Digital Signal Processing

Markus Kuhn

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

`http://www.cl.cam.ac.uk/Teaching/2005/DSP/`

Michaelmas 2005 – Part II

Signals

$\longrightarrow$ flow of information

$\longrightarrow$ measured quantity that varies with time (or position)

$\longrightarrow$ electrical signal received from a transducer
(microphone, thermometer, accelerometer, antenna, etc.)

$\longrightarrow$ 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 can only deal easily with time-dependent signals, therefore 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

$\longrightarrow$ amplify or filter out embedded information

$\longrightarrow$ detect patterns

$\longrightarrow$ prepare the signal to survive a transmission channel

$\longrightarrow$ prevent interference with other signals sharing a medium

$\longrightarrow$ undo distortions contributed by a transmission channel

$\longrightarrow$ compensate for sensor deficiencies

$\longrightarrow$ find information encoded in a different domain

To do so, we also need

$\longrightarrow$ methods to measure, characterise, model and simulate transmission channels

$\longrightarrow$ mathematical tools that split common channels and transformations into easily manipulated building blocks

Analog electronics

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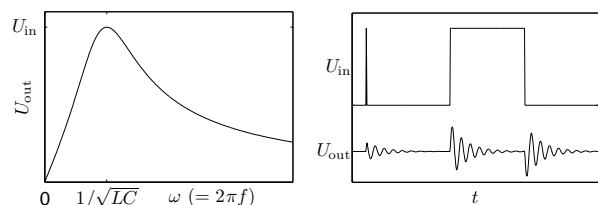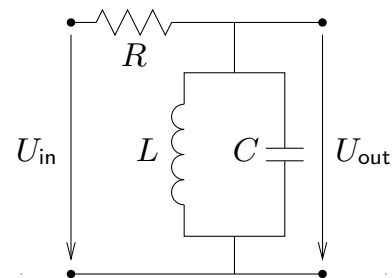
$$\frac{U_{\text{in}} - U_{\text{out}}}{R} = \frac{1}{L} \int_{-\infty}^{t} U_{\text{out}} \mathrm{d}\tau + C \frac{\mathrm{d}U_{\text{out}}}{\mathrm{d}t}$$

Digital signal processing

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

Advantages:

$\longrightarrow$ noise is easy to control after initial quantization

$\longrightarrow$ highly linear (within limited dynamic range)

$\longrightarrow$ complex algorithms fit into a single chip

$\longrightarrow$ flexibility, parameters can easily be varied in software

$\longrightarrow$ digital processing is insensitive to component tolerances, aging, environmental conditions, electromagnetic interference

But:

$\longrightarrow$ discrete time processing artifacts (aliasing)

$\longrightarrow$ can require significantly more power (battery, cooling)

$\longrightarrow$ digital clock and switching cause interference

Typical DSP applications

$\longrightarrow$ communication systems

modulation/demodulation, channel equalization, echo cancellation

$\longrightarrow$ consumer electronics

perceptual coding of audio and video on DVDs, speech synthesis, speech recognition

$\longrightarrow$ music

synthetic instruments, audio effects, noise reduction

$\longrightarrow$ medical diagnostics

magnetic-resonance and ultrasonic imaging, computer tomography, ECG, EEG, MEG, AED, audiology

$\longrightarrow$ geophysics

seismology, oil exploration

$\longrightarrow$ astronomy

VLBI, speckle interferometry

$\longrightarrow$ experimental physics

sensor-data evaluation

$\longrightarrow$ aviation

radar, radio navigation

$\longrightarrow$ security

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

$\longrightarrow$ engineering

control systems, feature extraction for pattern recognition

Syllabus

Signals and systems. Discrete sequences and systems, their types and properties. Linear time-invariant systems, convolution. Harmonic phasors are the eigen functions of linear time-invariant systems. Review of complex arithmetic. Some examples from electronics, optics and acoustics.

MATLAB. Use of MATLAB on PWF machines to perform numerical experiments and visualise the results in homework exercises.

Fourier transform. Harmonic phasors as orthogonal base functions. Forms of the Fourier transform, convolution theorem, Dirac's delta function, impulse combs in the time and frequency domain.

Discrete sequences and spectra. Periodic sampling of continuous signals, periodic signals, aliasing, sampling and reconstruction of low-pass and band-pass signals, spectral inversion.

Discrete Fourier transform. Continuous *versus* discrete Fourier transform, symmetry, linearity, review of the FFT, real-valued FFT.

Spectral estimation. Leakage and scalloping phenomena, windowing, zero padding.

Finite and infinite impulse-response filters. Properties of filters, implementation forms, window-based FIR design, use of frequency-inversion to obtain high-pass filters, use of modulation to obtain band-pass filters, FFT-based convolution, polynomial representation, z-transform, zeros and poles, use of analog IIR design techniques (Butterworth, Chebyshev I/II, elliptic filters).

Random sequences and noise. Random variables, stationary processes, autocorrelation, crosscorrelation, deterministic crosscorrelation sequences, filtered random sequences, white noise, exponential averaging.

Correlation coding. Random vectors, dependence versus correlation, covariance, decorrelation, matrix diagonalisation, eigen decomposition, Karhunen-Loève transform, principal/independent component analysis. Relation to orthogonal transform coding using fixed basis vectors, such as DCT.

Lossy versus lossless compression. What information is discarded by human senses and can be eliminated by encoders? Perceptual scales, masking, spatial resolution, colour coordinates, some demonstration experiments.

Quantization, image and audio coding standards. A/μ-law coding, delta coding, JPEG photographic still-image compression, motion compensation, MPEG video encoding, MPEG audio encoding.

Note: The last three lectures on audio-visual coding were last year part of the Information Theory and Coding course.

Objectives

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

$\longrightarrow$ apply basic properties of time-invariant linear systems

$\longrightarrow$ understand sampling, aliasing, convolution, filtering, the pitfalls of spectral estimation

$\longrightarrow$ explain the above in time and frequency domain representations

$\longrightarrow$ use filter-design software

$\longrightarrow$ visualise and discuss digital filters in the z-domain

$\longrightarrow$ use the FFT for convolution, deconvolution, filtering

$\longrightarrow$ implement, apply and evaluate simple DSP applications in MATLAB

$\longrightarrow$ apply transforms that reduce correlation between several signal sources

$\longrightarrow$ understand and explain limits in human perception that are exploited by lossy compression techniques

$\longrightarrow$ provide a good overview of the principles and characteristics of several widely-used compression techniques and standards for audio-visual signals

Textbooks

$\longrightarrow$ R.G. Lyons: *Understanding digital signal processing.* Prentice-Hall, 2004. (£45)

$\longrightarrow$ A.V. Oppenheim, R.W. Schafer: *Discrete-time signal processing.* 2nd ed., Prentice-Hall, 1999. (£47)

$\longrightarrow$ J. Stein: *Digital signal processing – a computer science perspective.* Wiley, 2000. (£74)

$\longrightarrow$ S.W. Smith: *Digital signal processing – a practical guide for engineers and scientists.* Newness, 2003. (£40)

$\longrightarrow$ K. Steiglitz: *A digital signal processing primer – with applications to digital audio and computer music.* Addison-Wesley, 1996. (£40)

$\longrightarrow$ Sanjit K. Mitra: *Digital signal processing – a computer-based approach.* McGraw-Hill, 2002. (£38)

Sequences and systems

A *discrete sequence* $\{x_n\}$ 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.
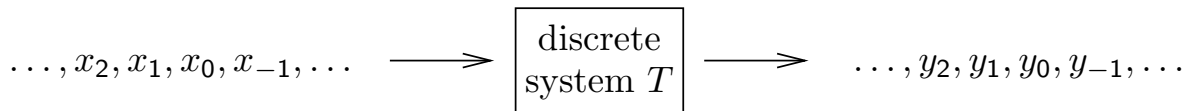
The notation is not well standardized. Some authors write $x[n]$ instead of x_n, others $x(n)$.

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

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

we call t_{s} the *sampling period* and $f_{\mathsf{s}} = 1/t_{\mathsf{s}}$ the *sampling frequency*. A *discrete system* T receives as input a sequence $\{x_n\}$ and transforms it into an output sequence $\{y_n\} = T\{x_n\}$:

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

Properties of sequences

A sequence $\{x_n\}$ is

$$\text{absolutely summable} \iff \sum_{n=-\infty}^{\infty} |x_n| < \infty$$

$$\text{square summable} \iff \sum_{n=-\infty}^{\infty} |x_n|^2 < \infty$$

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

A square-summable sequence is also called an *energy signal*, and

$$\sum_{n=-\infty}^{\infty} |x_n|^2$$

its energy. This terminology reflects that if U is a voltage supplied to a load resistor R, then $P = UI = U^2/R$ is the power consumed.

So even where we drop physical units (e.g., volts) for simplicity in sequence calculations, it is still customary to refer to the squared values of a sequence as *power* and to its sum or integral over time as *energy*.

A non-square-summable sequence is a *power signal* if its average power

$$\lim_{k \to \infty} \frac{1}{1+2k} \sum_{n=-k}^{k} |x_n|^2$$

exists.

Special sequences

Unit-step sequence:

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

Impulse sequence:

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

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 $\{x'_n\}$

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

Examples:

The *accumulator system*

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

is a causal, linear, time-invariant system with memory, as are the *backward difference system*

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

the *M-point moving average system*

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

and the *exponential averaging system*

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

Examples for time-invariant non-linear memory-less systems are

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

examples for linear but not time-invariant systems are

$$
\begin{aligned}
y_n &= \begin{cases} x_n, & n \geq 0 \\ 0, & n < 0 \end{cases} = x_n \cdot u_n \\
y_n &= x_{\lfloor n/4 \rfloor} \\
y_n &= x_n \cdot \Re(e^{\omega j n})
\end{aligned}
$$

and examples for linear time-invariant non-causal system are

$$
\begin{aligned}
y_n &= \frac{1}{2}(x_{n-1} + x_{n+1}) \\
y_n &= \sum_{k=-9}^{9} x_{n+k} \cdot \frac{\sin(\pi k \omega)}{\pi k \omega} \cdot [0.5 + 0.5 \cdot \cos(\pi k / 10)]
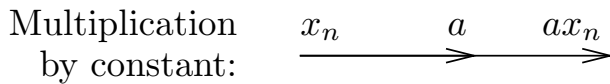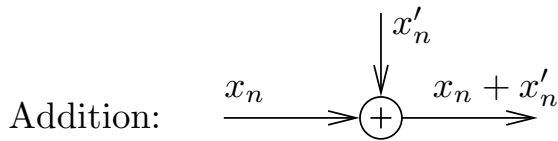\end{aligned}
$$

Constant-coefficient difference equations

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

$$y_n = b_0 \cdot x_n - \sum_{k=1}^{N} a_k \cdot y_{n-k}$$
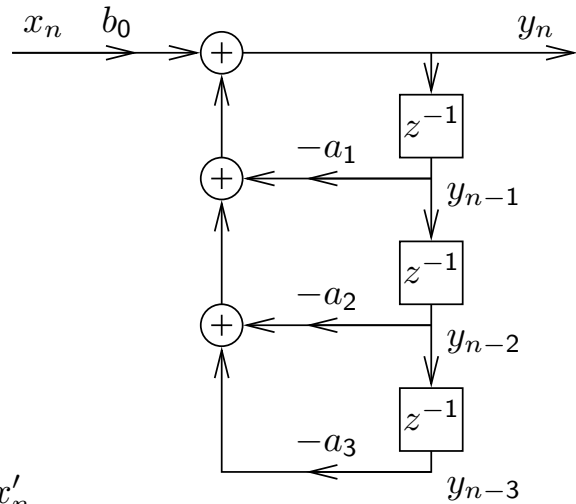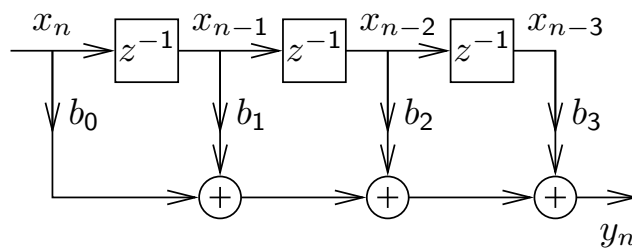
Block diagram representation of sequence operations:

Addition:

$$x_n \xrightarrow{\qquad} \underset{x_n'}{\bigoplus} \xrightarrow{x_n + x_n'}$$

Multiplication by constant:

$$x_n \xrightarrow{\quad a \quad} a x_n$$

Delay:

$$x_n \xrightarrow{\quad} \boxed{z^{-1}} \xrightarrow{x_{n-1}}$$
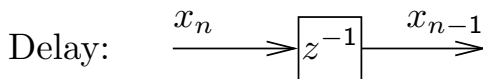
The a_k and b_m are constant coefficients.

or

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

or the combination of both:

$$\sum_{k=0}^{N} a_k \cdot y_{n-k} = \sum_{m=0}^{M} b_m \cdot x_{n-m}$$
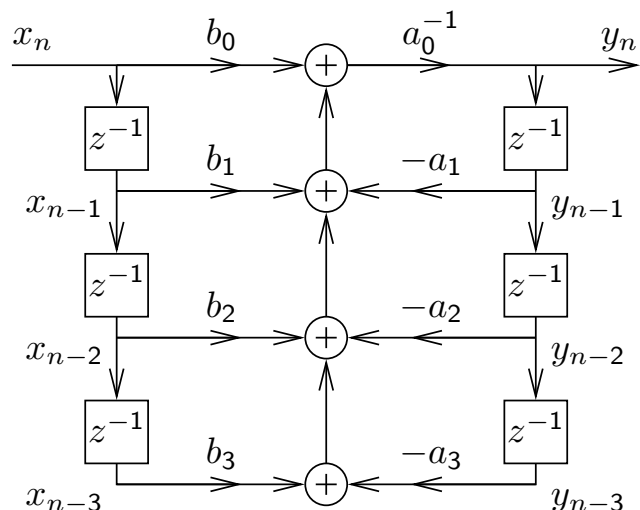
The MATLAB function `filter` is an efficient implementation of the last variant.

