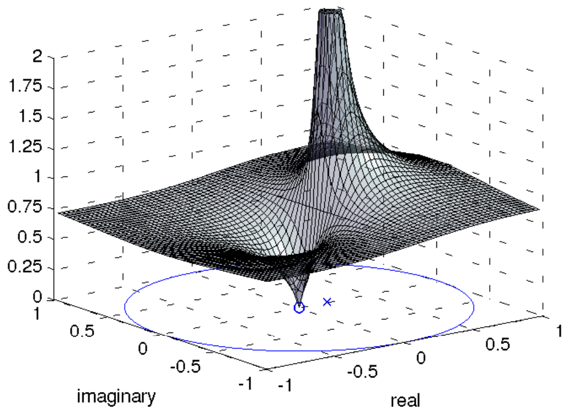
$$H(z) = \frac{0.8}{1 - 0.2 \cdot z^{-1}} = \frac{0.8z}{z - 0.2}$$

has a *zero* at $c_1 = 0$ and a *pole* at $d_1 = 0.2$. Amplitude $|H(z)|$:

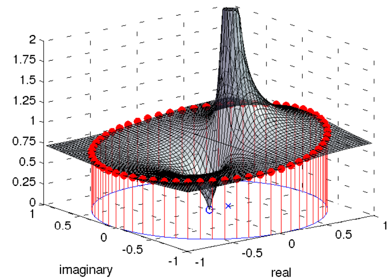
$a_0 = 1$, $a_1 = -0.2$, $b_0 = 0.8$

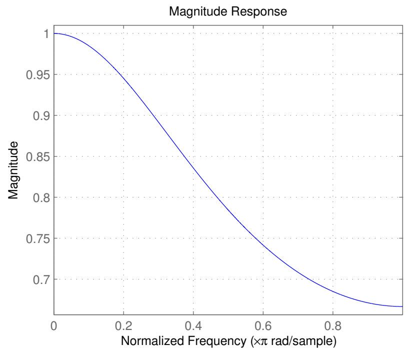
$x_n = \delta_n \Rightarrow y_n =$





$H(z) = \frac{0.8}{1 - 0.2 \cdot z^{-1}} = \frac{0.8z}{z - 0.2}$ (cont'd)





Run this LTI filter at sampling frequency f_s and test it with sinusoidial input (frequency f, amplitude 1): $x_n = \cos(2\pi f n / f_s)$

Output: $y_n = A(f) \cdot \cos(2\pi f n / f_s + \theta(f))$

What are the *gain* $A(f)$ and *phase delay* $\theta(f)$ at frequency f?

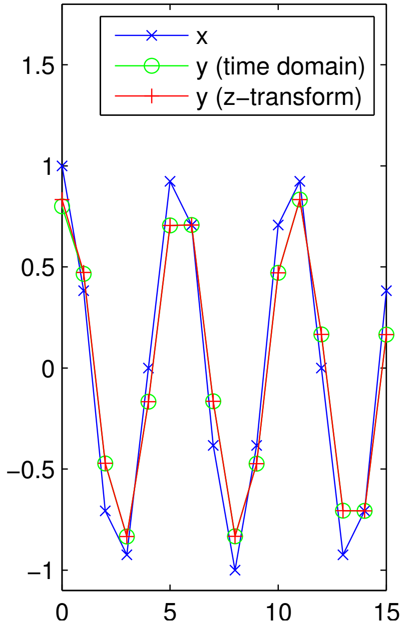
Answer: $A(f) = |H(\mathrm{e}^{\mathrm{j}2\pi f / f_s})|$

$$\theta(f) = \angle H(\mathrm{e}^{\mathrm{j}2\pi f / f_s}) = \tan^{-1} \frac{\Im\{H(\mathrm{e}^{\mathrm{j}\pi f / f_s})\}}{\Re\{H(\mathrm{e}^{\mathrm{j}\pi f / f_s})\}}$$

Example: $f_s = 8$ kHz, $f = 2$ kHz (normalized frequency $f / \frac{f_s}{2} = 0.5$) $\Rightarrow$ Gain $A(2\ \text{kHz}) = |H(\mathrm{e}^{\mathrm{j}\pi/2})| = |H(\mathrm{j})| = \left|\frac{0.8\mathrm{j}}{\mathrm{j}-0.2}\right| = \left|\frac{0.8\mathrm{j}(-\mathrm{j}-0.2)}{(\mathrm{j}-0.2)(-\mathrm{j}-0.2)}\right| = \left|\frac{0.8 - 0.16\mathrm{j}}{1+0.04}\right| = \sqrt{\frac{0.8^2 + 0.16^2}{1.04^2}} = 0.784\ldots$

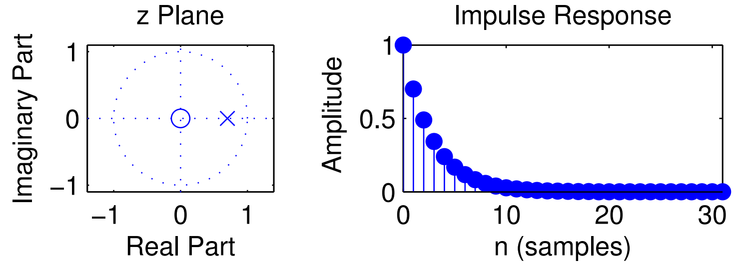
Visual verification in MATLAB:

```
n = 0:15;
fs = 8000;
f = 1500;
x = cos(2*pi*f*n/fs);
b = [0.8];  a = [1 -0.2];
y1 = filter(b,a,x);
z = exp(j*2*pi*f/fs);
H = 0.8*z/(z-0.2);
A = abs(H);
theta = atan(imag(H)/real(H));
y2 = A*cos(2*pi*f*n/fs+theta);
plot(n, x, 'bx-', ...
     n, y1, 'go-', ...
     n, y2, 'r+-')
legend('x', ...
       'y (time domain)', ...
       'y (z-transform)')
ylim([-1.1 1.8])
```

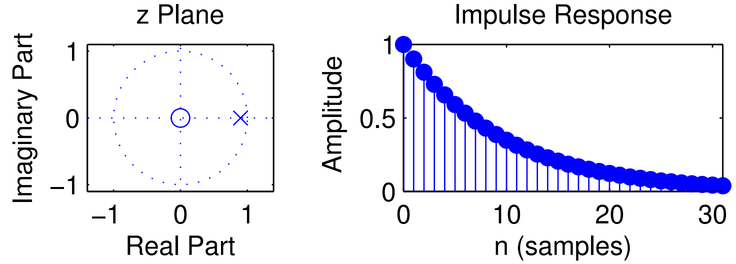


$H(z) = \frac{z}{z-0.7} = \frac{1}{1-0.7\cdot z^{-1}}$

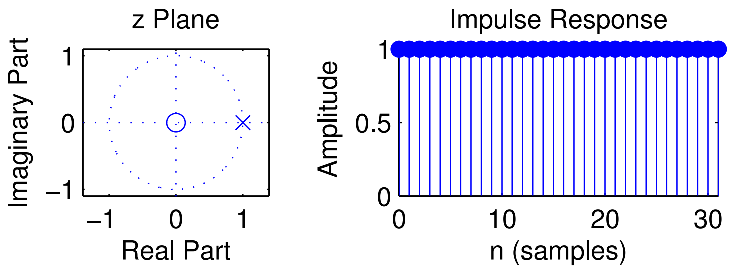
How do poles affect time domain?



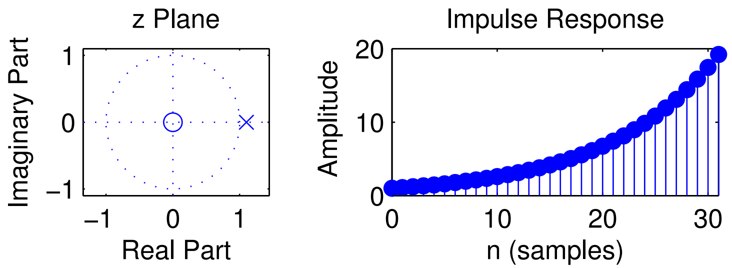
$H(z) = \frac{z}{z-0.9} = \frac{1}{1-0.9\cdot z^{-1}}$



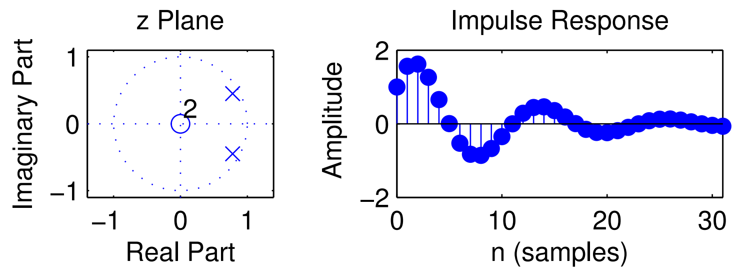
$H(z) = \frac{z}{z-1} = \frac{1}{1-z^{-1}}$



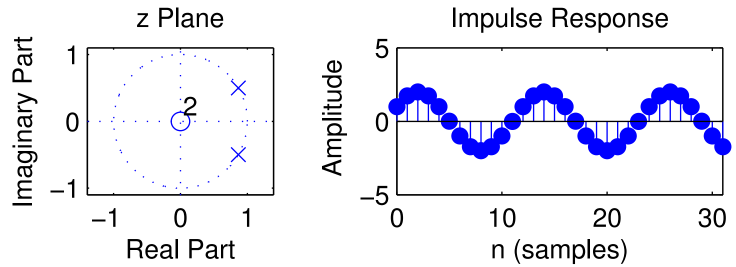
$H(z) = \frac{z}{z-1.1} = \frac{1}{1-1.1\cdot z^{-1}}$



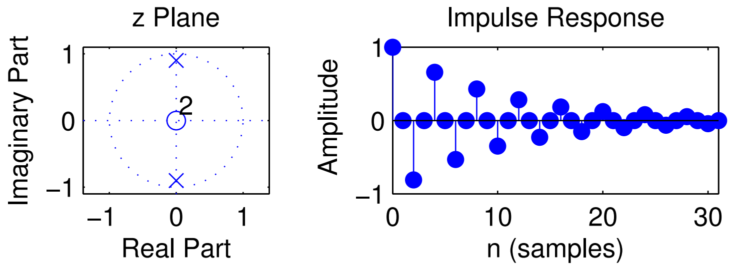
$H(z) = \frac{z^2}{(z-0.9\cdot \mathrm{e}^{\mathrm{j}\pi/6})\cdot(z-0.9\cdot \mathrm{e}^{-\mathrm{j}\pi/6})} = \frac{1}{1-1.8\cos(\pi/6)z^{-1}+0.9^2\cdot z^{-2}}$



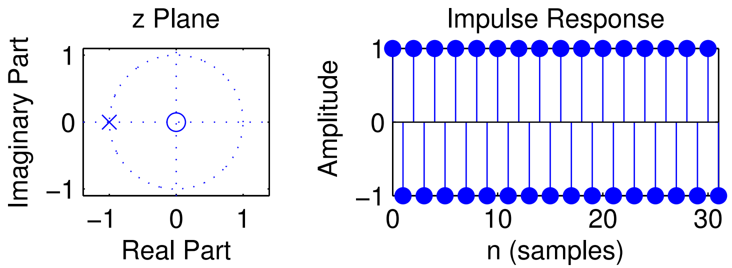
$H(z) = \frac{z^2}{(z-\mathrm{e}^{\mathrm{j}\pi/6})\cdot(z-\mathrm{e}^{-\mathrm{j}\pi/6})} = \frac{1}{1-2\cos(\pi/6)z^{-1}+z^{-2}}$



$$H(z) = \frac{z^2}{(z-0.9\cdot\mathrm{e}^{\mathrm{j}\pi/2})\cdot(z-0.9\cdot\mathrm{e}^{-\mathrm{j}\pi/2})} = \frac{1}{1-1.8\cos(\pi/2)z^{-1}+0.9^2\cdot z^{-2}} = \frac{1}{1+0.9^2\cdot z^{-2}}$$



$$H(z) = \frac{z}{z+1} = \frac{1}{1+z^{-1}}$$



Properties of the z-transform

As with the Fourier transform, convolution in the time domain corresponds to complex multiplication in the z-domain:

$$\{x_n\} \bullet\!\!-\!\!\circ\; X(z),\;\; \{y_n\} \bullet\!\!-\!\!\circ\; Y(z) \Rightarrow \{x_n\} * \{y_n\} \bullet\!\!-\!\!\circ\; X(z)\cdot Y(z)$$

Delaying a sequence by one corresponds in the z-domain to multiplication with z^{-1}:

$$\{x_{n-\Delta n}\} \bullet\!\!-\!\!\circ\; X(z)\cdot z^{-\Delta n}$$

IIR filter design goals

The design of a filter starts with specifying the desired parameters:

- The *passband* is the frequency range where we want to approximate a gain of one.
- The *stopband* is the frequency range where we want to approximate a gain of zero.
- The *order* of a filter is the number of poles it uses in the z-domain, and equivalently the number of delay elements necessary to implement it.
- Both passband and stopband will in practice not have gains of exactly one and zero, respectively, but may show several deviations from these ideal values, and these *ripples* may have a specified maximum quotient between the highest and lowest gain.
- There will in practice not be an abrupt change of gain between passband and stopband, but a *transition band* where the frequency response will gradually change from its passband to its stopband value.

IIR filter design techniques

The designer can then trade off conflicting goals such as: small transition band, low order, low ripple amplitude or absence of ripples.