Convolution

All linear time-invariant (LTI) systems can be represented in the form

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

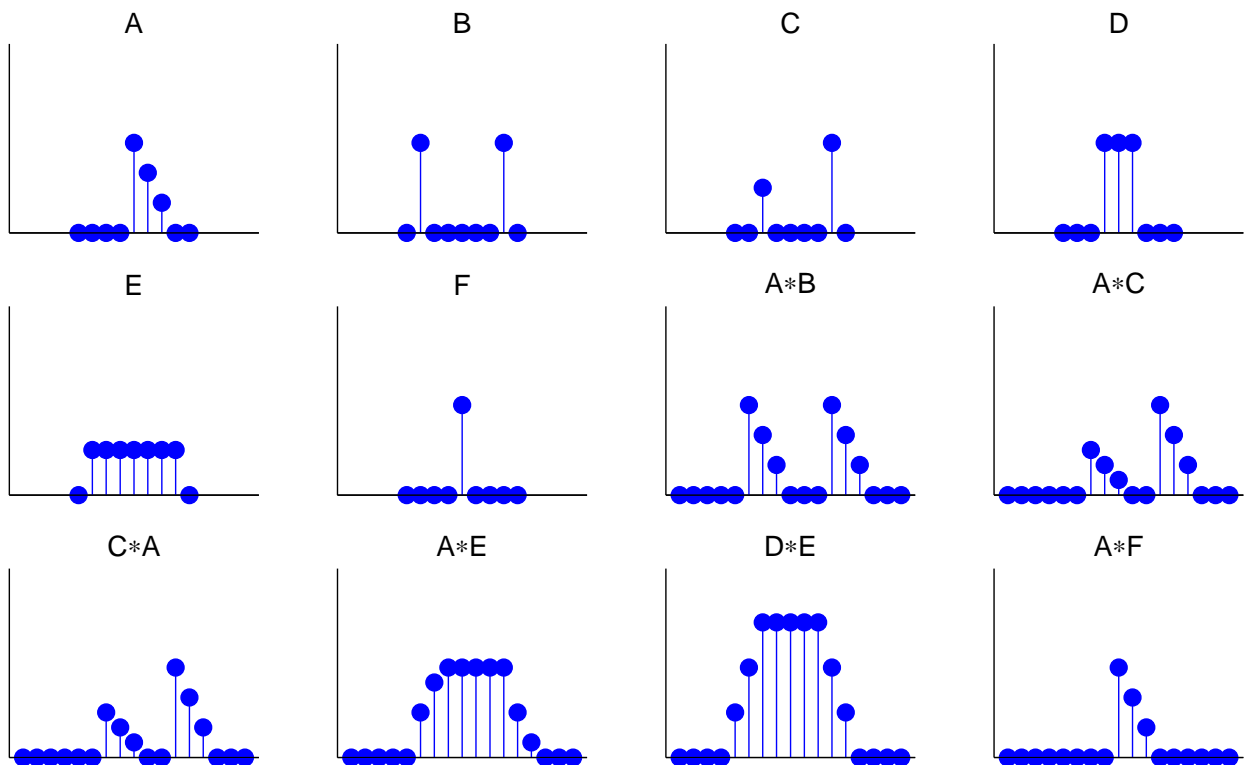where $\{a_k\}$ is a suitably chosen sequence of coefficients.

This operation over sequences is called *convolution* and 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:

$\longrightarrow$ Convolution is associative

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

$\longrightarrow$ Convolution is commutative

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

$\longrightarrow$ Convolution is linear

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

$\longrightarrow$ The impulse sequence (slide 13) is neutral under convolution

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

$\longrightarrow$ Sequence shifting is equivalent to convolving with a shifted impulse

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

Can all LTI systems be represented by 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:

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

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

Exercise 1 What type of discrete system (linear/non-linear, time-invariant/ non-time-invariant, causal/non-causal, causal, memory-less, etc.) is:

(a) $y_n = |x_n|$

(b) $y_n = -x_{n-1} + 2x_n - x_{n+1}$

(c) $y_n = \prod_{i=0}^{8} x_{n-i}$

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

(e) $y_n = \dfrac{3x_{n-1} + x_{n-2}}{x_{n-3}}$

(f) $y_n = x_n \cdot e^{n/14}$

(g) $y_n = x_n \cdot u_n$

(h) $y_n = \sum_{i=-\infty}^{\infty} x_i \cdot \delta_{i-n+2}$

Exercise 2

Prove that convolution is (a) commutative and (b) associative.

Exercise 3 A *finite-length sequence* is non-zero only at a finite number of positions. If m and n are the first and last non-zero positions, respectively, then we call $n - m + 1$ the *length* of that sequence. What maximum length can the result of convolving two sequences of length k and l have?

Exercise 4 The length-3 sequence $a_0 = -3$, $a_1 = 2$, $a_2 = 1$ is convolved with a second sequence $\{b_n\}$ of length 5.

(a) Write down this linear operation as a matrix multiplication involving a matrix A, a vector $\vec{b} \in \mathbb{R}^5$, and a result vector $\vec{c}$.

(b) Use MATLAB to multiply your matrix by the vector $\vec{b} = (1, 0, 0, 2, 2)$ and compare the result with that of using the conv function.

(c) Use the MATLAB facilities for solving systems of linear equations to undo the above convolution step.

Exercise 5 (a) Find a pair of sequences $\{a_n\}$ and $\{b_n\}$, where either contains at least three different values and where the convolution $\{a_n\} * \{b_n\}$ results in an all-zero sequence.

(b) Does every LTI system T have an inverse LTI system T^{-1} such that $\{x_n\} = T^{-1}T\{x_n\}$ for all sequences $\{x_n\}$? Why?

Direct form I and II implementations
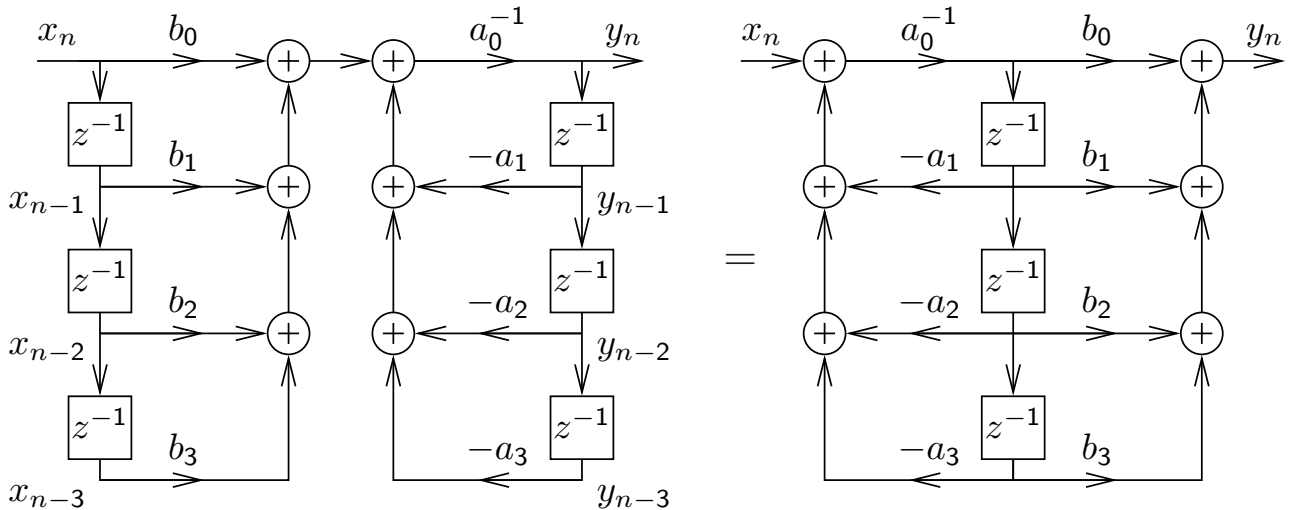
The block diagram representation of the constant-coefficient difference equation on slide 18 is called the *direct form I implementation*. The number of delay elements can be halved by using the commutativity of convolution to swap the two feedback loops, leading to the *direct form II implementation* of the same LTI system.

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):
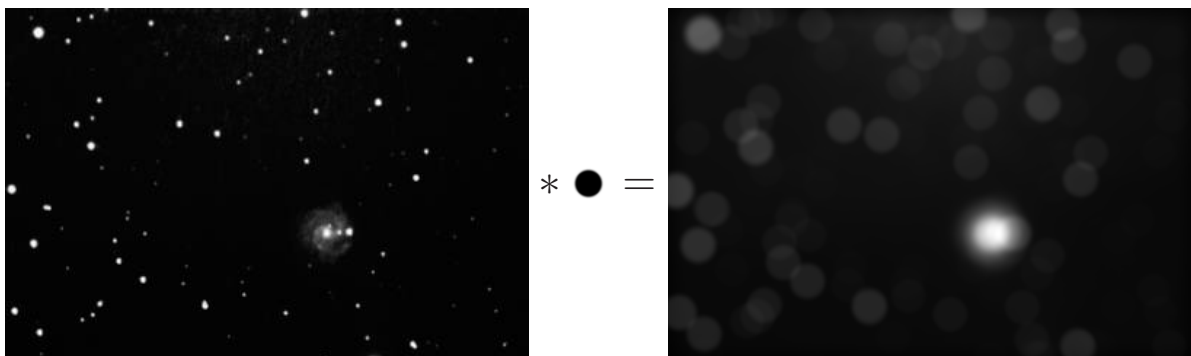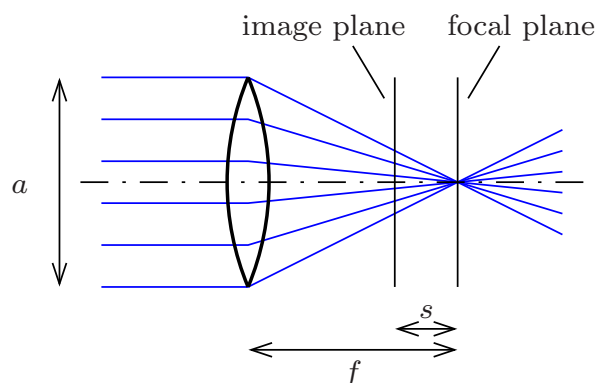

Point-spread function h (disk, $r = \frac{as}{2f}$):

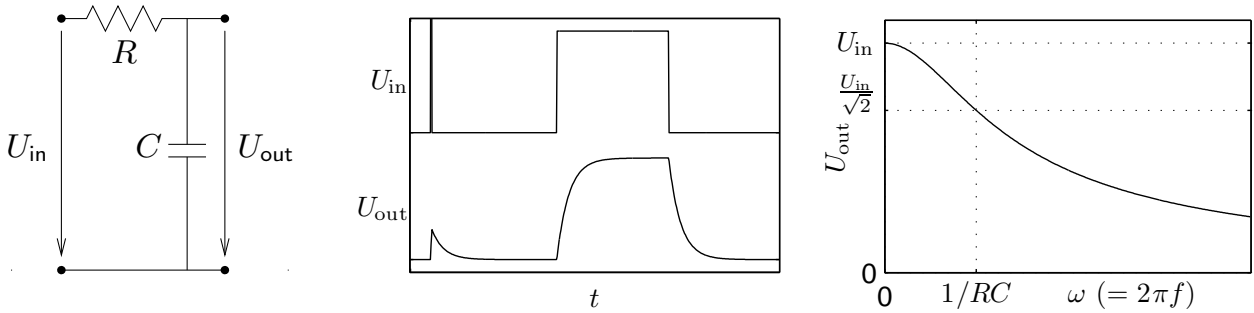$$h(x, y) = \begin{cases} \frac{1}{r^2 \pi}, & x^2 + y^2 \leq r^2 \\ 0, & x^2 + y^2 > r^2 \end{cases}$$

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 this 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:

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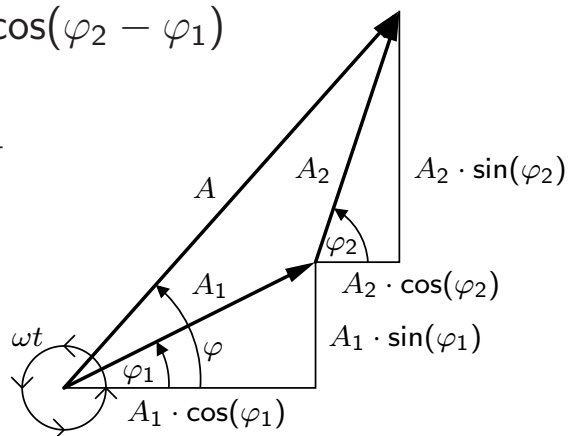
with

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

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

Sine waves of any phase can be from 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 $\{y_n\}$ as decomposition into a sum of impulses.)

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

$\implies$ **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:

$$\int_{-\infty}^{\infty} \sin(\omega_1 t + \varphi_1) \cdot \sin(\omega_2 t + \varphi_2)\, \mathrm{d}t = 0$$

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

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

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 $\sqrt{-1} = j$ story).