Design techniques for making these tradeoffs for analog filters (involving capacitors, resistors, coils) can also be used to design digital IIR filters:

Butterworth filters: Have no ripples, gain falls monotonically across the pass and transition band. Within the passband, the gain drops slowly down to $1-\sqrt{1/2}$ (-3 dB). Outside the passband, it drops asymptotically by a factor 2^N per octave ($N\cdot 20$ dB/decade).

Chebyshev type I filters: Distribute the gain error uniformly throughout the passband (equiripples) and drop off monotonically outside.

Chebyshev type II filters: Distribute the gain error uniformly throughout the stopband (equiripples) and drop off monotonically in the passband.

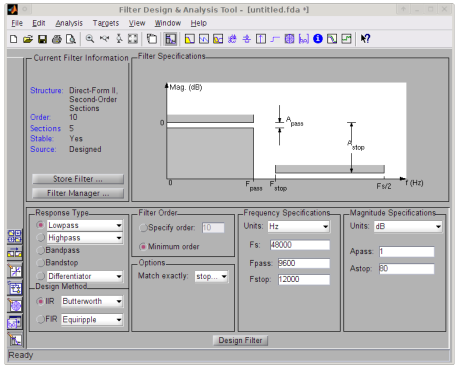
Elliptic filters (Cauer filters): Distribute the gain error as equiripples both in the passband and stopband. This type of filter is optimal in terms of the combination of the passband-gain tolerance, stopband-gain tolerance, and transition-band width that can be achieved at a given filter order.

IIR filter design in MATLAB

The aforementioned filter-design techniques are implemented in the MATLAB Signal Processing Toolbox in the functions `butter`, `cheby1`, `cheby2`, and `ellip`. They output the coefficients a_n and b_n of the difference equation that describes the filter.

These can be applied with `filter` to a sequence, or can be visualized with `zplane` as poles/zeros in the z-domain, with `impz` as an impulse response, and with `freqz` as an amplitude and phase spectrum.

The commands `sptool` and `fdatool` provide interactive GUIs to design digital filters.

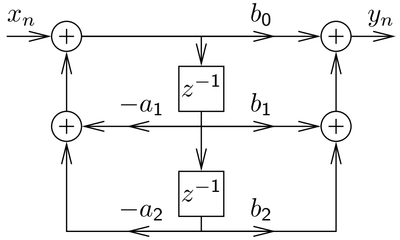


MATLAB fdatool

Cascade of filter sections

Higher-order IIR filters can be numerically unstable (quantization noise).

A commonly used trick is to split a higher-order IIR filter design into a cascade of l second-order (biquad) filter sections of the form:



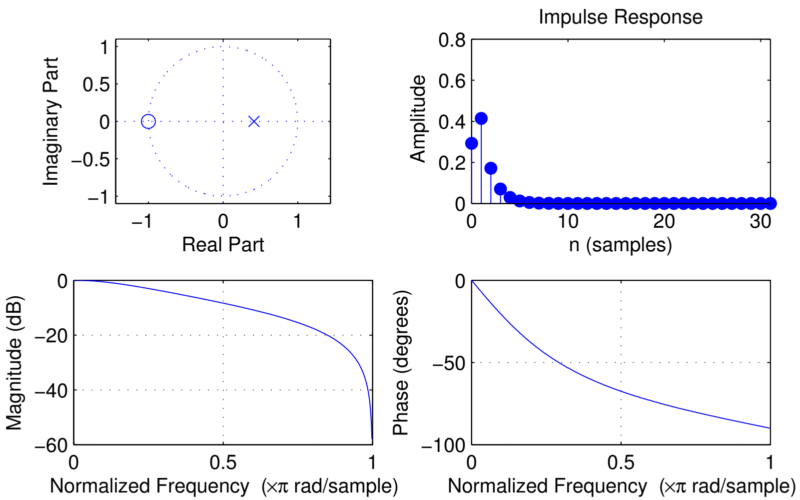
$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

Filter sections $H_1, H_2, \ldots, H_l$ are then applied sequentially to the input sequence, resulting in a filter

$$H(z) = \prod_{k=1}^{l} H_k(z) = \prod_{k=1}^{l} \frac{b_{k,0} + b_{k,1} z^{-1} + b_{k,2} z^{-2}}{1 + a_{k,1} z^{-1} + a_{k,2} z^{-2}}$$

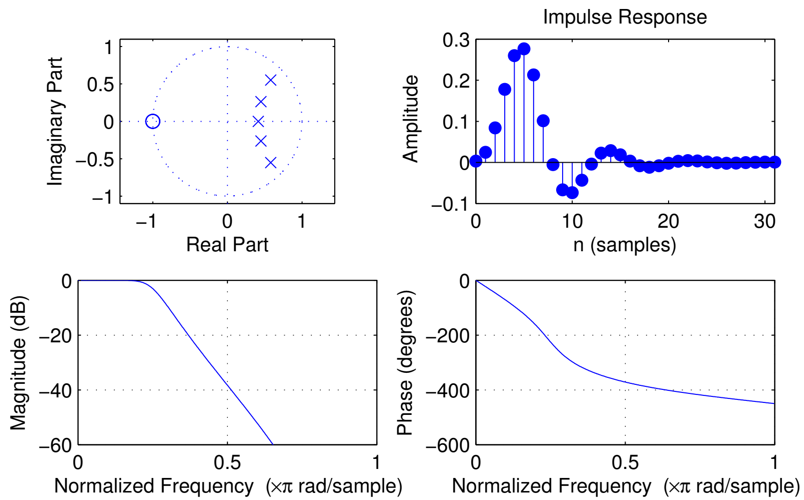
Each section implements one pair of poles and one pair of zeros. Jackson's algorithm for pairing poles and zeros into sections: pick the pole pair closest to the unit circle, and place it into a section along with the nearest pair of zeros; repeat until no poles are left.

Butterworth filter design example



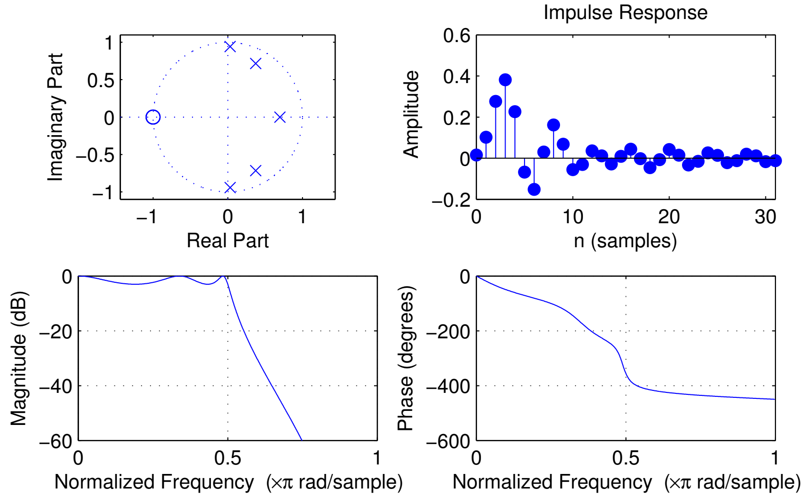
order: 1, cutoff frequency (−3 dB): 0.25 × $f_s/2$

Butterworth filter design example



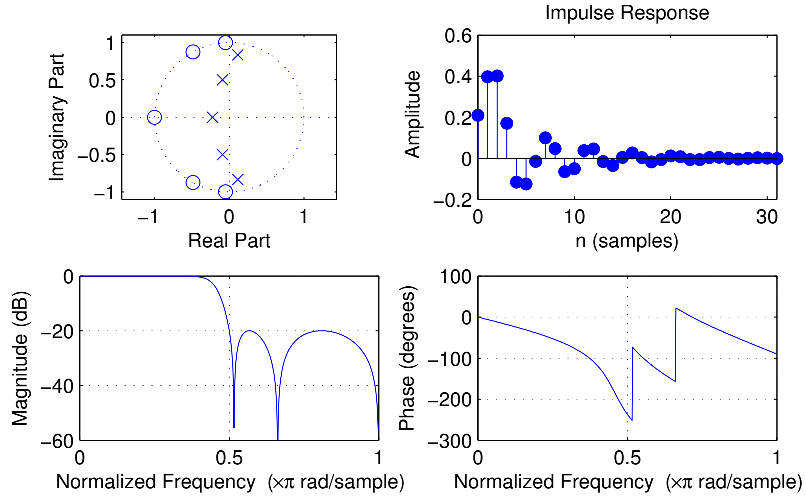
order: 5, cutoff frequency (−3 dB): 0.25 × $f_s/2$

Chebyshev type I filter design example



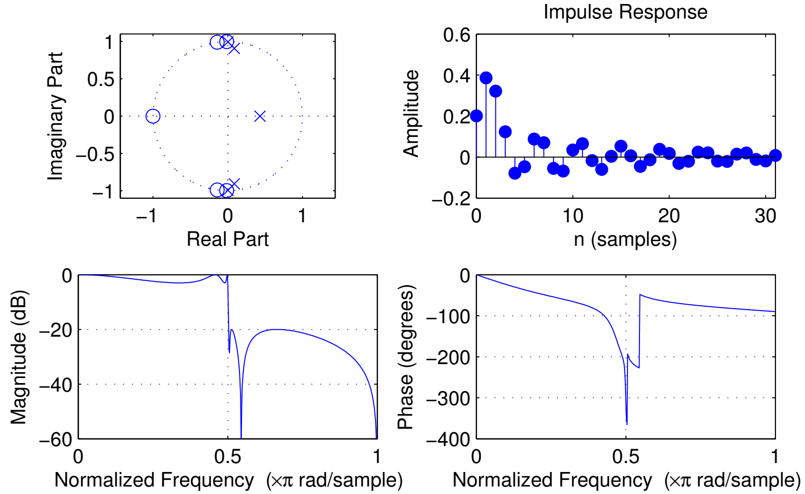
order: 5, cutoff frequency: $0.5 \times f_s/2$, pass-band ripple: -3 dB

Chebyshev type II filter design example



order: 5, cutoff frequency: $0.5 \times f_s/2$, stop-band ripple: -20 dB

Elliptic filter design example



order: 5, cutoff frequency: $0.5 \times f_s/2$, pass-band ripple: -3 dB, stop-band ripple: -20 dB

Random sequences and noise

A *discrete random sequence* $\{x_n\}$ is a sequence of numbers

$$\dots, x_{-2}, x_{-1}, x_0, x_1, x_2, \dots$$

where each value x_n is the outcome of a random variable $\mathbf{x}_n$ in a corresponding sequence of random variables

$$\dots, \mathbf{x}_{-2}, \mathbf{x}_{-1}, \mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \dots$$

Such a collection of random variables is called a *random process*. Each individual random variable $\mathbf{x}_n$ is characterized by its probability distribution function

$$P_{\mathbf{x}_n}(a) = \text{Prob}(\mathbf{x}_n \le a)$$

and the entire random process is characterized completely by all joint probability distribution functions

$$P_{\mathbf{x}_{n_1},\dots,\mathbf{x}_{n_k}}(a_1, \dots, a_k) = \text{Prob}(\mathbf{x}_{n_1} \le a_1 \wedge \dots \wedge \mathbf{x}_{n_k} \le a_k)$$

for all possible sets $\{\mathbf{x}_{n_1}, \dots, \mathbf{x}_{n_k}\}$.

Two random variables $\mathbf{x}_n$ and $\mathbf{x}_m$ are called *independent* if

$$P_{\mathbf{x}_n,\mathbf{x}_m}(a,b) = P_{\mathbf{x}_n}(a) \cdot P_{\mathbf{x}_m}(b)$$

and a random process is called *stationary* if

$$P_{\mathbf{x}_{n_1+l},\dots,\mathbf{x}_{n_k+l}}(a_1,\dots,a_k) = P_{\mathbf{x}_{n_1},\dots,\mathbf{x}_{n_k}}(a_1,\dots,a_k)$$

for all l, that is, if the probability distributions are time invariant.

The derivative $p_{\mathbf{x}_n}(a) = P'_{\mathbf{x}_n}(a)$ is called the *probability density function*, and helps us to define quantities such as the

- *expected value* $\mathsf{E}(\mathbf{x}_n) = \int a p_{\mathbf{x}_n}(a)\,\mathrm{d}a$
- *mean-square value* (average power) $\mathsf{E}(|\mathbf{x}_n|^2) = \int |a|^2 p_{\mathbf{x}_n}(a)\,\mathrm{d}a$
- *variance* $\mathsf{Var}(\mathbf{x}_n) = \mathsf{E}[|\mathbf{x}_n - \mathsf{E}(\mathbf{x}_n)|^2] = \mathsf{E}(|\mathbf{x}_n|^2) - |\mathsf{E}(\mathbf{x}_n)|^2$
- *correlation* $\mathsf{Cor}(\mathbf{x}_n, \mathbf{x}_m) = \mathsf{E}(\mathbf{x}_n \cdot \mathbf{x}_m^*)$
- *covariance* $\mathsf{Cov}(\mathbf{x}_n, \mathbf{x}_m) = \mathsf{E}[(\mathbf{x}_n - \mathsf{E}(\mathbf{x}_n)) \cdot (\mathbf{x}_m - \mathsf{E}(\mathbf{x}_m))^*] = \mathsf{E}(\mathbf{x}_n\mathbf{x}_m^*) - \mathsf{E}(\mathbf{x}_n)\mathsf{E}(\mathbf{x}_m)^*$

Remember that $\mathsf{E}(\cdot)$ is linear, that is $\mathsf{E}(a\mathbf{x}) = a\mathsf{E}(\mathbf{x})$ and $\mathsf{E}(\mathbf{x}+\mathbf{y}) = \mathsf{E}(\mathbf{x}) + \mathsf{E}(\mathbf{y})$. Also, $\mathsf{Var}(a\mathbf{x}) = a^2\mathsf{Var}(\mathbf{x})$ and, if $\mathbf{x}$ and $\mathbf{y}$ are independent, $\mathsf{Var}(\mathbf{x}+\mathbf{y}) = \mathsf{Var}(\mathbf{x}) + \mathsf{Var}(\mathbf{y})$.