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

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

$$\sin(\omega t) = \frac{1}{2j}\left(e^{j\omega t} - e^{-j\omega t}\right)$$

or

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

or

$$\cos(\omega n + \varphi) = \Re\{e^{j\omega n + \varphi}\} = \Re\{(e^{j\omega})^n \cdot e^{j\varphi}\}$$
$$\sin(\omega n + \varphi) = \Im\{e^{j\omega n + \varphi}\} = \Im\{(e^{j\omega})^n \cdot e^{j\varphi}\}$$

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 $e^{j\omega}$.

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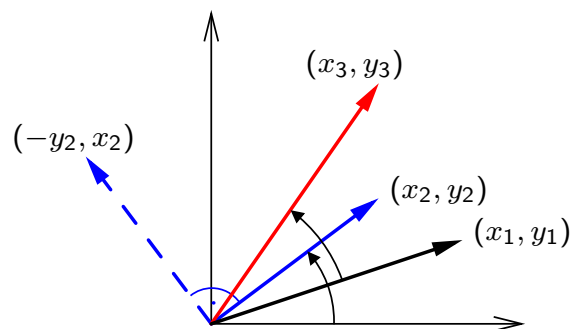
Rotation of a 2D vector in (x, y)-form is notationally slightly messy, but fortunately $j^2 = -1$ does exactly what is required here:

$$\begin{pmatrix} x_3 \\ y_3 \end{pmatrix} = \begin{pmatrix} x_2 & -y_2 \\ y_2 & x_2 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$

$$= \begin{pmatrix} x_1 x_2 - y_1 y_2 \\ x_1 y_2 + x_2 y_1 \end{pmatrix}$$

$$z_1 = x_1 + jy_1, \quad z_2 = x_2 + jy_2$$

$$z_1 \cdot z_2 = x_1 x_2 - y_1 y_2 + j(x_1 y_2 + x_2 y_1)$$

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

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

(where $j^2 = -1$) 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

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

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 30, where the argument of H is the base, we would write $H(e^{j\omega})$.

33

Recall: Fourier transform

The Fourier integral transform and its inverse are defined as

$$\mathcal{F}\{g(t)\}(\omega) \;=\; G(\omega) \;=\; \alpha \int_{-\infty}^{\infty} g(t) \cdot e^{\mp j\omega t}\, dt$$

$$\mathcal{F}^{-1}\{G(\omega)\}(t) \;=\; g(t) \;=\; \beta \int_{-\infty}^{\infty} G(\omega) \cdot e^{\pm j\omega t}\, d\omega$$

where α and β are constants chosen such that $\alpha\beta = 1/(2\pi)$.

Many equivalent forms of the Fourier transform are used in the literature, and there is no strong consensus on whether the forward transform uses $e^{-j\omega t}$ and the backwards transform $e^{j\omega t}$, or vice versa. Some authors set $\alpha = 1$ and $\beta = 1/(2\pi)$, to keep the convolution theorem free of a constant prefactor; others use $\alpha = \beta = 1/\sqrt{2\pi}$, in the interest of symmetry.

The substitution $\omega = 2\pi f$ leads to a form without prefactors:

$$\mathcal{F}\{h(t)\}(f) \;=\; H(f) \;=\; \int_{-\infty}^{\infty} h(t) \cdot e^{\mp 2\pi j f t}\, dt$$

$$\mathcal{F}^{-1}\{H(f)\}(t) \;=\; h(t) \;=\; \int_{-\infty}^{\infty} H(f) \cdot e^{\pm 2\pi j f t}\, df$$

34

Another notation is in the continuous case

$$\mathcal{F}\{h(t)\}(\omega) \;=\; H(\mathrm{e}^{\mathrm{j}\omega}) \;=\; \int_{-\infty}^{\infty} h(t) \;\cdot \mathrm{e}^{-\mathrm{j}\omega t}\,\mathrm{d}t$$

$$\mathcal{F}^{-1}\{H(\mathrm{e}^{\mathrm{j}\omega})\}(t) \;=\; h(t) \;=\; \frac{1}{2\pi}\int_{-\infty}^{\infty} H(\mathrm{e}^{\mathrm{j}\omega}) \cdot \mathrm{e}^{\mathrm{j}\omega t}\,\mathrm{d}\omega$$

and in the discrete-sequence case

$$\mathcal{F}\{h_n\}(\omega) \;=\; H(\mathrm{e}^{\mathrm{j}\omega}) \;=\; \sum_{n=-\infty}^{\infty} h_n \cdot \mathrm{e}^{-\mathrm{j}\omega n}$$

$$\mathcal{F}^{-1}\{H(\mathrm{e}^{\mathrm{j}\omega})\}(t) \;=\; h_n \;=\; \frac{1}{2\pi}\int_{-\pi}^{\pi} H(\mathrm{e}^{\mathrm{j}\omega}) \cdot \mathrm{e}^{\mathrm{j}\omega n}\,\mathrm{d}\omega$$

which treats the Fourier transform as a special case of the z-transform (to be introduced shortly).

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:

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

Frequency shifting:

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

Parseval's theorem (total power):

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

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:

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

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

Dirac's delta function

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

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

$$\int_{-\infty}^{\infty} \delta(x)\,\mathrm{d}x = 1$$
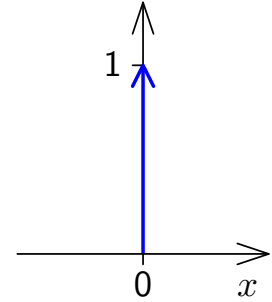
and can be thought of as the limit of function sequences such as

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

or

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

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 Dirac's delta function:

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

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

$$\frac{1}{2\pi}\int_{-\infty}^{\infty} e^{\pm \mathrm{j}\omega t}\mathrm{d}\omega = \delta(t)$$
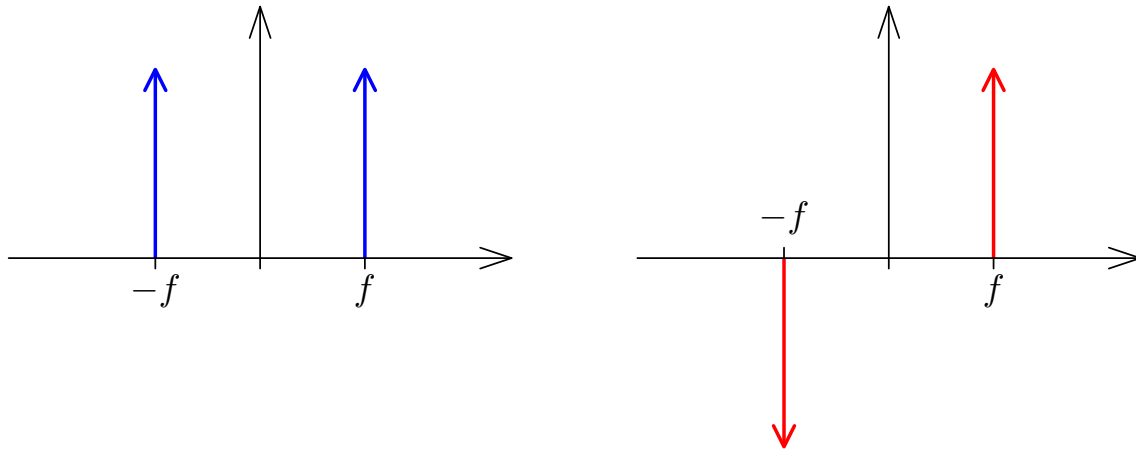
Fourier transform:

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

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

http://mathworld.wolfram.com/DeltaFunction.html

Sine and cosine in the frequency domain

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

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

Fourier transform symmetries

We call a function $x(t)$

$$\begin{aligned} \text{odd:} \quad x(-t) &= -x(t) \\ \text{even:} \quad x(-t) &= x(t) \end{aligned}$$

and $\cdot^*$ is the complex conjugate, such that $(a+jb)^* = (a-jb)$.
Then

$$\begin{aligned}
x(t) \text{ is real} &\Leftrightarrow X(-f) = [X(f)]^* \\
x(t) \text{ is imaginary} &\Leftrightarrow X(-f) = -[X(f)]^* \\
x(t) \text{ is even} &\Leftrightarrow X(f) \text{ is even} \\
x(t) \text{ is odd} &\Leftrightarrow X(f) \text{ is odd} \\
x(t) \text{ is real and even} &\Leftrightarrow X(f) \text{ is real and even} \\
x(t) \text{ is real and odd} &\Leftrightarrow X(f) \text{ is imaginary and odd} \\
x(t) \text{ is imaginary and even} &\Leftrightarrow X(f) \text{ is imaginary and even} \\
x(t) \text{ is imaginary and odd} &\Leftrightarrow X(f) \text{ is real and odd}
\end{aligned}$$

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

$$x_n = x(t_s \cdot n) = x(n/f_s)$$

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

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

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.

43

The Fourier transform of a Dirac comb

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

is another Dirac comb

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

$$\int_{-\infty}^{\infty} \sum_{n=-\infty}^{\infty} \delta(t - t_s n)\, e^{2\pi \mathrm{j} f t} \mathrm{d}t = \frac{1}{t_s} \sum_{n=-\infty}^{\infty} \delta\left(f - \frac{n}{t_s} \right).$$
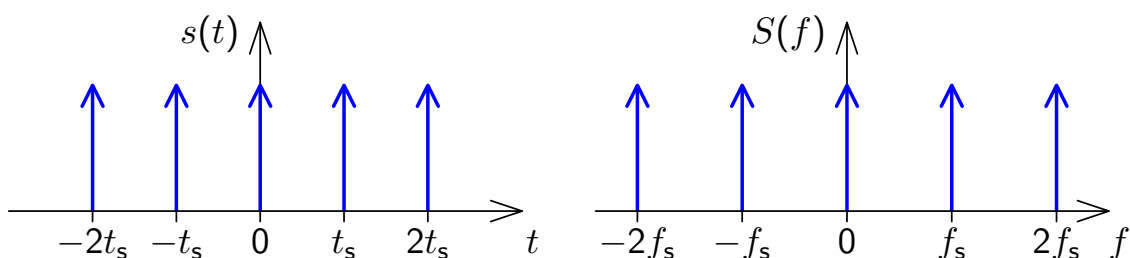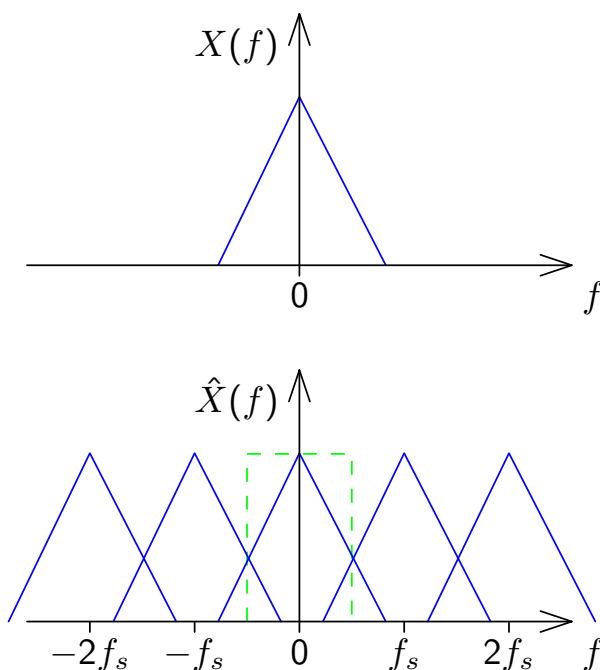

44

Sampling, aliasing and Nyquist limit
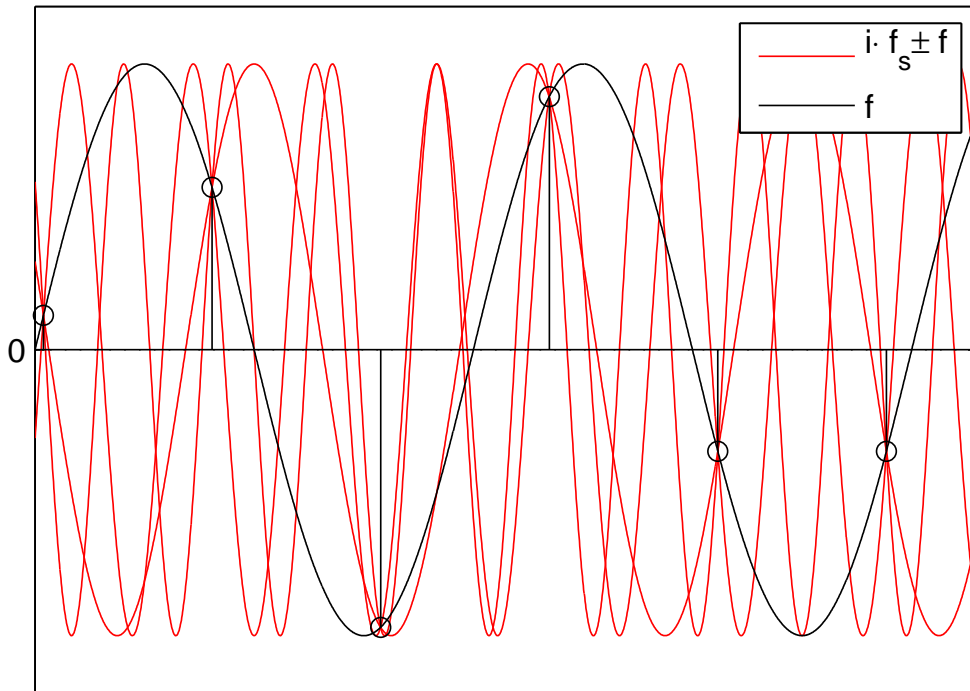
A wave $\cos(2\pi t f)$ sampled at frequency f_s cannot be distinguished from $\cos(2\pi t(k f_s \pm f))$ for any $k \in \mathbb{Z}$. Therefore, ensure $|f| < f_s/2$.
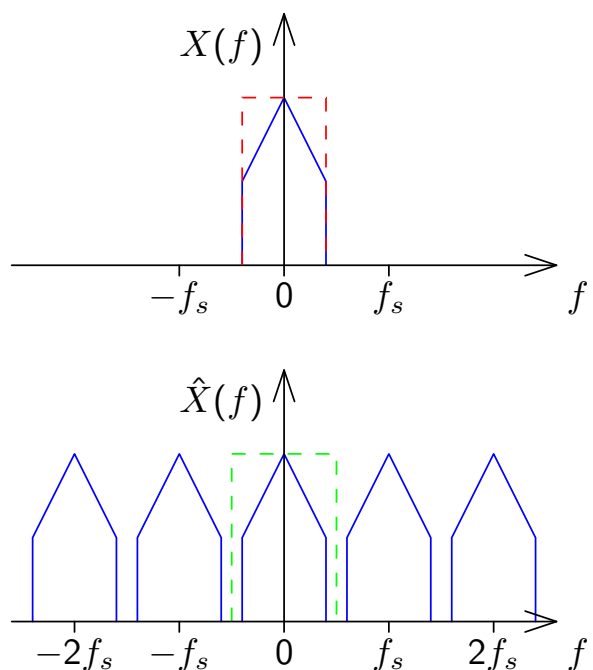
Frequency-domain view of sampling