A stationary random process $\{\mathbf{x}_n\}$ can be characterized by its mean value

$$m_x = \mathsf{E}(\mathbf{x}_n),$$

its variance

$$\sigma_x^2 = \mathsf{E}(|\mathbf{x}_n - m_x|^2) = \gamma_{xx}(0)$$

(σ_x is also called *standard deviation*), its *autocorrelation sequence*

$$\phi_{xx}(k) = \mathsf{E}(\mathbf{x}_{n+k} \cdot \mathbf{x}_n^*)$$

and its *autocovariance sequence*

$$\gamma_{xx}(k) = \mathsf{E}[(\mathbf{x}_{n+k} - m_x) \cdot (\mathbf{x}_n - m_x)^*] = \phi_{xx}(k) - |m_x|^2$$

A pair of stationary random processes $\{\mathbf{x}_n\}$ and $\{\mathbf{y}_n\}$ can, in addition, be characterized by its *crosscorrelation sequence*

$$\phi_{xy}(k) = \mathsf{E}(\mathbf{x}_{n+k} \cdot \mathbf{y}_n^*)$$

and its *crosscovariance sequence*

$$\gamma_{xy}(k) = \mathsf{E}[(\mathbf{x}_{n+k} - m_x) \cdot (\mathbf{y}_n - m_y)^*] = \phi_{xy}(k) - m_x m_y^*$$

Deterministic crosscorrelation sequence

For deterministic sequences $\{x_n\}$ and $\{y_n\}$, the *crosscorrelation sequence* is

$$c_{xy}(k) = \sum_{i=-\infty}^{\infty} x_{i+k} y_i.$$

After dividing through the overlapping length of the finite sequences involved, $c_{xy}(k)$ can be used to estimate, from a finite sample of a stationary random sequence, the underlying $\phi_{xy}(k)$. MATLAB's `xcorr` function does that with option `unbiased`.

If $\{x_n\}$ is similar to $\{y_n\}$, but lags l elements behind ($x_n \approx y_{n-l}$), then $c_{xy}(l)$ will be a peak in the crosscorrelation sequence. It is therefore widely calculated to locate shifted versions of a known sequence in another one.

The deterministic crosscorrelation sequence is a close cousin of the convolution, with just the second input sequence mirrored:

$$\{c_{xy}(n)\} = \{x_n\} * \{y_{-n}\}$$

It can therefore be calculated equally easily via the Fourier transform:

$$C_{xy}(f) = X(f) \cdot Y^*(f)$$

Swapping the input sequences mirrors the output sequence: $c_{xy}(k) = c_{yx}(-k)$.

Deterministic autocorrelation sequence

Equivalently, we define the deterministic autocorrelation sequence in the time domain as

$$c_{xx}(k) = \sum_{i=-\infty}^{\infty} x_{i+k} x_i.$$

which corresponds in the frequency domain to

$$C_{xx}(f) = X(f) \cdot X^*(f) = |X(f)|^2.$$

In other words, the Fourier transform $C_{xx}(f)$ of the autocorrelation sequence $\{c_{xx}(n)\}$ of a sequence $\{x_n\}$ is identical to the squared amplitudes of the Fourier transform, or *power spectrum*, of $\{x_n\}$.

This suggests, that the Fourier transform of the autocorrelation sequence of a random process might be a suitable way for defining the power spectrum of that random process.

What can we say about the phase in the Fourier spectrum of a time-invariant random process?

Filtered random sequences

Let $\{x_n\}$ be a random sequence from a stationary random process. The output

$$y_n = \sum_{k=-\infty}^{\infty} h_k \cdot x_{n-k} = \sum_{k=-\infty}^{\infty} h_{n-k} \cdot x_k$$

of an LTI applied to it will then be another random sequence, characterized by

$$m_y = m_x \sum_{k=-\infty}^{\infty} h_k$$

and

$$\phi_{yy}(k) = \sum_{i=-\infty}^{\infty} \phi_{xx}(k-i) c_{hh}(i), \quad \text{where} \quad \begin{array}{rcl} \phi_{xx}(k) & = & \mathsf{E}(\mathbf{x}_{n+k} \cdot \mathbf{x}_n^*) \\ c_{hh}(k) & = & \sum_{i=-\infty}^{\infty} h_{i+k} h_i. \end{array}$$

In other words:

$$\{y_n\} = \{h_n\} * \{x_n\} \quad \Rightarrow \quad \begin{array}{rcl} \{\phi_{yy}(n)\} & = & \{c_{hh}(n)\} * \{\phi_{xx}(n)\} \\ \Phi_{yy}(f) & = & |H(f)|^2 \cdot \Phi_{xx}(f) \end{array}$$

Similarly:

$$\{y_n\} = \{h_n\} * \{x_n\} \quad \Rightarrow \quad \begin{array}{rcl} \{\phi_{yx}(n)\} & = & \{h_n\} * \{\phi_{xx}(n)\} \\ \Phi_{yx}(f) & = & H(f) \cdot \Phi_{xx}(f) \end{array}$$

White noise

A random sequence $\{x_n\}$ is a *white noise* signal, if $m_x = 0$ and

$$\phi_{xx}(k) = \sigma_x^2 \delta_k.$$

The power spectrum of a white noise signal is flat:

$$\Phi_{xx}(f) = \sigma_x^2.$$

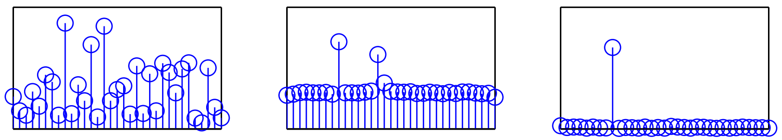
Application example:

Where an LTI $\{y_n\} = \{h_n\} * \{x_n\}$ can be observed to operate on white noise $\{x_n\}$ with $\phi_{xx}(k) = \sigma_x^2 \delta_k$, the crosscorrelation between input and output will reveal the impulse response of the system:

$$\phi_{yx}(k) = \sigma_x^2 \cdot h_k$$

where $\phi_{yx}(k) = \phi_{xy}(-k) = \mathsf{E}(\mathbf{y}_{n+k} \cdot \mathbf{x}_n^*)$.

DFT averaging



The above diagrams show different types of spectral estimates of a sequence $x_i = \sin(2\pi \mathrm{j} \times 8/64) + \sin(2\pi \mathrm{j} \times 14.32/64) + n_i$ with $\phi_{nn}(i) = 4\delta_i$.

Left is a single 64-element DFT of $\{x_i\}$ (with rectangular window). The flat spectrum of white noise is only an expected value. In a single discrete Fourier transform of such a sequence, the significant variance of the noise spectrum becomes visible. It almost drowns the two peaks from sine waves.

After cutting $\{x_i\}$ into 1000 windows of 64 elements each, calculating their DFT, and plotting the average of their absolute values, the centre figure shows an approximation of the expected value of the amplitude spectrum, with a flat noise floor. Taking the absolute value before spectral averaging is called *incoherent averaging*, as the phase information is thrown away.

The rightmost figure was generated from the same set of 1000 windows, but this time the complex values of the DFTs were averaged *before* the absolute value was taken. This is called *coherent averaging* and, because of the linearity of the DFT, identical to first averaging the 1000 windows and then applying a single DFT and taking its absolute value. The windows start 64 samples apart. Only periodic waveforms with a period that divides 64 are not averaged away. This periodic averaging step suppresses both the noise and the second sine wave.

Periodic averaging

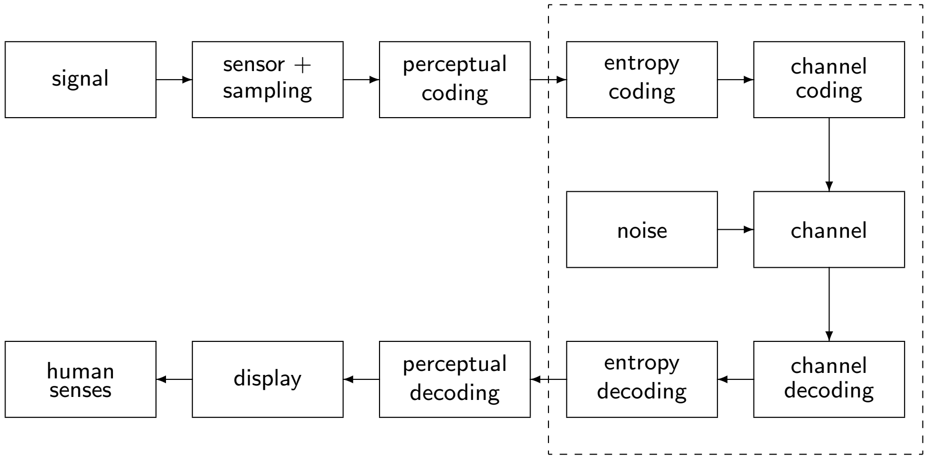
If a zero-mean signal $\{x_i\}$ has a periodic component with period p, the periodic component can be isolated by *periodic averaging*:

$$\bar{x}_i = \lim_{k \to \infty} \frac{1}{2k+1} \sum_{n=-k}^{k} x_{i+pn}$$

Periodic averaging corresponds in the time domain to convolution with a Dirac comb $\sum_n \delta_{i-pn}$. In the frequency domain, this means multiplication with a Dirac comb that eliminates all frequencies but multiples of $1/p$.

Audiovisual data compression

Structure of modern audiovisual communication systems:



Audio-visual lossy coding today typically consists of these steps:

- A *transducer* converts the original stimulus into a voltage.
- This analog signal is then *sampled and quantized*.
  The digitization parameters (sampling frequency, quantization levels) are preferably chosen generously beyond the ability of human senses or output devices.
- The digitized sensor-domain signal is then *transformed into a perceptual domain*.
  This step often mimics some of the first neural processing steps in humans.
- This signal is *quantized* again, based on a *perceptual model* of what level of quantization-noise humans can still sense.
- The resulting quantized levels may still be highly statistically dependent. A *prediction or decorrelation transform* exploits this and produces a less dependent symbol sequence of lower entropy.
- An *entropy coder* turns that into an apparently-random bit string, whose length approximates the remaining entropy.

The first neural processing steps in humans are in effect often a kind of decorrelation transform; our eyes and ears were optimized like any other AV communications system. This allows us to use the same transform for decorrelating and transforming into a perceptually relevant domain.

Outline of the remaining lectures

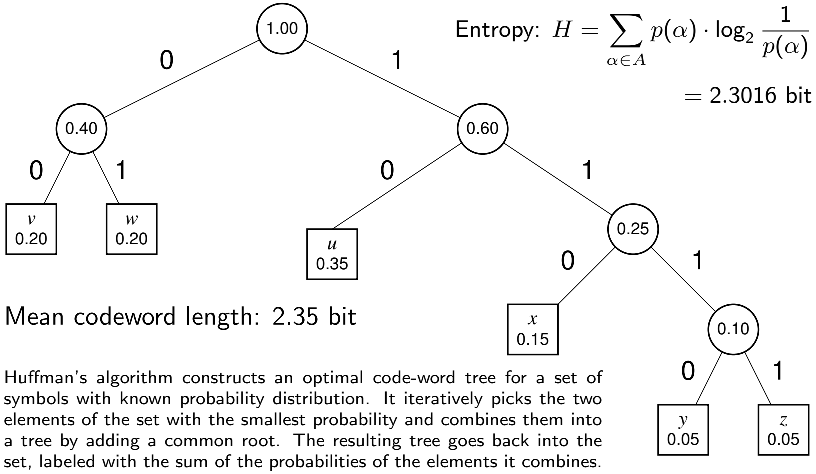
- Quick review of entropy coding
- Transform coding: techniques for converting sequences of highly-dependent symbols into less-dependent lower-entropy sequences.
  - run-length coding
  - decorrelation, Karhunen-Loève transform (PCA)
  - other orthogonal transforms (especially DCT)
- Introduction to some characteristics and limits of human senses
  - perceptual scales and sensitivity limits
  - colour vision
  - human hearing limits, critical bands, audio masking
- Quantization techniques to remove information that is irrelevant to human senses

- Image and audio coding standards
  - A/μ-law coding (digital telephone network)
  - JPEG
  - (MPEG video)
  - MPEG audio

Literature

- D. Salomon: A guide to data compression methods. ISBN 0387952608, 2002.
- L. Gulick, G. Gescheider, R. Frisina: Hearing. ISBN 0195043073, 1989.
- H. Schiffman: Sensation and perception. ISBN 0471082082, 1982.

Entropy coding review – Huffman



Entropy: $H = \sum_{\alpha \in A} p(\alpha) \cdot \log_2 \frac{1}{p(\alpha)}$

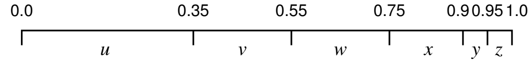
$= 2.3016$ bit

Mean codeword length: 2.35 bit

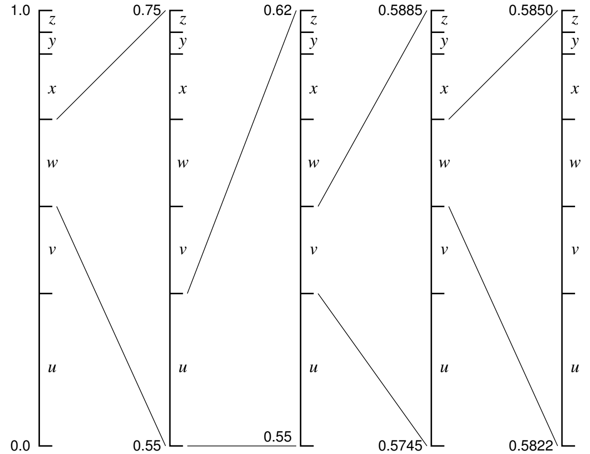
Huffman's algorithm constructs an optimal code-word tree for a set of symbols with known probability distribution. It iteratively picks the two elements of the set with the smallest probability and combines them into a tree by adding a common root. The resulting tree goes back into the set, labeled with the sum of the probabilities of the elements it combines. The algorithm terminates when less than two elements are left.

Entropy coding review – arithmetic coding

Partition [0,1] according to symbol probabilities:



Encode text $wuvw\ldots$ as numeric value (0.58...) in nested intervals:



Arithmetic coding

Several advantages:

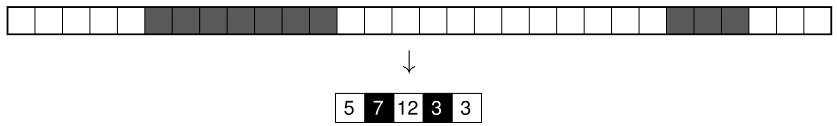
- Length of output bitstring can approximate the theoretical information content of the input to within 1 bit.
- Performs well with probabilities > 0.5, where the information per symbol is less than one bit.
- Interval arithmetic makes it easy to change symbol probabilities (no need to modify code-word tree) $\Rightarrow$ convenient for adaptive coding

Can be implemented efficiently with fixed-length arithmetic by rounding probabilities and shifting out leading digits as soon as leading zeros appear in interval size. Usually combined with adaptive probability estimation.

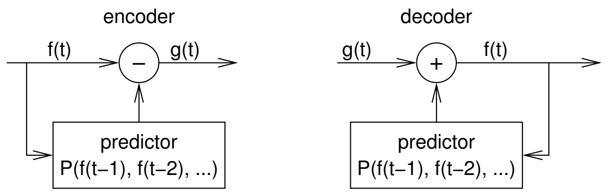
Huffman coding remains popular because of its simplicity and lack of patent-licence issues.

Coding of sources with memory and correlated symbols

Run-length coding:



Predictive coding:



$$\text{Delta coding (DPCM):} \qquad P(x) = x$$

$$\text{Linear predictive coding:} \qquad P(x_1, \ldots, x_n) = \sum_{i=1}^{n} a_i x_i$$

Old (Group 3 MH) fax code

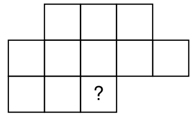
- Run-length encoding plus modified Huffman code
- Fixed code table (from eight sample pages)
- separate codes for runs of white and black pixels
- *termination code* in the range 0–63 switches between black and white code
- *makeup code* can extend length of a run by a multiple of 64
- termination run length 0 needed where run length is a multiple of 64
- single white column added on left side before transmission
- makeup codes above 1728 equal for black and white
- 12-bit end-of-line marker: 000000000001 (can be prefixed by up to seven zero-bits to reach next byte boundary)

Example: line with 2 w, 4 b, 200 w, 3 b, EOL → 1000|011|010111|10011|10|000000000001

pixels	white code	black code
0	00110101	0000110111
1	000111	010
2	0111	11
3	1000	10
4	1011	011
5	1100	0011
6	1110	0010
7	1111	00011
8	10011	000101
9	10100	000100
10	00111	0000100
11	01000	0000101
12	001000	0000111
13	000011	00000100
14	110100	00000111
15	110101	000011000
16	101010	0000010111
...	...	...
63	00110100	000001100111
64	11011	0000001111
128	10010	000011001000
192	010111	000011001001
...	...	...
1728	010011011	0000001100101

Modern (JBIG) fax code

Performs context-sensitive arithmetic coding of binary pixels. Both encoder and decoder maintain statistics on how the black/white probability of each pixel depends on these 10 previously transmitted neighbours:



Based on the counted numbers n_{black} and n_{white} of how often each pixel value has been encountered so far in each of the 1024 contexts, the probability for the next pixel being black is estimated as

$$p_{\text{black}} = \frac{n_{\text{black}} + 1}{n_{\text{white}} + n_{\text{black}} + 2}$$

The encoder updates its estimate only after the newly counted pixel has been encoded, such that the decoder knows the exact same statistics.

Joint Bi-level Expert Group: International Standard ISO 11544, 1993.
Example implementation: `http://www.cl.cam.ac.uk/~mgk25/jbigkit/`

Statistical dependence

Random variables X, Y are *dependent* iff $\exists x, y$:

$$P(X = x \wedge Y = y) \neq P(X = x) \cdot P(Y = y).$$

If X, Y are dependent, then

$$\begin{aligned} \Rightarrow \quad & \exists x, y: \; P(X = x \,|\, Y = y) \neq P(X = x) \vee \\ & \qquad\quad\;\; P(Y = y \,|\, X = x) \neq P(Y = y) \\ \Rightarrow \quad & H(X|Y) < H(X) \;\vee\; H(Y|X) < H(Y) \end{aligned}$$

Application

Where x is the value of the next symbol to be transmitted and y is the vector of all symbols transmitted so far, accurate knowledge of the conditional probability $P(X = x \,|\, Y = y)$ will allow a transmitter to remove all redundancy.

An application example of this approach is JBIG, but there y is limited to 10 past single-bit pixels and $P(X = x \,|\, Y = y)$ is only an estimate.

Practical limits of measuring conditional probabilities

The practical estimation of conditional probabilities, in their most general form, based on statistical measurements of example signals, quickly reaches practical limits. JBIG needs an array of only $2^{11} = 2048$ counting registers to maintain estimator statistics for its 10-bit context.

If we wanted to encode each 24-bit pixel of a colour image based on its statistical dependence of the full colour information from just ten previous neighbour pixels, the required number of

$$(2^{24})^{11} \approx 3 \times 10^{80}$$

registers for storing each probability will exceed the estimated number of particles in this universe. (Neither will we encounter enough pixels to record statistically significant occurrences in all $(2^{24})^{10}$ contexts.)

This example is far from excessive. It is easy to show that in colour images, pixel values show statistical significant dependence across colour channels, and across locations more than eight pixels apart.

A simpler approximation of dependence is needed: correlation.

Correlation

Two random variables $X \in \mathbb{R}$ and $Y \in \mathbb{R}$ are *correlated* iff

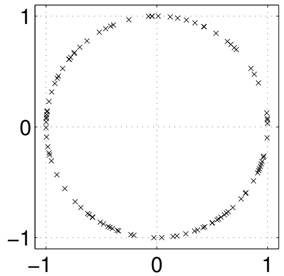
$$\mathsf{E}\{[X - \mathsf{E}(X)] \cdot [Y - \mathsf{E}(Y)]\} \neq 0$$

where $\mathsf{E}(\cdots)$ denotes the *expected value* of a random-variable term.

Correlation implies dependence, but dependence does not always lead to correlation (see example to the right).

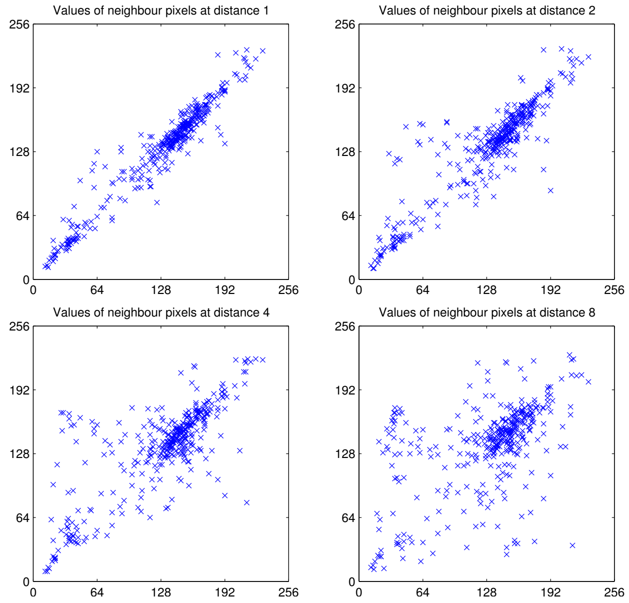
However, most dependency in audio-visual data is a consequence of correlation, which is algorithmically much easier to exploit.

Dependent but not correlated:



Positive correlation: higher $X \Leftrightarrow$ higher Y, lower $X \Leftrightarrow$ lower Y
Negative correlation: lower $X \Leftrightarrow$ higher Y, higher $X \Leftrightarrow$ lower Y

Correlation of neighbour pixels



Covariance and correlation

We define the *covariance* of two random variables X and Y as

$$\mathsf{Cov}(X,Y) = \mathsf{E}\{[X - \mathsf{E}(X)] \cdot [Y - \mathsf{E}(Y)]\} = \mathsf{E}(X \cdot Y) - \mathsf{E}(X) \cdot \mathsf{E}(Y)$$

and the *variance* as $\mathsf{Var}(X) = \mathsf{Cov}(X,X) = \mathsf{E}\{[X - \mathsf{E}(X)]^2\}$.

The *Pearson correlation coefficient*

$$\rho_{X,Y} = \frac{\mathsf{Cov}(X,Y)}{\sqrt{\mathsf{Var}(X) \cdot \mathsf{Var}(Y)}}$$

is a normalized form of the covariance. It is limited to the range $[-1, 1]$.

If the correlation coefficient has one of the values $\rho_{X,Y} = \pm 1$, this implies that X and Y are exactly linearly dependent, i.e. $Y = aX + b$, with $a = \mathsf{Cov}(X,Y)/\mathsf{Var}(X)$ and $b = \mathsf{E}(Y) - \mathsf{E}(X)$.

Covariance Matrix

For a random vector $\mathbf{X} = (X_1, X_2, \ldots, X_n) \in \mathbb{R}^n$ we define the *covariance matrix*

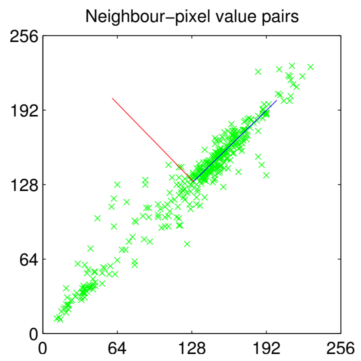
$$\mathsf{Cov}(\mathbf{X}) = \mathsf{E}\left((\mathbf{X} - \mathsf{E}(\mathbf{X})) \cdot (\mathbf{X} - \mathsf{E}(\mathbf{X}))^{\mathsf{T}}\right) = (\mathsf{Cov}(X_i, X_j))_{i,j} =$$

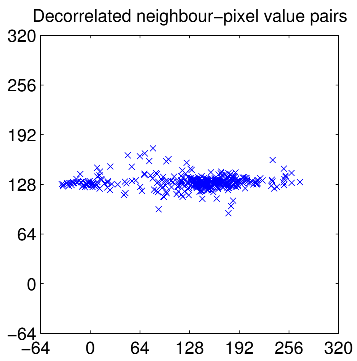
$$\begin{pmatrix} \mathsf{Cov}(X_1,X_1) & \mathsf{Cov}(X_1,X_2) & \mathsf{Cov}(X_1,X_3) & \cdots & \mathsf{Cov}(X_1,X_n) \\ \mathsf{Cov}(X_2,X_1) & \mathsf{Cov}(X_2,X_2) & \mathsf{Cov}(X_2,X_3) & \cdots & \mathsf{Cov}(X_2,X_n) \\ \mathsf{Cov}(X_3,X_1) & \mathsf{Cov}(X_3,X_2) & \mathsf{Cov}(X_3,X_3) & \cdots & \mathsf{Cov}(X_3,X_n) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathsf{Cov}(X_n,X_1) & \mathsf{Cov}(X_n,X_2) & \mathsf{Cov}(X_n,X_3) & \cdots & \mathsf{Cov}(X_n,X_n) \end{pmatrix}$$

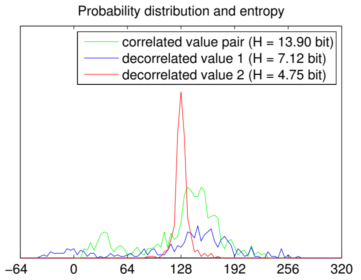
The elements of a random vector $\mathbf{X}$ are uncorrelated if and only if $\mathsf{Cov}(\mathbf{X})$ is a diagonal matrix.

$\mathsf{Cov}(X,Y) = \mathsf{Cov}(Y,X)$, so all covariance matrices are *symmetric*: $\mathsf{Cov}(\mathbf{X}) = \mathsf{Cov}^{\mathsf{T}}(\mathbf{X})$.

Decorrelation by coordinate transform







Idea: Take the values of a group of correlated symbols (e.g., neighbour pixels) as a random vector. Find a coordinate transform (multiplication with an orthonormal matrix) that leads to a new random vector whose covariance matrix is diagonal. The vector components in this transformed coordinate system will no longer be correlated. This will hopefully reduce the entropy of some of these components.