Without anti-aliasing filter

With anti-aliasing filter

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

$$H(f) = \begin{cases} 1 & \text{if } -\frac{f_s}{2} < f \leq \frac{f_s}{2} \\ 0 & \text{otherwise} \end{cases}.$$

This leads, after an inverse Fourier transform, to the impulse response
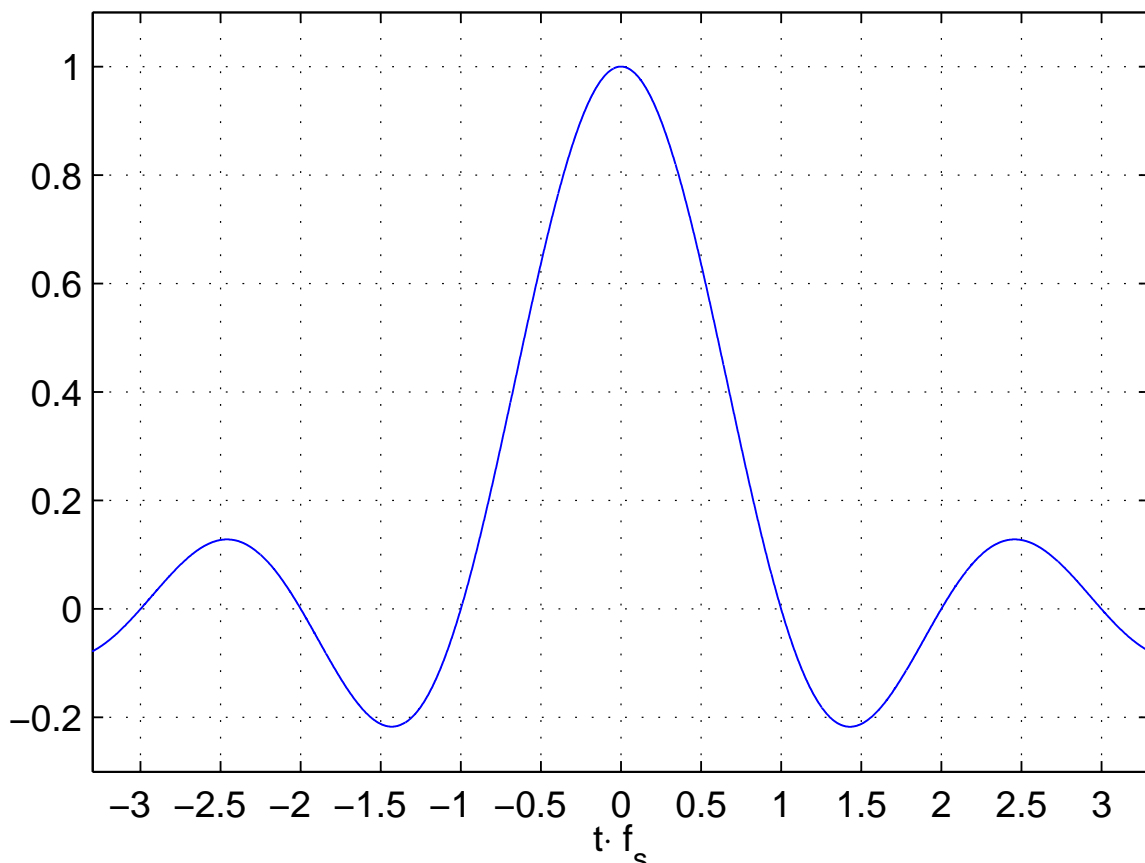
$$h(t) = \frac{\sin \pi t f_s}{\pi t f_s}.$$

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 rectangular filter with cut−off frequency $f_s/2$

Band-pass signal sampling

Sampled signals can be reconstructed as long as their doublesided bandwidth (counting both the positive and negative frequency range) does not exceed the sampling frequency. The anti-aliasing filter used can also be a band-pass filter instead of a low-pass filter.

A suitably bandlimited signal $x(t)$ with center frequency f_c sampled at frequency f_s can be reconstructed by convolution with

$$h(t) = \frac{\sin \pi t f_s}{\pi t f_s} \cdot \cos(2\pi t f_c).$$
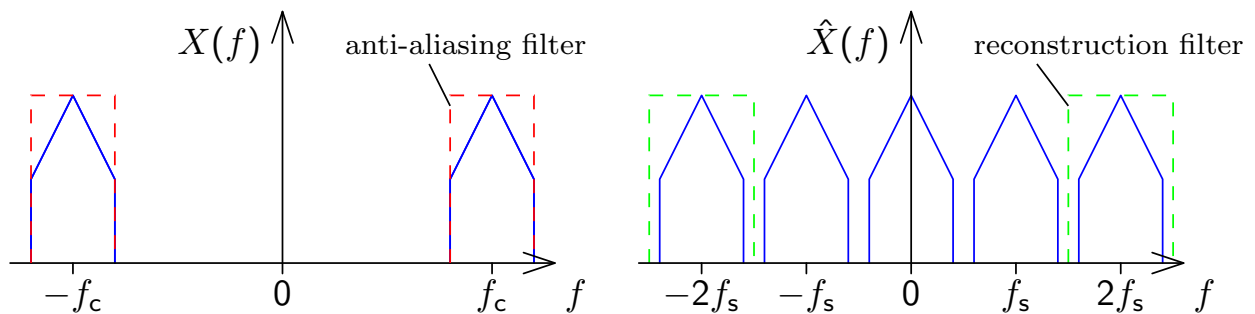
If the signal spectrum is symmetric around f_c, and f_c is a multiple of the sampling frequency f_s, then aliases of the negative frequency spectrum will exactly match those of the positive half.

49

Exercise 6

- Generate a one second long Gaussian noise sequence $\{r_n\}$ (using MATLAB function `randn`) with a sampling rate of 300 Hz.

- Use the `fir1(50, 45/150)` function to design a finite impulse response low-pass filter with a cut-off frequency of 45 Hz. Use the `filtfilt` function in order to apply that filter to the generated noise signal, resulting in the filtered noise signal $\{x_n\}$.

- Then sample $\{x_n\}$ at 100 Hz by setting all but every third sample value to zero, resulting in sequence $\{y_n\}$.

- Generate another low-pass filter with a cut-off frequency of 50 Hz and apply it to $\{y_n\}$, in order to interpolate the reconstructed filtered noise signal $\{z_n\}$. Multiply the result by three, to compensate the energy lost during sampling.

- Plot $\{x_n\}$, $\{y_n\}$, and $\{z_n\}$. Finally compare $\{x_n\}$ and $\{z_n\}$.

Why should the first filter have a lower cut-off frequency than the second?

50

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.
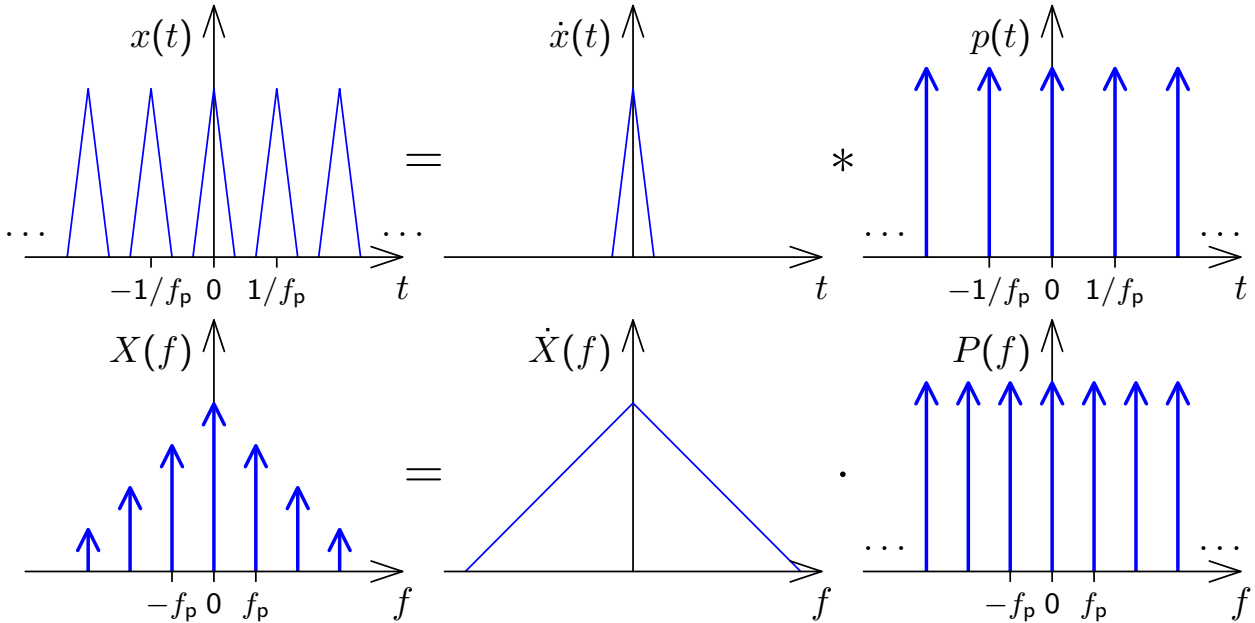
Spectrum of a sampled signal

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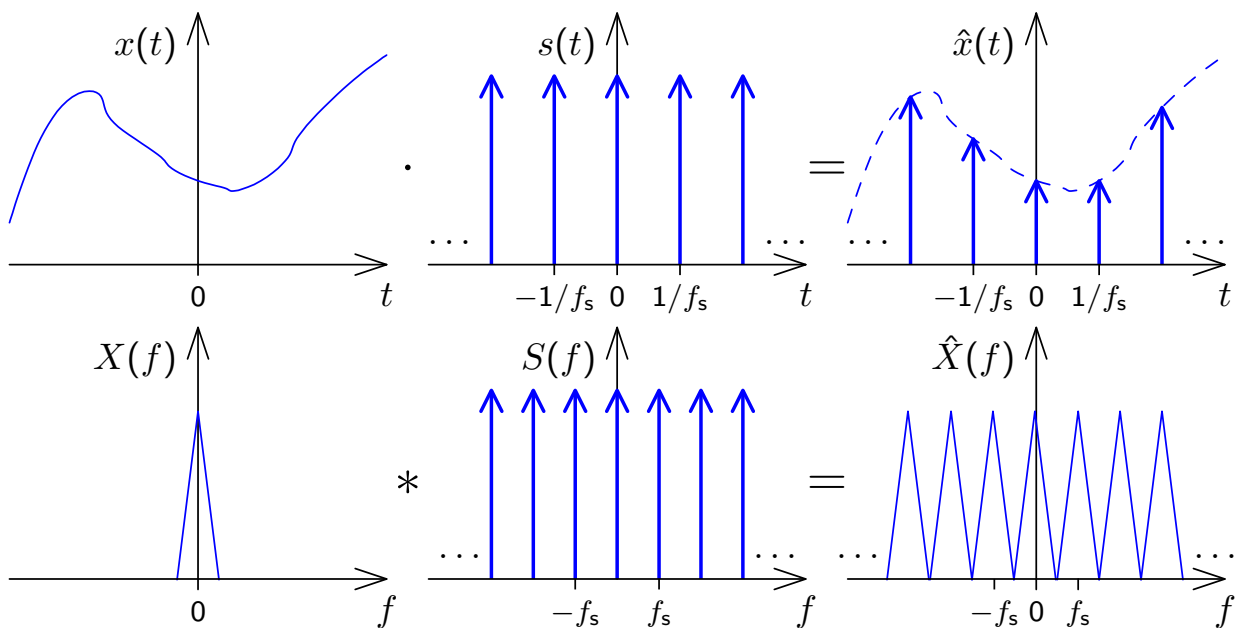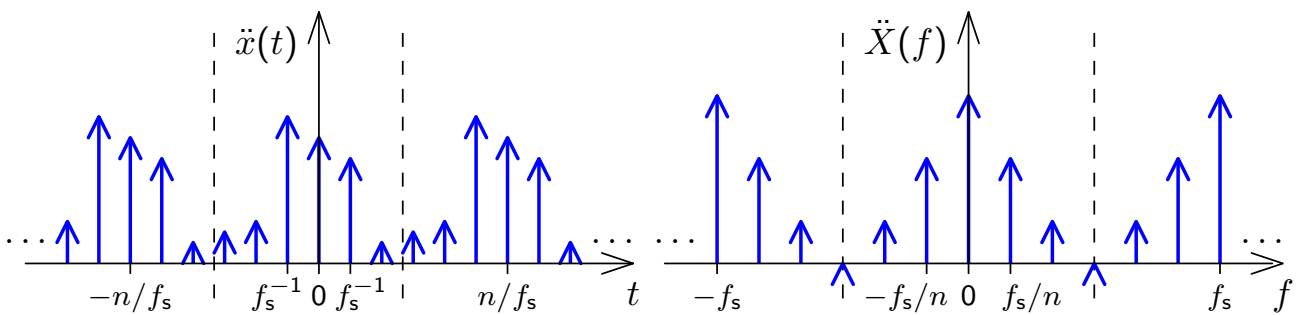
Continuous vs discrete Fourier transform

- 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.