Theorem: Let $\mathbf{X} \in \mathbb{R}^n$ and $\mathbf{Y} \in \mathbb{R}^n$ be random vectors that are linearly dependent with $\mathbf{Y} = A\mathbf{X} + b$, where $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$ are constants. Then

$$\begin{aligned} \mathsf{E}(\mathbf{Y}) &= A \cdot \mathsf{E}(\mathbf{X}) + b \\ \mathsf{Cov}(\mathbf{Y}) &= A \cdot \mathsf{Cov}(\mathbf{X}) \cdot A^\mathsf{T} \end{aligned}$$

Proof: The first equation follows from the linearity of the expected-value operator $\mathsf{E}(\cdot)$, as does $\mathsf{E}(A \cdot \mathbf{X} \cdot B) = A \cdot \mathsf{E}(\mathbf{X}) \cdot B$ for matrices A, B. With that, we can transform

$$\begin{aligned} \mathsf{Cov}(\mathbf{Y}) &= \mathsf{E}\left((\mathbf{Y} - \mathsf{E}(\mathbf{Y})) \cdot (\mathbf{Y} - \mathsf{E}(\mathbf{Y}))^\mathsf{T}\right) \\ &= \mathsf{E}\left((A\mathbf{X} - A\mathsf{E}(\mathbf{X})) \cdot (A\mathbf{X} - A\mathsf{E}(\mathbf{X}))^\mathsf{T}\right) \\ &= \mathsf{E}\left(A(\mathbf{X} - \mathsf{E}(\mathbf{X})) \cdot (\mathbf{X} - \mathsf{E}(\mathbf{X}))^\mathsf{T} A^\mathsf{T}\right) \\ &= A \cdot \mathsf{E}\left((\mathbf{X} - \mathsf{E}(\mathbf{X})) \cdot (\mathbf{X} - \mathsf{E}(\mathbf{X}))^\mathsf{T}\right) \cdot A^\mathsf{T} \\ &= A \cdot \mathsf{Cov}(\mathbf{X}) \cdot A^\mathsf{T} \end{aligned}$$

Quick review: eigenvectors and eigenvalues

We are given a square matrix $A \in \mathbb{R}^{n\times n}$. The vector $x \in \mathbb{R}^n$ is an *eigenvector* of A if there exists a scalar value $\lambda \in \mathbb{R}$ such that

$$Ax = \lambda x.$$

The corresponding λ is the *eigenvalue* of A associated with x.

The length of an eigenvector is irrelevant, as any multiple of it is also an eigenvector. Eigenvectors are in practice normalized to length 1.

Spectral decomposition

Any real, *symmetric* matrix $A = A^\mathsf{T} \in \mathbb{R}^{n\times n}$ can be diagonalized into the form

$$A = U\Lambda U^\mathsf{T},$$

where $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \dots, \lambda_n)$ is the diagonal matrix of the ordered eigenvalues of A (with $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$), and the columns of U are the n corresponding orthonormal eigenvectors of A.

Karhunen-Loève transform (KLT)

We are given a random vector variable $\mathbf{X} \in \mathbb{R}^n$. The correlation of the elements of $\mathbf{X}$ is described by the covariance matrix $\mathsf{Cov}(\mathbf{X})$.

How can we find a transform matrix A that decorrelates $\mathbf{X}$, i.e. that turns $\mathsf{Cov}(A\mathbf{X}) = A \cdot \mathsf{Cov}(\mathbf{X}) \cdot A^\mathsf{T}$ into a diagonal matrix? A would provide us the transformed representation $\mathbf{Y} = A\mathbf{X}$ of our random vector, in which all elements are mutually uncorrelated.

Note that $\mathsf{Cov}(\mathbf{X})$ is symmetric. It therefore has n real eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ and a set of associated mutually orthogonal eigenvectors $b_1, b_2, \dots, b_n$ of length 1 with

$$\mathsf{Cov}(\mathbf{X}) b_i = \lambda_i b_i.$$

We convert this set of equations into matrix notation using the matrix $B = (b_1, b_2, \dots, b_n)$ that has these eigenvectors as columns and the diagonal matrix $D = \mathrm{diag}(\lambda_1, \lambda_2, \dots, \lambda_n)$ that consists of the corresponding eigenvalues:

$$\mathsf{Cov}(\mathbf{X}) B = BD$$

B is orthonormal, that is $BB^{\mathsf{T}} = I$.

Multiplying the above from the right with B^{T} leads to the *spectral decomposition*

$$\mathrm{Cov}(\mathbf{X}) = BDB^{\mathsf{T}}$$

of the covariance matrix. Similarly multiplying instead from the left with B^{T} leads to

$$B^{\mathsf{T}}\mathrm{Cov}(\mathbf{X})B = D$$

and therefore shows with

$$\mathrm{Cov}(B^{\mathsf{T}}\mathbf{X}) = D$$

that the eigenvector matrix B^{T} is the wanted transform.

The *Karhunen-Loève transform* (also known as *Hotelling transform* or *Principal Component Analysis*) is the multiplication of a correlated random vector $\mathbf{X}$ with the orthonormal eigenvector matrix B^{T} from the spectral decomposition $\mathrm{Cov}(\mathbf{X}) = BDB^{\mathsf{T}}$ of its covariance matrix. This leads to a decorrelated random vector $B^{\mathsf{T}}\mathbf{X}$ whose covariance matrix is diagonal.

Karhunen-Loève transform example I

colour image | red channel | green channel | blue channel

The colour image (left) has $m = r^2$ pixels, each of which is an $n = 3$-dimensional RGB vector

$$I_{x,y} = (r_{x,y}, g_{x,y}, b_{x,y})^{\mathsf{T}}$$

The three rightmost images show each of these colour planes separately as a black/white image.

We want to apply the KLT on a set of such $\mathbb{R}^n$ colour vectors. Therefore, we reformat the image I into an $n \times m$ matrix of the form

$$S = \begin{pmatrix} r_{1,1} & r_{1,2} & r_{1,3} & \cdots & r_{r,r} \\ g_{1,1} & g_{1,2} & g_{1,3} & \cdots & g_{r,r} \\ b_{1,1} & b_{1,2} & b_{1,3} & \cdots & b_{r,r} \end{pmatrix}$$

We can now define the mean colour vector

$$\bar{S}_c = \frac{1}{m}\sum_{i=1}^{m} S_{c,i}, \quad \bar{S} = \begin{pmatrix} 0.4839 \\ 0.4456 \\ 0.3411 \end{pmatrix}$$

and the covariance matrix

$$C_{c,d} = \frac{1}{m-1}\sum_{i=1}^{m}(S_{c,i} - \bar{S}_c)(S_{d,i} - \bar{S}_d)$$

$$C = \begin{pmatrix} 0.0328 & 0.0256 & 0.0160 \\ 0.0256 & 0.0216 & 0.0140 \\ 0.0160 & 0.0140 & 0.0109 \end{pmatrix}$$

[When estimating a covariance from a number of samples, the sum is divided by the number of samples *minus one*. This takes into account the variance of the mean $\bar{S}_c$, which is not the exact expected value, but only an estimate of it.]

The resulting covariance matrix has three eigenvalues 0.0622, 0.0025, and 0.0006

Karhunen-Loève transform example I

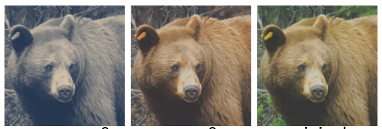
Before KLT:

red | green | blue

After KLT:

u | v | w

Projections on eigenvector subspaces:



$v = w = 0$ | $w = 0$ | original

We finally apply the orthogonal 3×3 transform matrix U, which we just used to diagonalize the covariance matrix, to the entire image:

$$T = U^{\mathsf{T}} \cdot \left[S - \begin{pmatrix} \bar{S}_1 & \bar{S}_1 & \cdots & \bar{S}_1 \\ \bar{S}_2 & \bar{S}_2 & \cdots & \bar{S}_2 \\ \bar{S}_3 & \bar{S}_3 & \cdots & \bar{S}_3 \end{pmatrix} \right] + \begin{pmatrix} \bar{S}_1 & \bar{S}_1 & \cdots & \bar{S}_1 \\ \bar{S}_2 & \bar{S}_2 & \cdots & \bar{S}_2 \\ \bar{S}_3 & \bar{S}_3 & \cdots & \bar{S}_3 \end{pmatrix}$$

The resulting transformed image

$$T = \begin{pmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{r,r} \\ v_{1,1} & v_{1,2} & v_{1,3} & \cdots & v_{r,r} \\ w_{1,1} & w_{1,2} & w_{1,3} & \cdots & w_{r,r} \end{pmatrix}$$

consists of three new "colour" planes whose pixel values have no longer any correlation to the pixels at the same coordinates in another plane. [The bear disappeared from the last of these (w), which represents mostly some of the green grass in the background.]

Spatial correlation

The previous example used the Karhunen-Loève transform in order to eliminate correlation between colour planes. While this is of some relevance for image compression, far more correlation can be found between neighbour pixels within each colour plane.

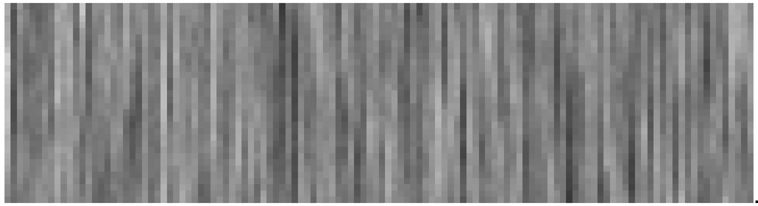
In order to exploit such correlation using the KLT, the sample set has to be extended from individual pixels to entire images. The underlying calculation is the same as in the preceeding example, but this time the columns of S are entire (monochrome) images. The rows are the different images found in the set of test images that we use to examine typical correlations between neighbour pixels.

In other words, we use the same formulas as in the previous example, but this time n is the number of pixels per image and m is the number of sample images. The Karhunen-Loève transform is here no longer a rotation in a 3-dimensional colour space, but it operates now in a *much* larger vector space that has as many dimensions as an image has pixels.

To keep things simple, we look in the next experiment only at $m = 9000$ 1-dimensional "images" with $n = 32$ pixels each. As a further simplification, we use not real images, but random noise that was filtered such that its amplitude spectrum is proportional to $1/f$, where f is the frequency. The result would be similar in a sufficiently large collection of real test images.

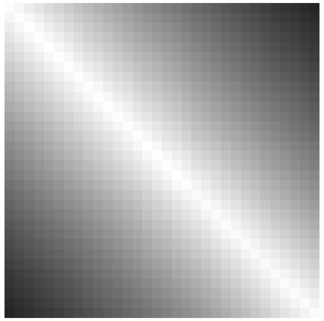
Karhunen-Loève transform example II

Matrix columns of S filled with samples of $1/f$ filtered noise

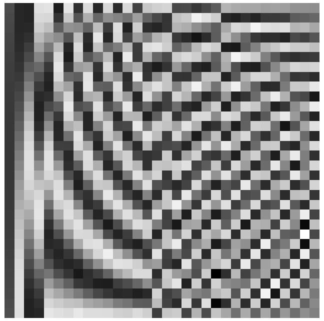


$\ldots$

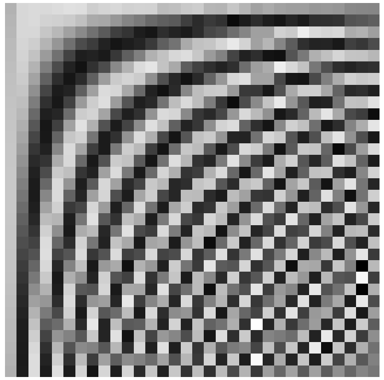
Covariance matrix C



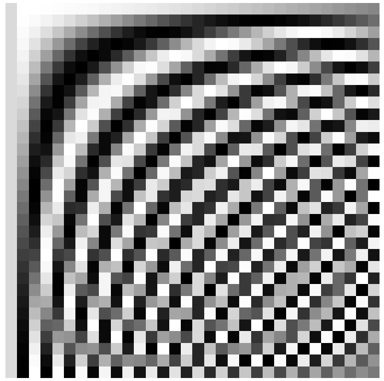
Matrix U with eigenvector columns



Matrix U' with normalised KLT eigenvector columns



Matrix with Discrete Cosine Transform base vector columns



Breakthrough: Ahmed/Natarajan/Rao discovered the DCT as an excellent approximation of the KLT for typical photographic images, but far more efficient to calculate.

Ahmed, Natarajan, Rao: Discrete Cosine Transform. IEEE Transactions on Computers, Vol. 23, January 1974, pp. 90–93.

Discrete cosine transform (DCT)

The forward and inverse discrete cosine transform

$$
\begin{aligned}
S(u) &= \frac{C(u)}{\sqrt{N/2}} \sum_{x=0}^{N-1} s(x) \cos \frac{(2x+1)u\pi}{2N} \\
s(x) &= \sum_{u=0}^{N-1} \frac{C(u)}{\sqrt{N/2}} S(u) \cos \frac{(2x+1)u\pi}{2N}
\end{aligned}
$$

with

$$
C(u) = \begin{cases} \frac{1}{\sqrt{2}} & u = 0 \\ 1 & u > 0 \end{cases}
$$

is an orthonormal transform:

$$
\sum_{x=0}^{N-1} \frac{C(u)}{\sqrt{N/2}} \cos \frac{(2x+1)u\pi}{2N} \cdot \frac{C(u')}{\sqrt{N/2}} \cos \frac{(2x+1)u'\pi}{2N} = \begin{cases} 1 & u = u' \\ 0 & u \neq u' \end{cases}
$$

Discrete cosine transform – 2D

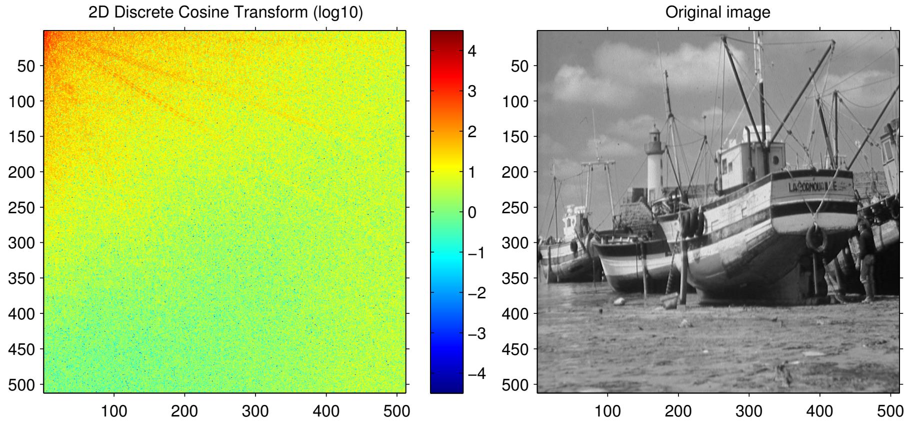
The 2-dimensional variant of the DCT applies the 1-D transform on both rows and columns of an image:

$$
S(u,v) = \frac{C(u)}{\sqrt{N/2}} \frac{C(v)}{\sqrt{N/2}} \cdot \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} s(x,y) \cos \frac{(2x+1)u\pi}{2N} \cos \frac{(2y+1)v\pi}{2N}
$$

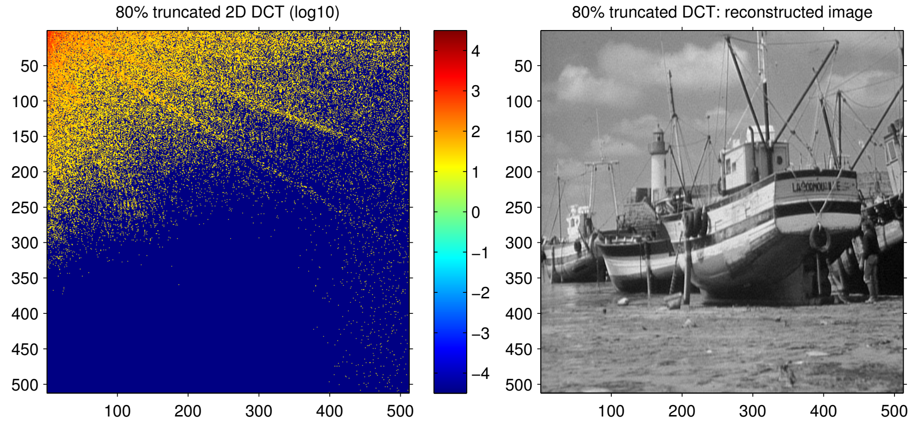
$$
s(x,y) = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} \frac{C(u)}{\sqrt{N/2}} \frac{C(v)}{\sqrt{N/2}} \cdot S(u,v) \cos \frac{(2x+1)u\pi}{2N} \cos \frac{(2y+1)v\pi}{2N}
$$

A range of fast algorithms have been found for calculating 1-D and 2-D DCTs (e.g., Ligtenberg/Vetterli).

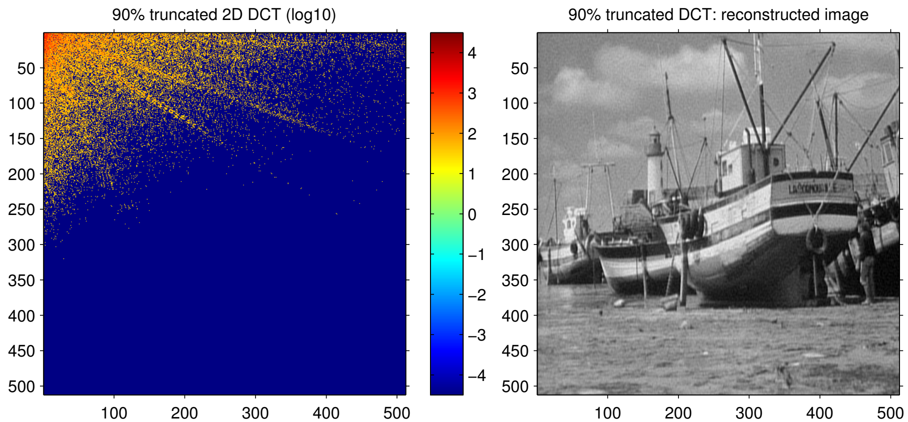
Whole-image DCT



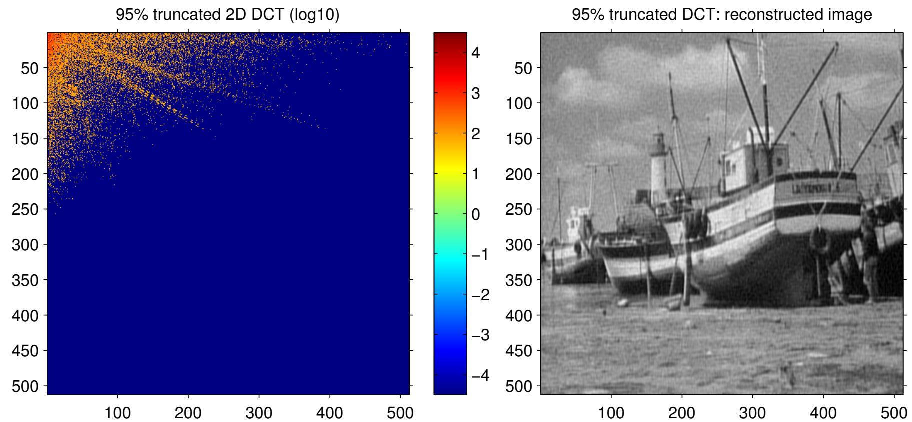
Whole-image DCT, 80% coefficient cutoff



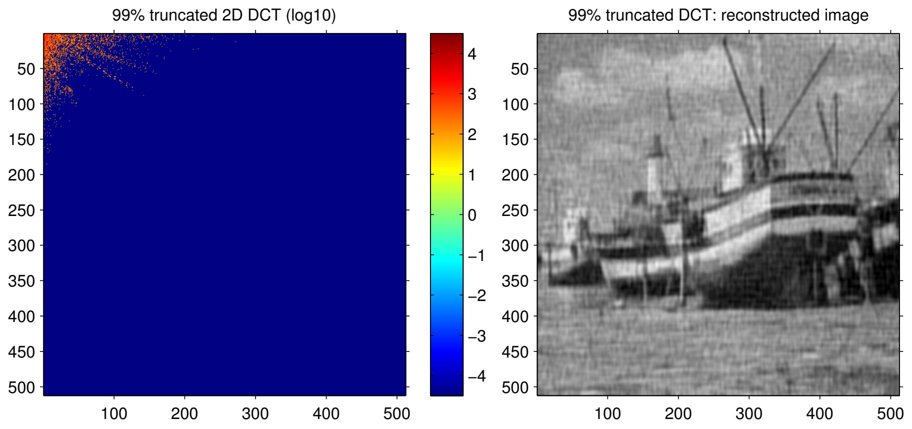
Whole-image DCT, 90% coefficient cutoff



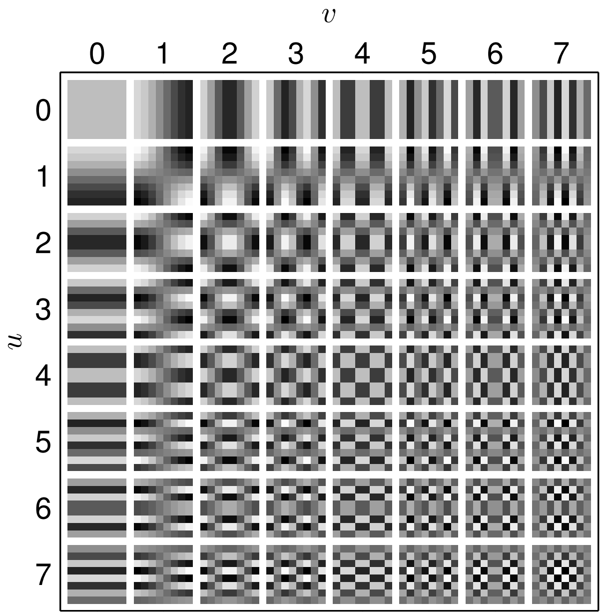
Whole-image DCT, 95% coefficient cutoff



Whole-image DCT, 99% coefficient cutoff



Base vectors of 8×8 DCT



Psychophysics of perception

Sensation limit (SL) = lowest intensity stimulus that can still be perceived

Difference limit (DL) = smallest perceivable stimulus difference at given intensity level

Weber's law

Difference limit $\Delta\phi$ is proportional to the intensity ϕ of the stimulus (except for a small correction constant a, to describe deviation of experimental results near SL):

$$\Delta\phi = c \cdot (\phi + a)$$

Fechner's scale

Define a perception intensity scale ψ using the sensation limit ϕ_0 as the origin and the respective difference limit $\Delta\phi = c \cdot \phi$ as a unit step. The result is a logarithmic relationship between stimulus intensity and scale value:

$$\psi = \log_c \frac{\phi}{\phi_0}$$

Fechner's scale matches older subjective intensity scales that follow differentiability of stimuli, e.g. the astronomical magnitude numbers for star brightness introduced by Hipparchos (≈150 BC).

Stevens' power law

A sound that is 20 DL over SL is perceived as more than twice as loud as one that is 10 DL over SL, i.e. Fechner's scale does not describe well perceived intensity. A rational scale attempts to reflect subjective relations perceived between different values of stimulus intensity ϕ. Stanley Smith Stevens observed that such rational scales ψ follow a power law:

$$\psi = k \cdot (\phi - \phi_0)^a$$

Example coefficients a: brightness 0.33, loudness 0.6, heaviness 1.45, temperature (warmth) 1.6.

RGB video colour coordinates

Hardware interface (VGA): red, green, blue signals with 0–0.7 V

Electron-beam current and photon count of cathode-ray displays are roughly proportional to $(v - v_0)^\gamma$, where v is the video-interface or control-grid voltage and γ is a device parameter that is typically in the range 1.5–3.0. In broadcast TV, this CRT non-linearity is compensated electronically in TV cameras. A welcome side effect is that it approximates Stevens' scale and therefore helps to reduce perceived noise.

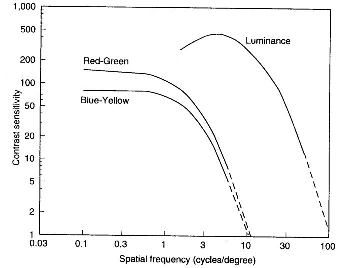
Software interfaces map RGB voltage linearly to $\{0, 1, \ldots, 255\}$ or 0–1.

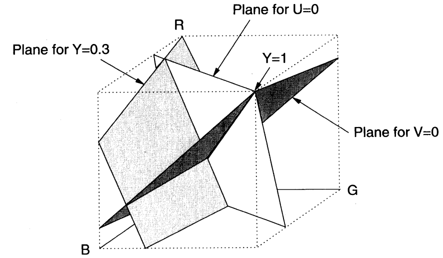
How numeric RGB values map to colour and luminosity depends at present still highly on the hardware and sometimes even on the operating system or device driver.

The new specification "sRGB" aims to standardize the meaning of an RGB value with the parameter $\gamma = 2.2$ and with standard colour coordinates of the three primary colours.

http://www.w3.org/Graphics/Color/sRGB, IEC 61966

YUV video colour coordinates





The human eye processes colour and luminosity at different resolutions. To exploit this phenomenon, many image transmission systems use a colour space with a luminance coordinate

$$Y = 0.3R + 0.6G + 0.1B$$

and colour ("chrominance") components

$$\begin{aligned} V &= R - Y = 0.7R - 0.6G - 0.1B \\ U &= B - Y = -0.3R - 0.6G + 0.9B \end{aligned}$$

YUV transform example

original

Y channel

U and V channels

The centre image shows only the luminance channel as a black/white image. In the right image, the luminance channel (Y) was replaced with a constant, such that only the chrominance information remains.

This example and the next make only sense when viewed in colour. On a black/white printout of this slide, only the Y channel information will be present.

Y versus UV sensitivity example

original

blurred U and V

blurred Y channel

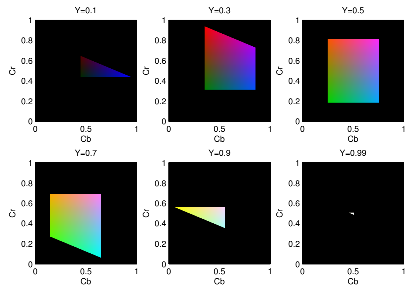
In the centre image, the chrominance channels have been severely low-pass filtered (Gaussian impulse response ). But the human eye perceives this distortion as far less severe than if the exact same filtering is applied to the luminance channel (right image).

YCrCb video colour coordinates

Since $-0.7 \le V \le 0.7$ and $-0.9 \le U \le 0.9$, a more convenient normalized encoding of chrominance is:

$$Cb = \frac{U}{2.0} + 0.5$$

$$Cr = \frac{V}{1.6} + 0.5$$



Modern image compression techniques operate on Y, Cr, Cb channels separately, using half the resolution of Y for storing Cr, Cb.