53

Discrete Fourier Transform (DFT)

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

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

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

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

54

Discrete Fourier Transform visualized

$$\begin{pmatrix} \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ \\ \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ \\ \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ \\ \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ \\ \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ \\ \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ \\ \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ \\ \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} = \begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \end{pmatrix}$$

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, $2f_{\text{s}}/n$, $3f_{\text{s}}/n$, $\ldots$, $f_{\text{s}}/2$, 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 $f_{\text{s}}/2$?

Fast Fourier Transform (FFT)

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

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

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

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

$$\forall i : x_i = \quad \Re(x_i) \iff \forall i : X_{n-i} = \quad X_i^*$$

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

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

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

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 trick 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?

$$\{z_i\} = \mathcal{F}^{-1}\left(\mathcal{F}\{x_i\} \cdot \mathcal{F}\{y_i\}\right)$$
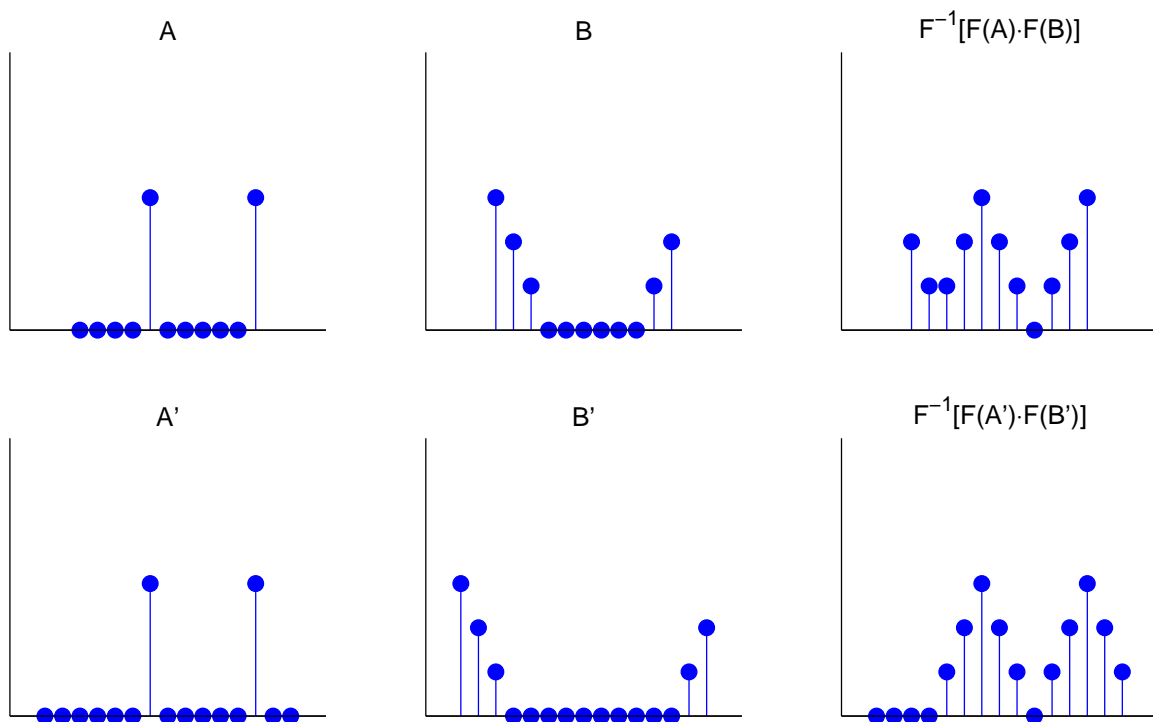
There is obviously no problem if this condition is fulfilled:

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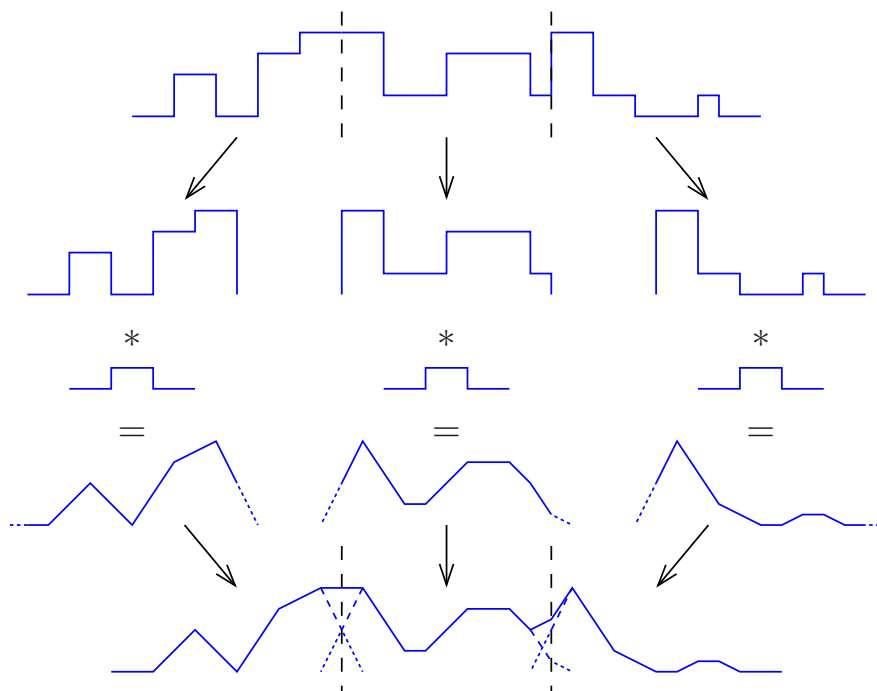
The regions originally added as zero padding are, after convolution, aligned to overlap with the unpadded ends of their respective neighbour blocks. The overlapping parts of the blocks are then added together.

Deconvolution

A signal $u(t)$ was distored 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) $u(t)$?
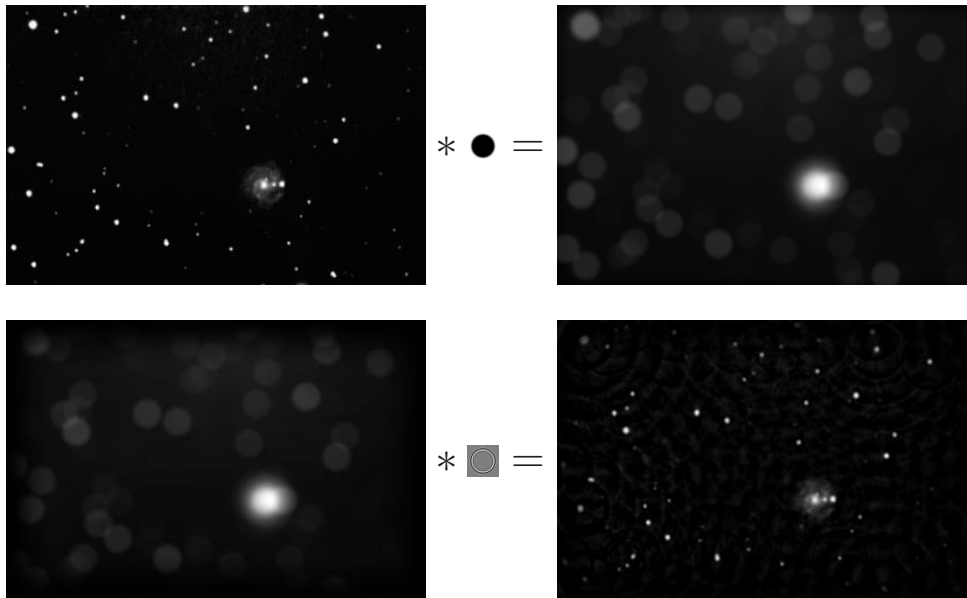
The convolution theorem turns the problem into one of multiplication:

$$s(t) \;=\; \int u(t-\tau) \cdot h(\tau) \cdot \mathrm{d}\tau$$

$$s \;=\; u * h$$

$$\mathcal{F}\{s\} \;=\; \mathcal{F}\{u\} \cdot \mathcal{F}\{h\}$$

$$\mathcal{F}\{u\} \;=\; \mathcal{F}\{s\}/\mathcal{F}\{h\}$$

$$u \;=\; \mathcal{F}^{-1}\{\mathcal{F}\{s\}/\mathcal{F}\{h\}\}$$

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

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

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:

$\longrightarrow$ Modify the Fourier transform of the impulse response, such that $|\mathcal{F}\{h\}(f)| > \epsilon$ for some experimentally chosen threshold ϵ.

$\longrightarrow$ 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:

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

Exercise 7 Use MATLAB to deconvolve the blurred stars from slide 26.

The files `stars-blurred.png` with the blurred-stars image and `stars-psf.png` with the impulse response (point-spread function) are available on the course-material web page. You may find the MATLAB functions `imread`, `double`, `imagesc`, `circshift`, `fft2`, `ifft2` of use.

Try different ways to control the noise (see above) and distortions near the margins (windowing). [The MATLAB image processing toolbox provides ready-made "professional" functions `deconvwnr`, `deconvreg`, `deconvlucy`, `edgetaper`, for such tasks. Do not use these, except perhaps to compare their outputs with the results of your own attempts.]

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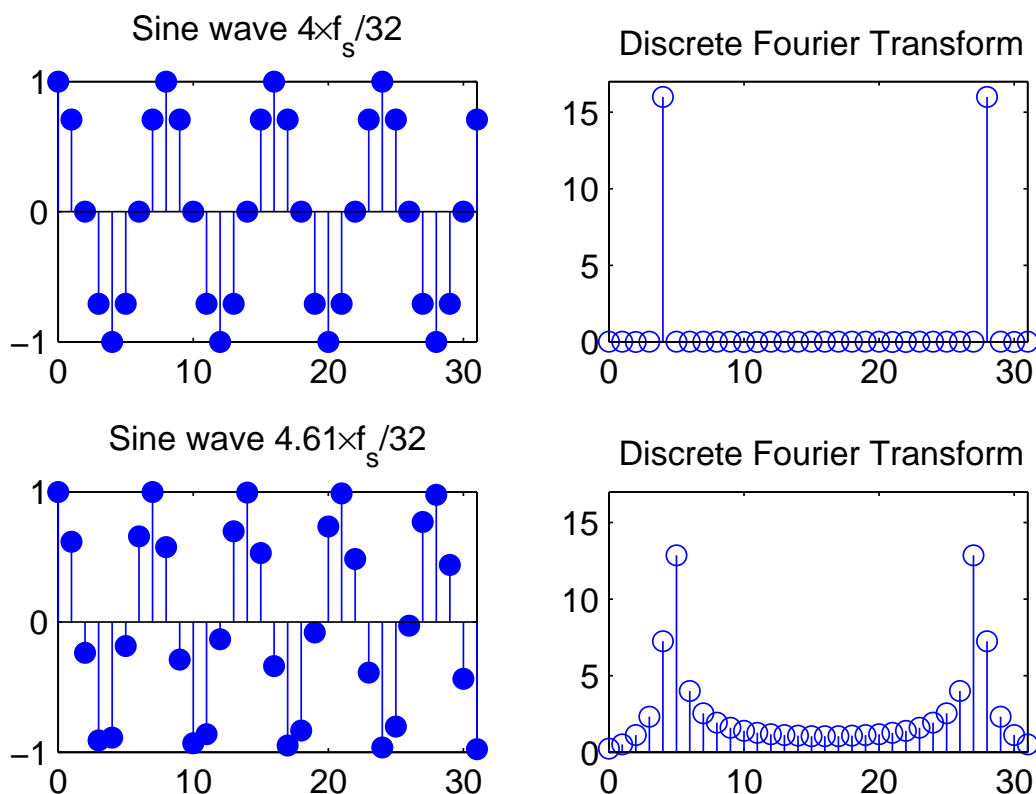
We introduced the DFT as a special case of the continuous Fourier transform, where the input is sampled *and periodic*.

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:

$\longrightarrow$ Its amplitude is lower (down to 63.7%).

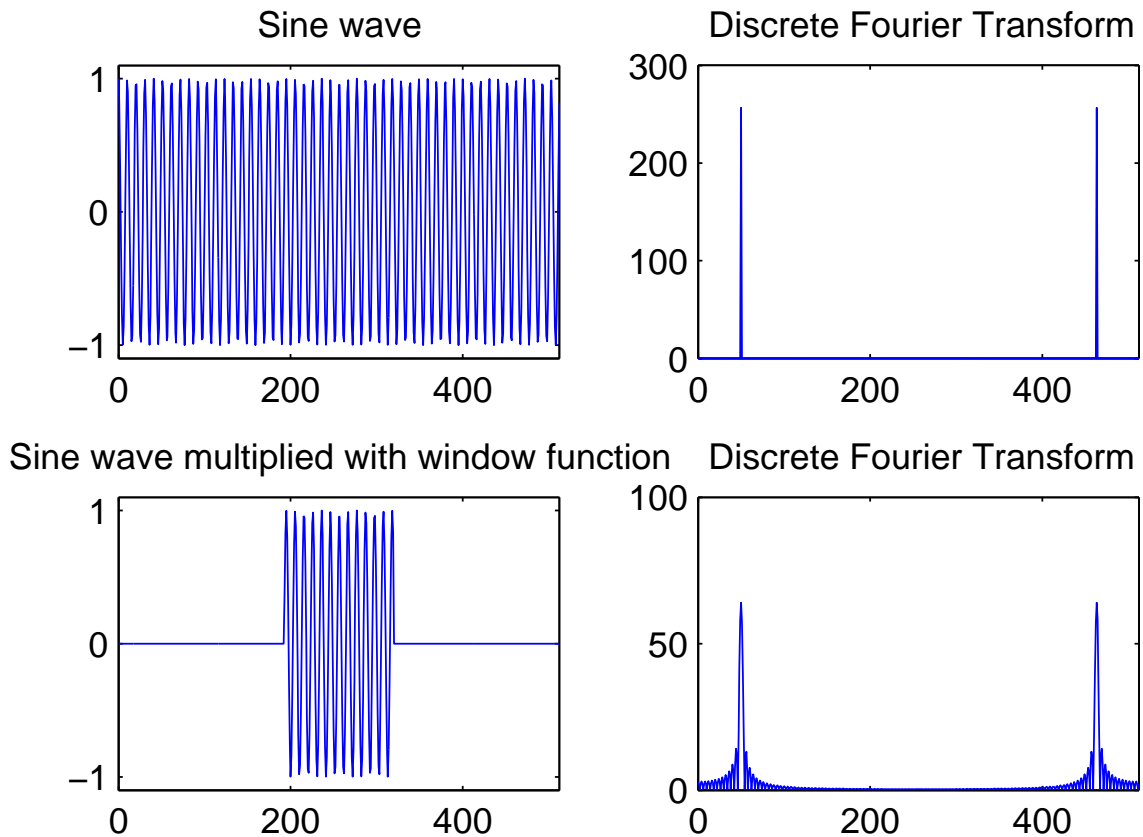$\longrightarrow$ Much signal energy has "leaked" to other frequencies.
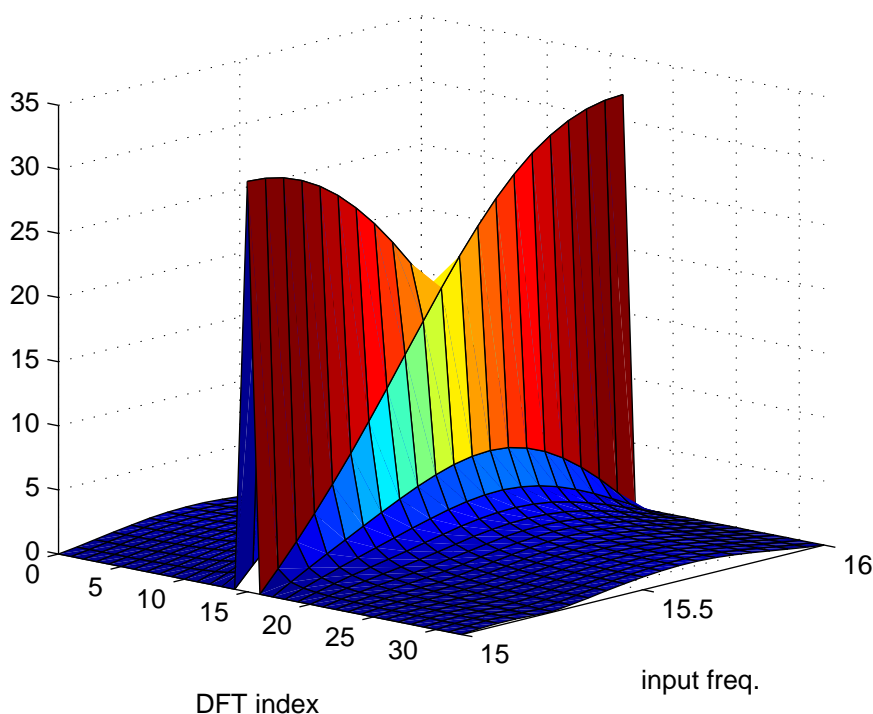
The *leakage* of energy to other frequency bins not only blurs the estimated spectrum. The peak amplitude also changes significantly as the frequency of a tone changes from that associated with one output bin to the next, a phenomenon known as *scalloping*. In the above graphic, an input sine wave gradually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

Windowing

Sine wave / Discrete Fourier Transform / Sine wave multiplied with window function / Discrete Fourier Transform

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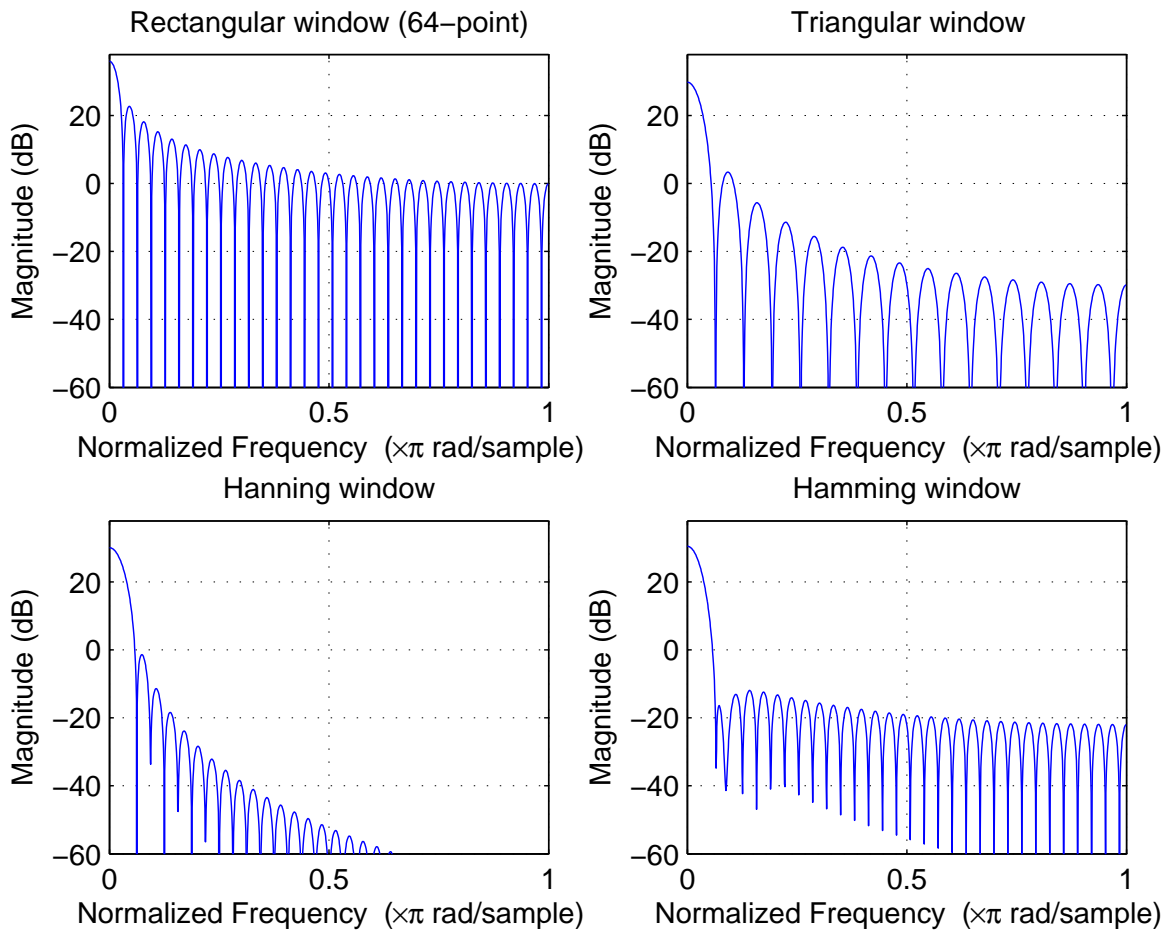
Some better window functions
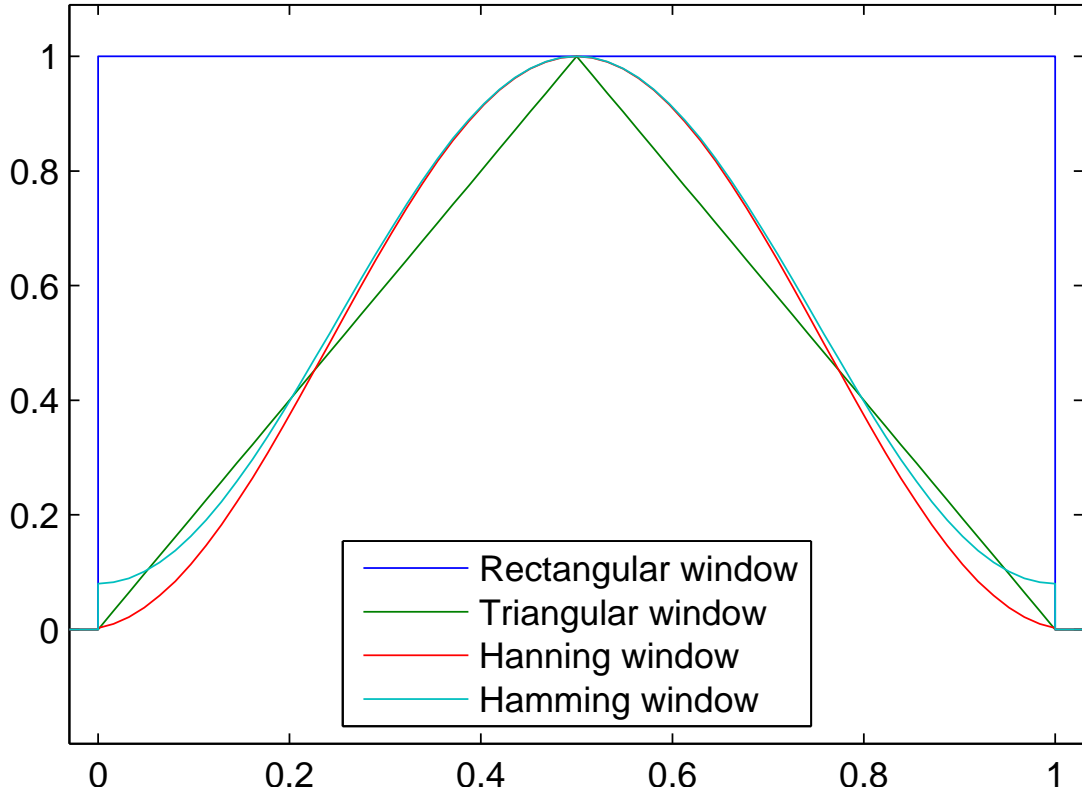
All these functions are 0 outside the interval [0,1].

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):

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

Hanning window (raised-cosine window, Hann 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

$$s_i = \cos(2\pi \cdot 3i/16) + \cos(2\pi \cdot 4i/16), \qquad i \in \{0, \dots, 15\}.$$
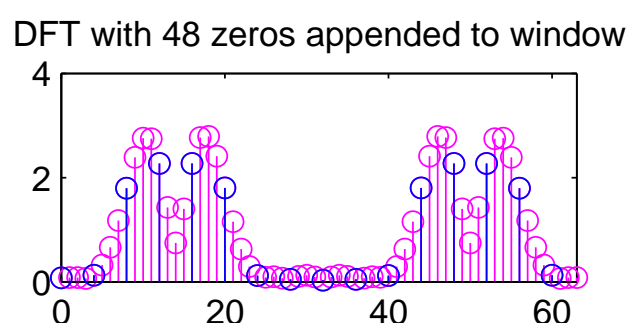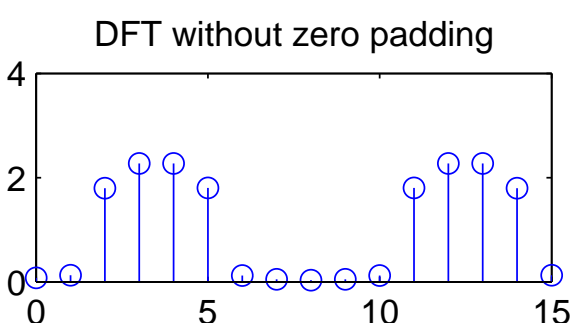
The left plot shows the DFT of the windowed sequence

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

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.

DFT without zero padding

DFT with 48 zeros appended to 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.