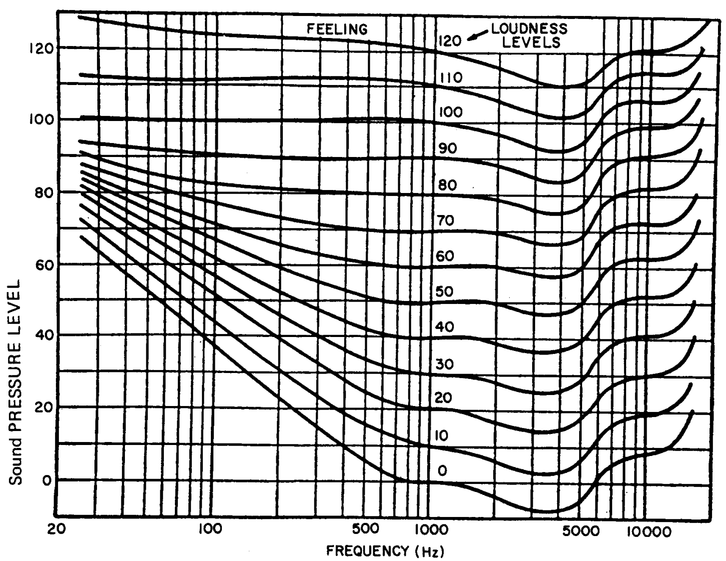
Some digital-television engineering terminology:

If each pixel is represented by its own Y, Cr and Cb byte, this is called a "4:4:4" format. In the compacter "4:2:2" format, a Cr and Cb value is transmitted only for every second pixel, reducing the horizontal chrominance resolution by a factor two. The "4:2:0" format transmits in alternating lines either Cr or Cb for every second pixel, thus halving the chrominance resolution both horizontally and vertically. The "4:1:1" format reduces the chrominance resolution horizontally by a quarter and "4:1:0" does so in both directions. [ITU-R BT.601]

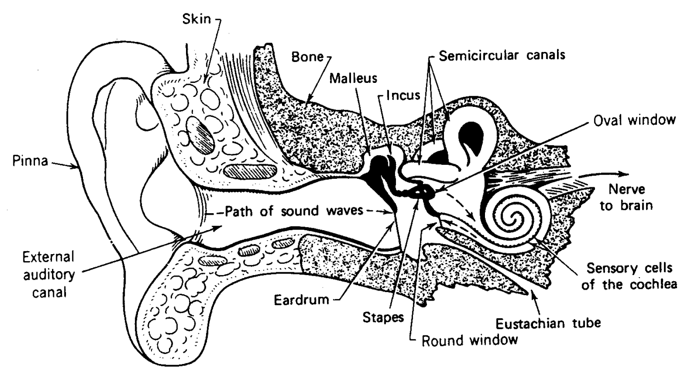
The human auditory system

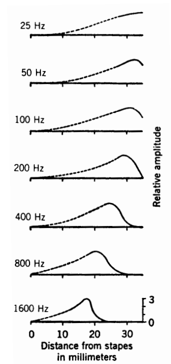
- frequency range 20–16000 Hz (babies: 20 kHz)
- sound pressure range 0–140 dB_{SPL} (about 10^{-5}–10^{2} pascal)
- mechanical filter bank (cochlea) splits input into frequency components, physiological equivalent of Fourier transform
- most signal processing happens in the frequency domain where phase information is lost
- some time-domain processing below 500 Hz and for directional hearing
- sensitivity and difference limit are frequency dependent

Equiloudness curves and the unit "phon"



Each curve represents a loudness level in "phon". At 1 kHz, the loudness unit phon is identical to dB_{SPL} and 0 phon is the sensation limit.





Sound waves cause vibration in the eardrum. The three smallest human bones in the middle ear (malleus, incus, stapes) provide an "impedance match" between air and liquid and conduct the sound via a second membrane, the oval window, to the cochlea. Its three chambers are rolled up into a spiral. The basilar membrane that separates the two main chambers decreases in stiffness along the spiral, such that the end near the stapes vibrates best at the highest frequencies, whereas for lower frequencies that amplitude peak moves to the far end.

Frequency discrimination and critical bands

A pair of pure tones (sine functions) cannot be distinguished as two separate frequencies if both are in the same frequency group ("critical band"). Their loudness adds up, and both are perceived with their average frequency.

The human ear has about 24 critical bands whose width grows non-linearly with the center frequency.

Each audible frequency can be expressed on the "Bark scale" with values in the range 0–24. A good closed-form approximation is

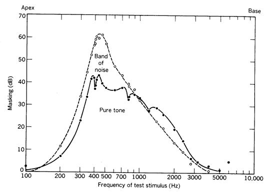
$$b \approx \frac{26.81}{1 + \frac{1960\ \text{Hz}}{f}} - 0.53$$

where f is the frequency and b the corresponding point on the Bark scale.

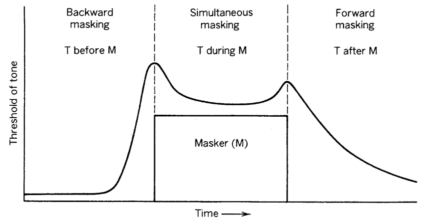
Two frequencies are in the same critical band if their distance is below 1 bark.

Masking

- Louder tones increase the sensation limit for nearby frequencies and suppress the perception of quieter tones.
- This increase is not symmetric. It extends about 3 barks to lower frequencies and 8 barks to higher ones.
- The sensation limit is increased less for pure tones of nearby frequencies, as these can still be perceived via their beat frequency. For the study of masking effects, pure tones therefore need to be distinguished from narrowband noise.
- Temporal masking: SL rises shortly before and after a masker.



Level of masking in decibels for test tones of various frequencies presented with a masker of either an 80-dB SPL 400 Hz tone or an 80-dB SPL narrow-band noise (90 Hz) with a center frequency of 410 Hz. From Egan and Hake (11).



Audio demo: loudness and masking

loudness.wav

Two sequences of tones with frequencies 40, 63, 100, 160, 250, 400, 630, 1000, 1600, 2500, 4000, 6300, 10000, and 16000 Hz.

- Sequence 1: tones have equal amplitude
- Sequence 2: tones have roughly equal perceived loudness
  Amplitude adjusted to IEC 60651 "A" weighting curve for soundlevel meters.

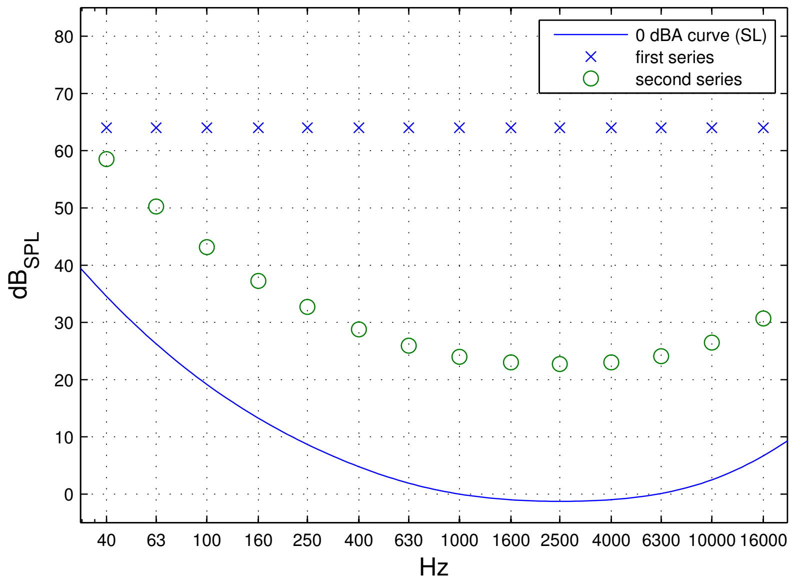
masking.wav

Twelve sequences, each with twelve probe-tone pulses and a 1200 Hz masking tone during pulses 5 to 8.

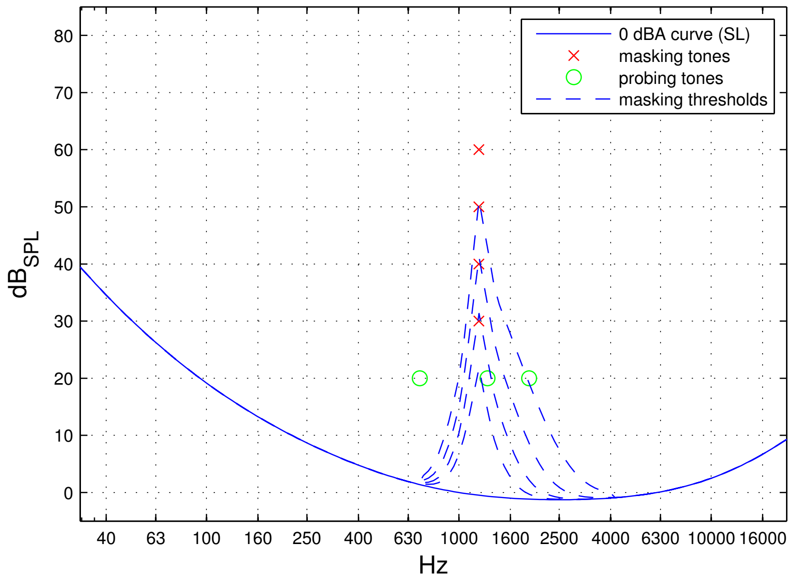
Probing tone frequency and relative masking tone amplitude:

	10 dB	20 dB	30 dB	40 dB
1300 Hz				
1900 Hz				
700 Hz				

Audio demo: loudness.wav

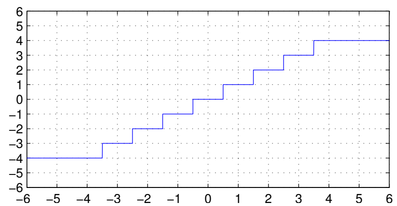


Audio demo: masking.wav

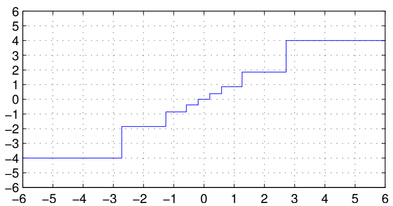


Quantization

Uniform/linear quantization:



Non-uniform quantization:



Quantization is the mapping from a continuous or large set of values (e.g., analog voltage, floating-point number) to a smaller set of (typically 2^8, 2^{12} or 2^{16}) values.

This introduces two types of error:

- the amplitude of *quantization noise* reaches up to half the maximum difference between neighbouring quantization levels
- *clipping* occurs where the input amplitude exceeds the value of the highest (or lowest) quantization level

Example of a linear quantizer (resolution R, peak value V):

$$y = \max\left\{-V, \min\left\{V, R\left\lfloor\frac{x}{R} + \frac{1}{2}\right\rfloor\right\}\right\}$$

Adding a noise signal that is uniformly distributed on $[0, 1]$ instead of adding $\frac{1}{2}$ helps to spread the frequency spectrum of the quantization noise more evenly. This is known as *dithering*.

Variant with even number of output values (no zero):

$$y = \max\left\{-V, \min\left\{V, R\left(\left\lfloor\frac{x}{R}\right\rfloor + \frac{1}{2}\right)\right\}\right\}$$

Improving the resolution by a factor of two (i.e., adding 1 bit) reduces the quantization noise by 6 dB.

Linearly quantized signals are easiest to process, but analog input levels need to be adjusted carefully to achieve a good tradeoff between the signal-to-quantization-noise ratio and the risk of clipping. Non-uniform quantization can reduce quantization noise where input values are not uniformly distributed and can approximate human perception limits.

Logarithmic quantization

Rounding the logarithm of the signal amplitude makes the quantization error scale-invariant and is used where the signal level is not very predictable. Two alternative schemes are widely used to make the logarithm function odd and linearize it across zero before quantization:

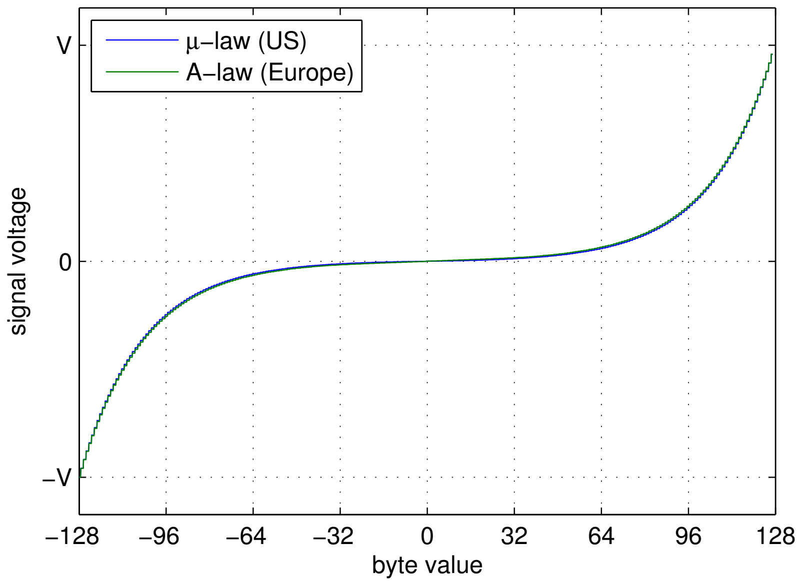
μ-law:

$$y = \frac{V \log(1 + \mu|x|/V)}{\log(1 + \mu)}\ \mathrm{sgn}(x) \quad \text{for } -V \le x \le V$$

A-law:

$$y = \begin{cases} \frac{A|x|}{1+\log A}\ \mathrm{sgn}(x) & \text{for } 0 \le |x| \le \frac{V}{A} \\ \frac{V\left(1+\log\frac{A|x|}{V}\right)}{1+\log A}\ \mathrm{sgn}(x) & \text{for } \frac{V}{A} \le |x| \le V \end{cases}$$

European digital telephone networks use A-law quantization ($A = 87.6$), North American ones use μ-law (μ=255), both with 8-bit resolution and 8 kHz sampling frequency (64 kbit/s). [ITU-T G.711]



Joint Photographic Experts Group – JPEG

Working group "ISO/TC97/SC2/WG8 (Coded representation of picture and audio information)" was set up in 1982 by the International Organization for Standardization.