Frequency inversion

In order to turn the spectrum $X(f)$ of a real-valued signal x_i sampled at f_s into an inverted spectrum $X'(f) = X(f_\mathsf{s}/2 - f)$, we merely have to shift the periodic spectrum by $f_\mathsf{s}/2$:
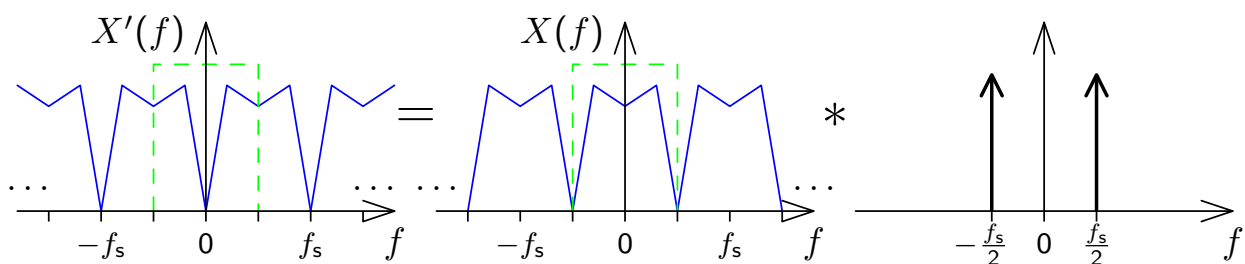
This can be accomplished by multiplying the sampled sequence x_i with $y_i = \cos \pi f_\mathsf{s} t = \cos \pi i$, which is nothing but multiplication with the sequence

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

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

Window-based design of FIR filters

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

$$H(f) = \begin{cases} 1 & \text{if } -f_c < f \le f_c \\ 0 & \text{otherwise} \end{cases}$$

and the impulse response

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

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:

$\longrightarrow$ the impulse response is infinitely long

$\longrightarrow$ this filter is not causal, that is $h(t) \ne 0$ for $t < 0$

Solutions:

$\longrightarrow$ Make the impulse response finite by multiplying the sampled $h(t)$ with a windowing function

$\longrightarrow$ 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

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

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

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

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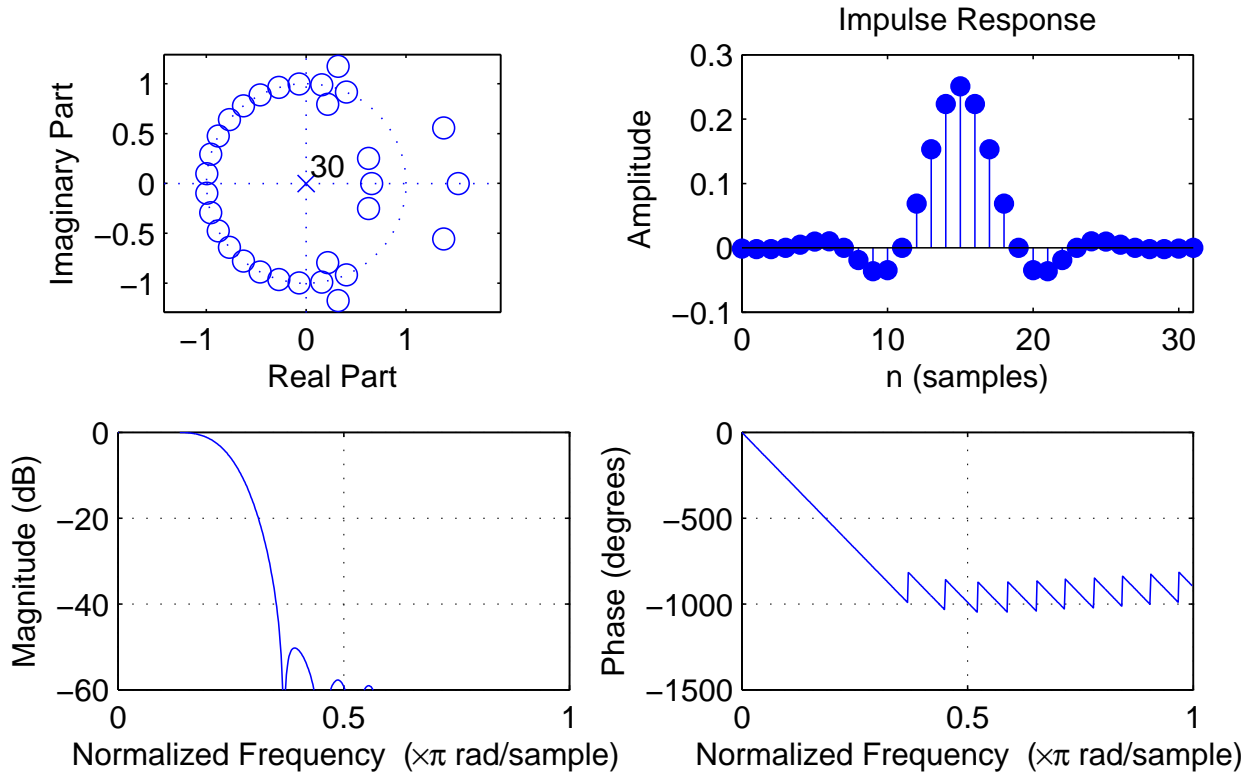
order: $n = 30$, cutoff frequency (-6 dB): $f_c = 0.25 \times f_s/2$, window: Hamming

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.

Converting a low-pass into a band-pass filter

To obtain a band-pass filter that attenuates all frequencies f outside the range $f_l < f < f_h$, we first design a low-pass filter with a cut-off frequency $(f_h - f_l)/2$ and multiply its impulse response with a sine wave of frequency $(f_h + f_l)/2$, before applying the usual windowing:

$$h_i = \frac{\sin[\pi(i - n/2)(f_h - f_l)/f_s]}{\pi(i - n/2)(f_h - f_l)/f_s} \cdot \sin[\pi(f_h + f_l)] \cdot w_i$$
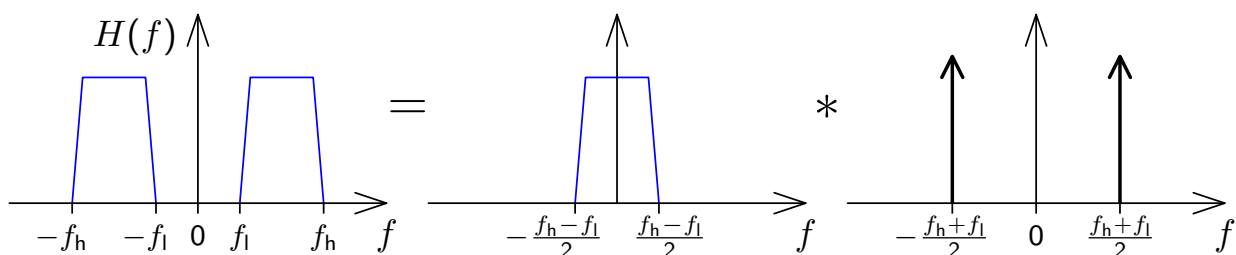
Exercise 8   Explain the difference between the DFT, FFT, and FFTW.

Exercise 9   Push-button telephones use a combination of two sine tones to signal, which button is currently being pressed:

	1209 Hz	1336 Hz	1477 Hz	1633 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

(a) You receive a digital telephone signal with a sampling frequency of 8 kHz. You cut a 256-sample window out of this sequence, multiply it with a windowing function and apply a 256-point DFT. What are the indices where the resulting vector $(X_0, X_1, \ldots, X_{255})$ will show the highest amplitude if button $\boxed{9}$ was pushed at the time of the recording?

(b) Use MATLAB to determine, which button sequence was typed in the touch tones recorded in the file `touchtone.wav` on the course-material web page.

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:

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

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

$$\texttt{conv([1 2 3], [2 1])} \text{ equals } \texttt{[2 5 8 3]}$$

Convolution of sequences is equivalent to polynomial multiplication:

$$\{h_n\} * \{x_n\} \;=\; \{y_n\} \quad \Rightarrow \quad y_n = \sum_{k=-\infty}^{\infty} h_k \cdot x_{n-k}$$

$$\downarrow \qquad \downarrow$$

$$H(v) \cdot X(v) \;=\; \left(\sum_{n=-\infty}^{\infty} h_n v^n \right) \cdot \left(\sum_{n=-\infty}^{\infty} x_n v^n \right)$$

$$= \sum_{n=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} h_k \cdot x_{n-k} \cdot v^n$$

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

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

Example of polynomial division:

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

$$
\begin{array}{r}
1 \;+\; av \;+\; a^2v^2 \;+\; \cdots \\
\hline
1-av \,\big|\, 1 \\
\underline{1 \;-\; av} \\
av \\
\underline{av \;-\; a^2v^2} \\
a^2v^2 \\
\underline{a^2v^2 \;-\; a^3v^3} \\
\cdots
\end{array}
$$

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 is 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 Fourier transform, which it contains on the complex unit circle ($|z| = 1$):

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

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

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

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

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

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.

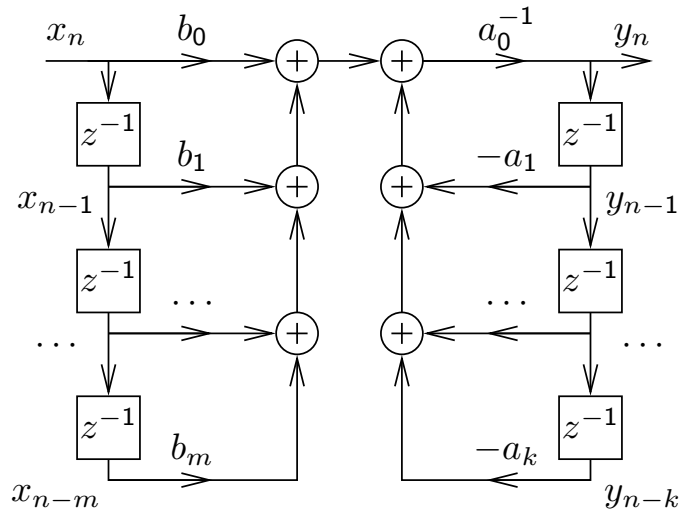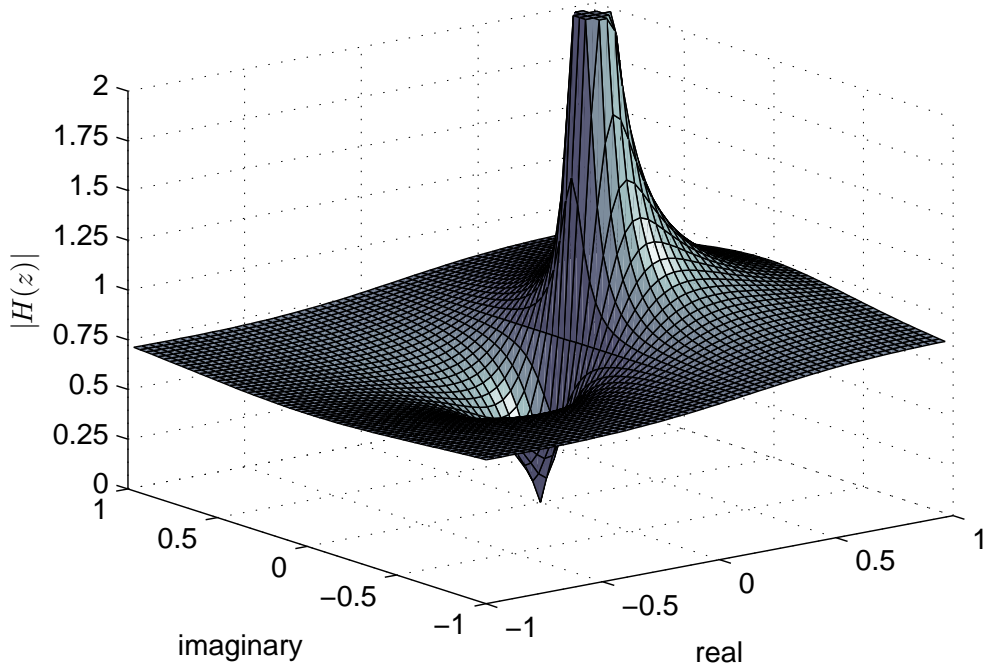As with the Fourier transform, convolution in the time domain corresponds to complex multiplication in the z-domain:

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

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

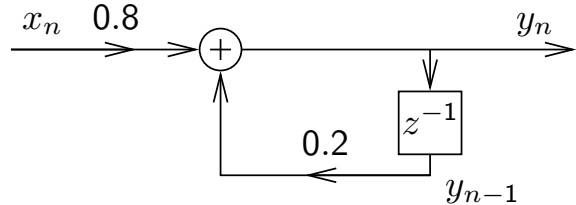$$\{x_{n-\Delta n}\} \;\bullet\!\!-\!\!\circ\; X(z) \cdot z^{-\Delta n}$$
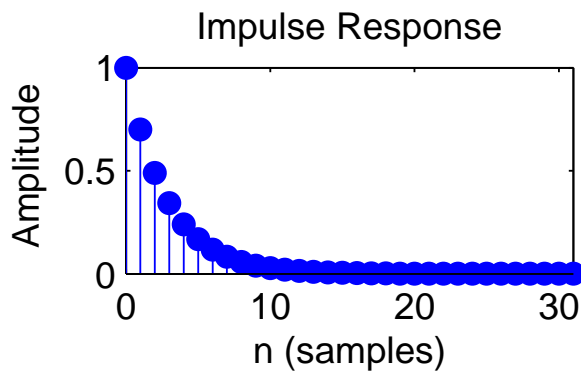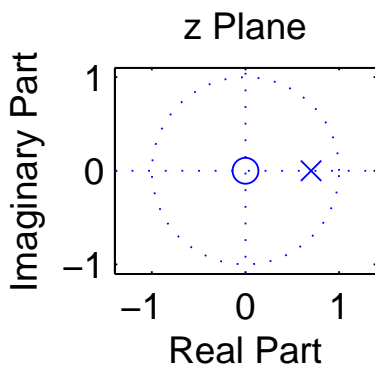
This example is an amplitude plot of

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

which features a zero at 0 and a pole at 0.2.