Goals:

- continuous tone gray-scale and colour images
- recognizable images at 0.083 bit/pixel
- useful images at 0.25 bit/pixel
- excellent image quality at 0.75 bit/pixel
- indistinguishable images at 2.25 bit/pixel
- feasibility of 64 kbit/s (ISDN fax) compression with late 1980s hardware (16 MHz Intel 80386).
- workload equal for compression and decompression

The JPEG standard (ISO 10918) was finally published in 1994.

William B. Pennebaker, Joan L. Mitchell: JPEG still image compression standard. Van Nostrad Reinhold, New York, ISBN 0442012721, 1993.

Gregory K. Wallace: The JPEG Still Picture Compression Standard. Communications of the ACM 34(4)30–44, April 1991, `http://doi.acm.org/10.1145/103085.103089`

Summary of the baseline JPEG algorithm

The most widely used lossy method from the JPEG standard:

- Colour component transform: 8-bit RGB $\rightarrow$ 8-bit YCrCb
- Reduce resolution of Cr and Cb by a factor 2
- For the rest of the algorithm, process Y, Cr and Cb components independently (like separate gray-scale images)
  The above steps are obviously skipped where the input is a gray-scale image.
- Split each image component into 8×8 pixel blocks
  Partial blocks at the right/bottom margin may have to be padded by repeating the last column/row until a multiple of eight is reached. The decoder will remove these padding pixels.
- Apply the 8×8 forward DCT on each block
  On unsigned 8-bit input, the resulting DCT coefficients will be signed 11-bit integers.

- Quantization: divide each DCT coefficient with the corresponding value from an 8×8 table, then round to the nearest integer:
  The two standard quantization-matrix examples for luminance and chrominance are:

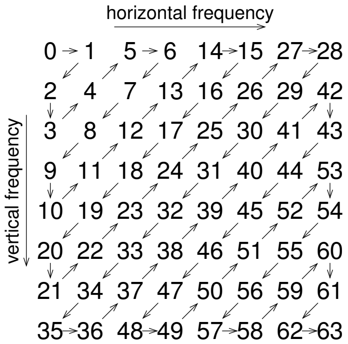
16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

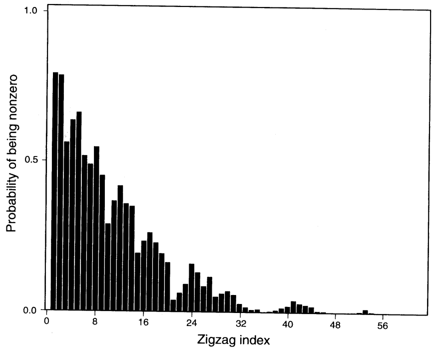
17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

- apply DPCM coding to quantized DC coefficients from DCT
- read remaining quantized values from DCT in zigzag pattern
- locate sequences of zero coefficients (run-length coding)
- apply Huffman coding on zero run-lengths and magnitude of AC values
- add standard header with compression parameters

`http://www.jpeg.org/`
Example implementation: `http://www.ijg.org/`

Storing DCT coefficients in zigzag order





After the 8×8 coefficients produced by the discrete cosine transform have been quantized, the values are processed in the above zigzag order by a run-length encoding step.

The idea is to group all higher-frequency coefficients together at the end of the sequence. As many image blocks contain little high-frequency information, the bottom-right corner of the quantized DCT matrix is often entirely zero. The zigzag scan helps the run-length coder to make best use of this observation.

Huffman coding in JPEG

s	value range
0	0
1	$-1, 1$
2	$-3, -2, 2, 3$
3	$-7 \ldots -4, 4 \ldots 7$
4	$-15 \ldots -8, 8 \ldots 15$
5	$-31 \ldots -16, 16 \ldots 31$
6	$-63 \ldots -32, 32 \ldots 63$
$\ldots$	$\ldots$
i	$-(2^i - 1) \ldots -2^{i-1}, 2^{i-1} \ldots 2^i - 1$

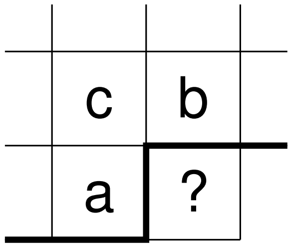
DCT coefficients have 11-bit resolution and would lead to huge Huffman tables (up to 2048 code words). JPEG therefore uses a Huffman table only to encode the magnitude category $s = \lceil \log_2(|v| + 1) \rceil$ of a DCT value v. A sign bit plus the $(s-1)$-bit binary value $|v| - 2^{s-1}$ are appended to each Huffman code word, to distinguish between the 2^s different values within magnitude category s.

When storing DCT coefficients in zigzag order, the symbols in the Huffman tree are actually tuples (r, s), where r is the number of zero coefficients preceding the coded value (run-length).

Lossless JPEG algorithm

In addition to the DCT-based lossy compression, JPEG also defines a lossless mode. It offers a selection of seven linear prediction mechanisms based on three previously coded neighbour pixels:

1 : $x = a$
2 : $x = b$
3 : $x = c$
4 : $x = a + b - c$
5 : $x = a + (b - c)/2$
6 : $x = b + (a - c)/2$
7 : $x = (a + b)/2$



Predictor 1 is used for the top row, predictor 2 for the left-most row. The predictor used for the rest of the image is chosen in a header. The difference between the predicted and actual value is fed into either a Huffman or arithmetic coder.

Advanced JPEG features

Beyond the baseline and lossless modes already discussed, JPEG provides these additional features:

- 8 or 12 bits per pixel input resolution for DCT modes
- 2–16 bits per pixel for lossless mode
- progressive mode permits the transmission of more-significant DCT bits or lower-frequency DCT coefficients first, such that a low-quality version of the image can be displayed early during a transmission
- the transmission order of colour components, lines, as well as DCT coefficients and their bits can be interleaved in many ways
- the hierarchical mode first transmits a low-resolution image, followed by a sequence of differential layers that code the difference to the next higher resolution

Not all of these features are widely used today. Several follow-on standards exist: JPEG XR uses a fully invertible DCT-like 4 × 4 block transform, JPEG 2000 uses a Cohen-Daubechies-Feauveau wavelet transform.

JPEG examples (baseline DCT)

1:5 (1.6 bit/pixel)

1:10 (0.8 bit/pixel)

JPEG examples (baseline DCT)

1:20 (0.4 bit/pixel)

1:50 (0.16 bit/pixel)

Better image quality at a compression ratio 1:50 can be achieved by applying DCT JPEG to a 50% scaled down version of the image (and then interpolate back to full resolution after decompression):

Moving Pictures Experts Group – MPEG

- MPEG-1: Coding of video and audio optimized for 1.5 Mbit/s (1× CD-ROM). ISO 11172 (1993).
- MPEG-2: Adds support for interlaced video scan, optimized for broadcast TV (2–8 Mbit/s) and HDTV, scalability options. Used by DVD and DVB. ISO 13818 (1995).
- MPEG-4: Adds advanced video codec (AVC) and advanced audio codec (AAC) for lower bitrate applications. ISO 14496 (2001).
- System layer multiplexes several audio and video streams, time stamp synchronization, buffer control.
- Standard defines decoder semantics.
- Asymmetric workload: Encoder needs significantly more computational power than decoder (for bit-rate adjustment, motion estimation, perceptual modeling, etc.)

`http://mpeg.chiariglione.org/`

MPEG video coding

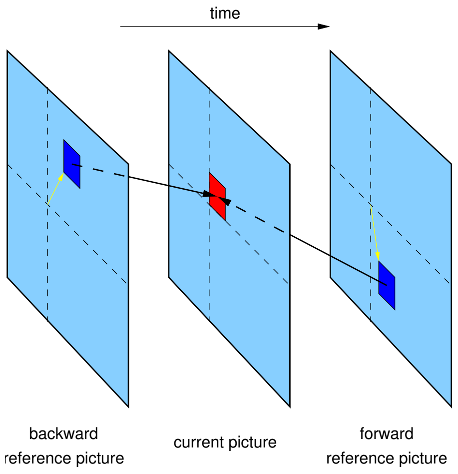
- Uses YCrCb colour transform, 8×8-pixel DCT, quantization, zigzag scan, run-length and Huffman encoding, similar to JPEG
- the zigzag scan pattern is adapted to handle interlaced fields
- Huffman coding with fixed code tables defined in the standard
  MPEG has no arithmetic coder option.
- adaptive quantization
- SNR and spatially scalable coding (enables separate transmission of a moderate-quality video signal and an enhancement signal to reduce noise or improve resolution)
- Predictive coding with motion compensation based on 16×16 macro blocks.

J. Mitchell, W. Pennebaker, Ch. Fogg, D. LeGall: MPEG video compression standard. ISBN 0412087715, 1997. (CL library: I.4.20)

B. Haskell et al.: Digital Video: Introduction to MPEG-2. Kluwer Academic, 1997. (CL library: I.4.27)

John Watkinson: The MPEG Handbook. Focal Press, 2001. (CL library: I.4.31)

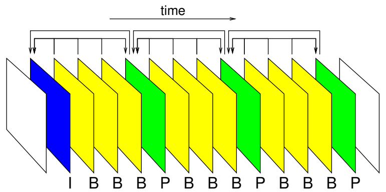
MPEG motion compensation



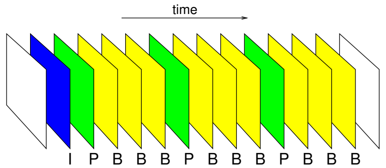
Each MPEG image is split into 16×16-pixel large *macroblocks*. The predictor forms a linear combination of the content of one or two other blocks of the same size in a preceding (and following) reference image. The relative positions of these reference blocks are encoded along with the differences.

MPEG reordering of reference images

Display order of frames:



Coding order:



MPEG distinguishes between I-frames that encode an image independent of any others, P-frames that encode differences to a previous P- or I-frame, and B-frames that interpolate between the two neighbouring B- and/or I-frames. A frame has to be transmitted before the first B-frame that makes a forward reference to it. This requires the coding order to differ from the display order.

MPEG audio coding

Three different algorithms are specified, each increasing the processing power required in the decoder.

Supported sampling frequencies: 32, 44.1 or 48 kHz.

Layer I

- Waveforms are split into segments of 384 samples each (8 ms at 48 kHz).
- Each segment is passed through an orthogonal filter bank that splits the signal into 32 subbands, each 750 Hz wide (for 48 kHz).
  This approximates the critical bands of human hearing.
- Each subband is then sampled at 1.5 kHz (for 48 kHz).
  12 samples per window → again 384 samples for all 32 bands
- This is followed by scaling, bit allocation and uniform quantization.
  Each subband gets a 6-bit scale factor (2 dB resolution, 120 dB range, like floating-point coding). Layer I uses a fixed bitrate without buffering. A bit allocation step uses the psychoacoustic model to distribute all available resolution bits across the 32 bands (0–15 bits for each sample). With a sufficient bit rate, the quantization noise will remain below the sensation limit.
- Encoded frame contains bit allocation, scale factors and sub-band samples.

Layer II

Uses better encoding of scale factors and bit allocation information.

Unless there is significant change, only one out of three scale factors is transmitted. Explicit zero code leads to odd numbers of quantization levels and wastes one codeword. Layer II combines several quantized values into a *granule* that is encoded via a lookup table (e.g., 3 × 5 levels: 125 values require 7 instead of 9 bits). Layer II is used in Digital Audio Broadcasting (DAB).

Layer III

- Modified DCT step decomposes subbands further into 18 or 6 frequencies
- dynamic switching between MDCT with 36-samples (28 ms, 576 freq.) and 12-samples (8 ms, 192 freq.)
  enables control of pre-echos before sharp percussive sounds (Heisenberg)
- non-uniform quantization
- Huffman entropy coding
- buffer with short-term variable bitrate
- joint stereo processing

MPEG audio layer III is the widely used "MP3" music compression format.

Psychoacoustic models

MPEG audio encoders use a psychoacoustic model to estimate the spectral and temporal masking that the human ear will apply. The subband quantization levels are selected such that the quantization noise remains below the masking threshold in each subband.

The masking model is not standardized and each encoder developer can chose a different one. The steps typically involved are:

- Fourier transform for spectral analysis
- Group the resulting frequencies into "critical bands" within which masking effects will not vary significantly
- Distinguish tonal and non-tonal (noise-like) components
- Apply masking function
- Calculate threshold per subband
- Calculate signal-to-mask ratio (SMR) for each subband

Masking is not linear and can be estimated accurately only if the actual sound pressure levels reaching the ear are known. Encoder operators usually cannot know the sound pressure level selected by the decoder user. Therefore the model must use worst-case SMRs.

Outlook

Further topics that we have not covered in this brief introductory tour through DSP, but for the understanding of which you should now have a good theoretical foundation:

- multirate systems
- effects of rounding errors
- adaptive filters
- DSP hardware architectures
- modulation and symbol detection techniques
- sound effects

If you find any typo or mistake in these lecture notes, please email `Markus.Kuhn@cl.cam.ac.uk`.

Some final thoughts about redundancy . . .

```
Aoccdrnig to rsceearh at Cmabrigde Uinervtisy, it deosn't
mttaer in waht oredr the ltteers in a wrod are, the olny
iprmoetnt tihng is taht the frist and lsat ltteer be at
the rghit pclae. The rset can be a total mses and you can
sitll raed it wouthit porbelm. Tihs is bcuseae the huamn
mnid deos not raed ervey lteter by istlef, but the wrod as
a wlohe.
```

. . . and perception

Count how many Fs there are in this text:

```
FINISHED FILES ARE THE RE-
SULT OF YEARS OF SCIENTIF-
IC STUDY COMBINED WITH THE
EXPERIENCE OF YEARS
```