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

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

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

z Plane

Impulse Response

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

z Plane

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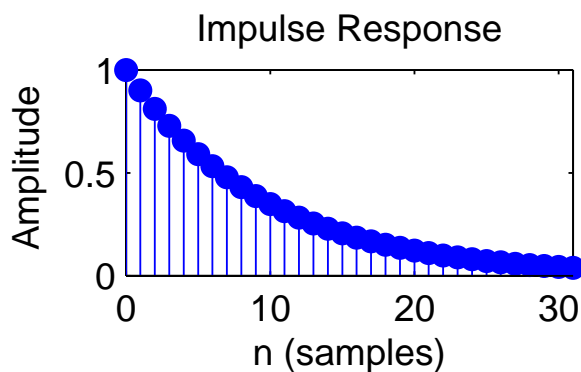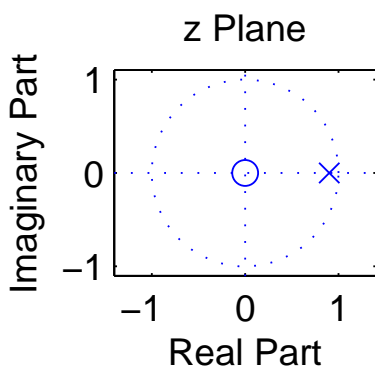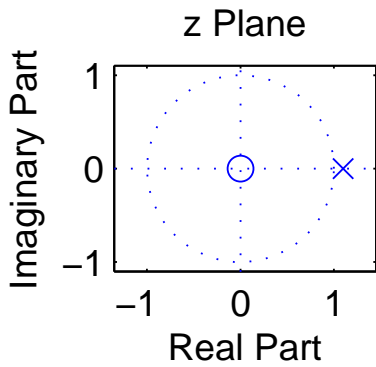
$$H(z) = \frac{z^2}{(z-0.9 \cdot e^{j\pi/6}) \cdot (z-0.9 \cdot e^{-j\pi/6})} = \frac{1}{1-1.8\cos(\pi/6)z^{-1}+0.9^2 \cdot z^{-2}}$$

z Plane

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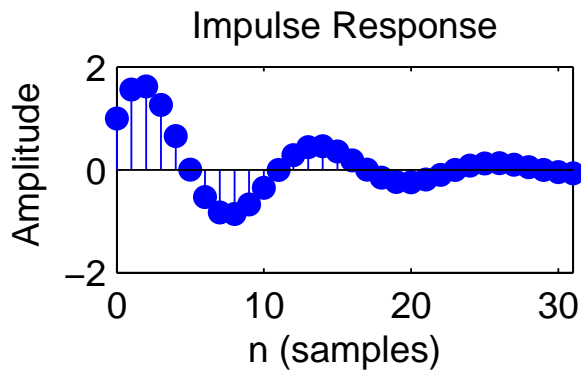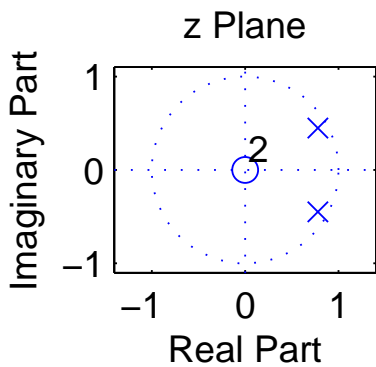
$$H(z) = \frac{z^2}{(z-e^{j\pi/6}) \cdot (z-e^{-j\pi/6})} = \frac{1}{1-2\cos(\pi/6)z^{-1}+z^{-2}}$$

z Plane

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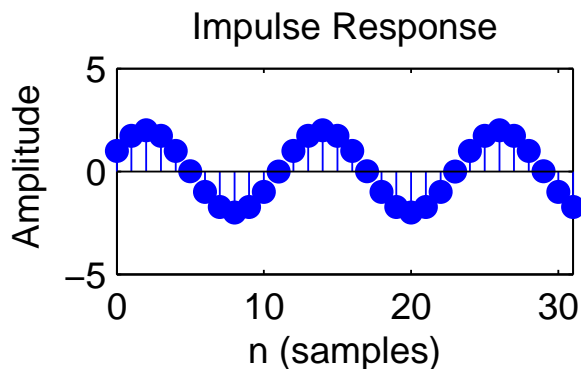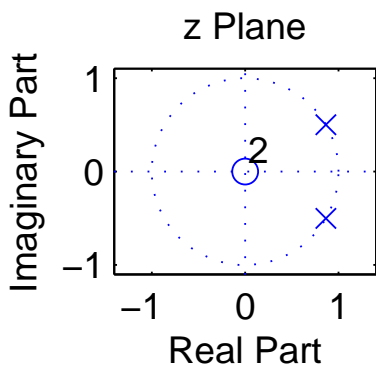
$$H(z) = \frac{z^2}{(z-0.9 \cdot e^{j\pi/2}) \cdot (z-0.9 \cdot e^{-j\pi/2})} = \frac{1}{1-1.8\cos(\pi/2)z^{-1}+0.9^2 \cdot z^{-2}} = \frac{1}{1+0.9^2 \cdot z^{-2}}$$

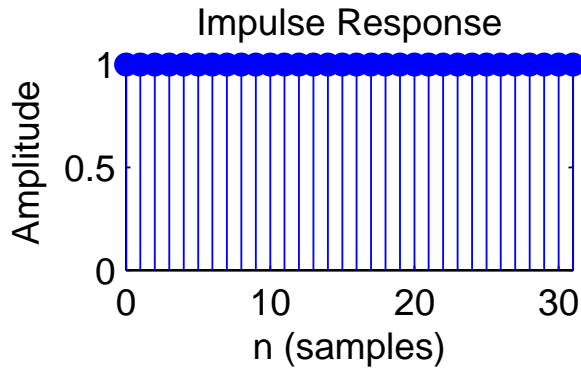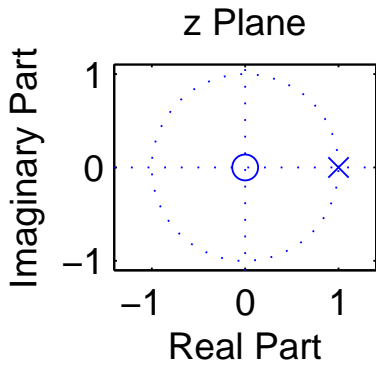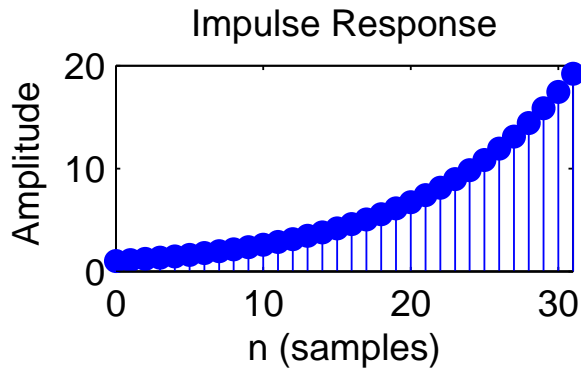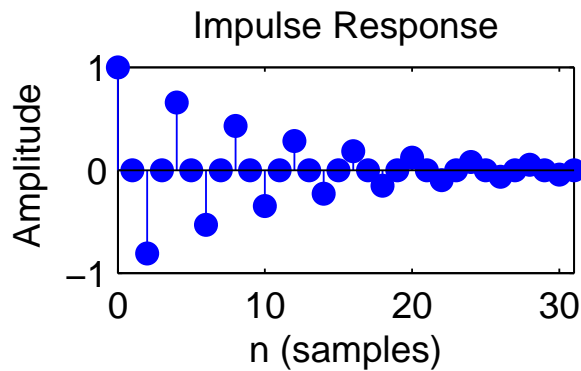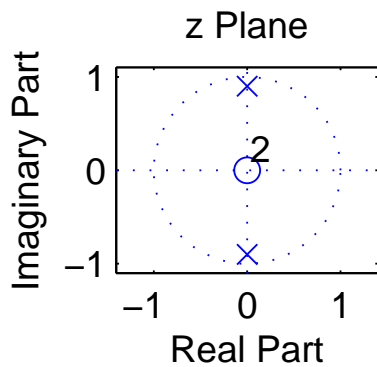$$H(z) = \frac{z}{z+1} = \frac{1}{1+z^{-1}}$$
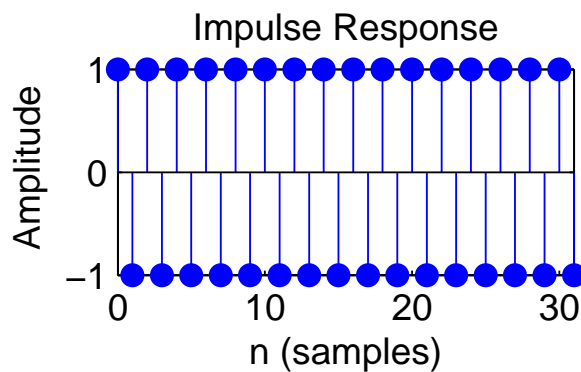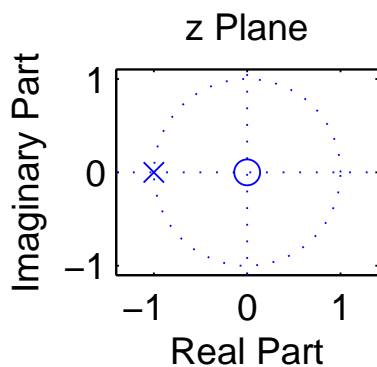
IIR Filter design techniques

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

$\longrightarrow$ The *passband* is the frequency range where we want to approximate a gain of one.

$\longrightarrow$ The *stopband* is the frequency range where we want to approximate a gain of zero.

$\longrightarrow$ 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.

$\longrightarrow$ 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.

$\longrightarrow$ 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.

The designer can then trade off conflicting goals such as a small transition band, a low order, a low ripple amplitude, or even an 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.

All these filter design techniques are implemented in the MATLAB Signal Processing Toolbox in the functions `butter`, `cheby1`, `cheby2`, and `ellip`, which 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.

Butterworth filter design example
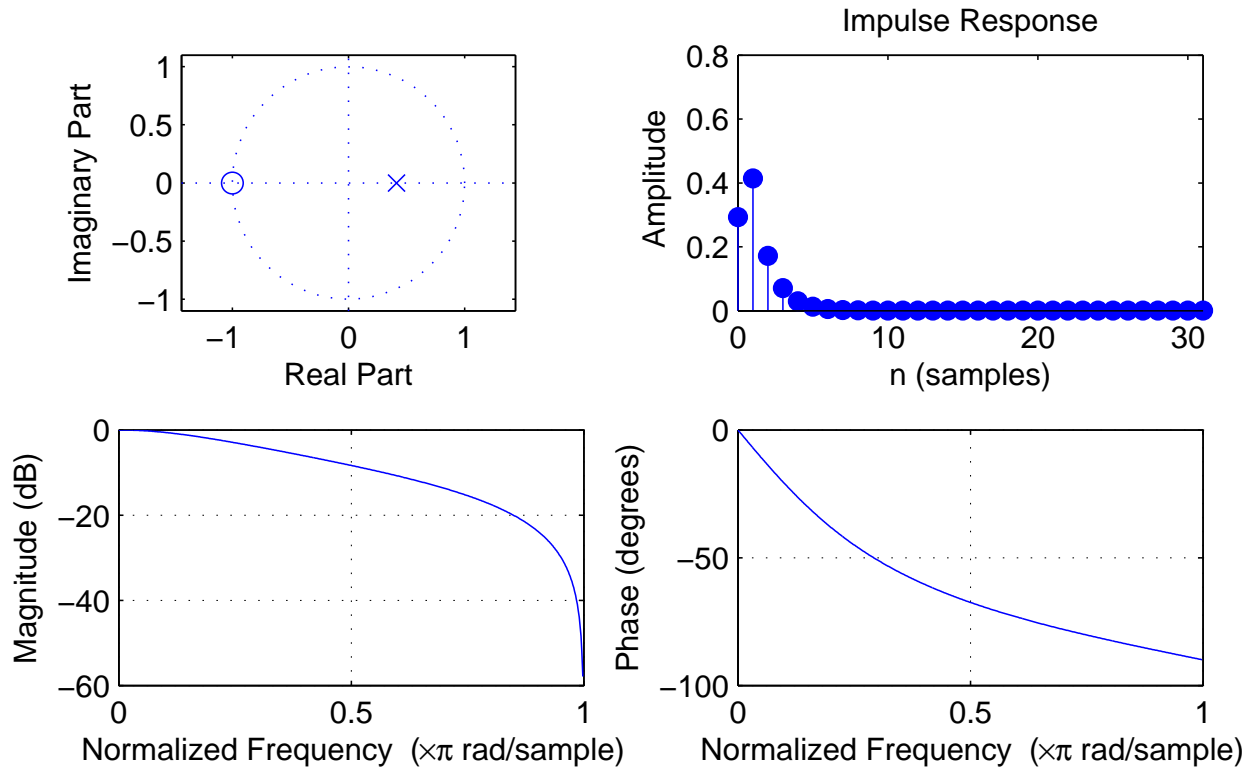
order: 1, cutoff frequency (-3 dB): $0.25 \times f_\mathsf{s}/2$

Butterworth filter design example
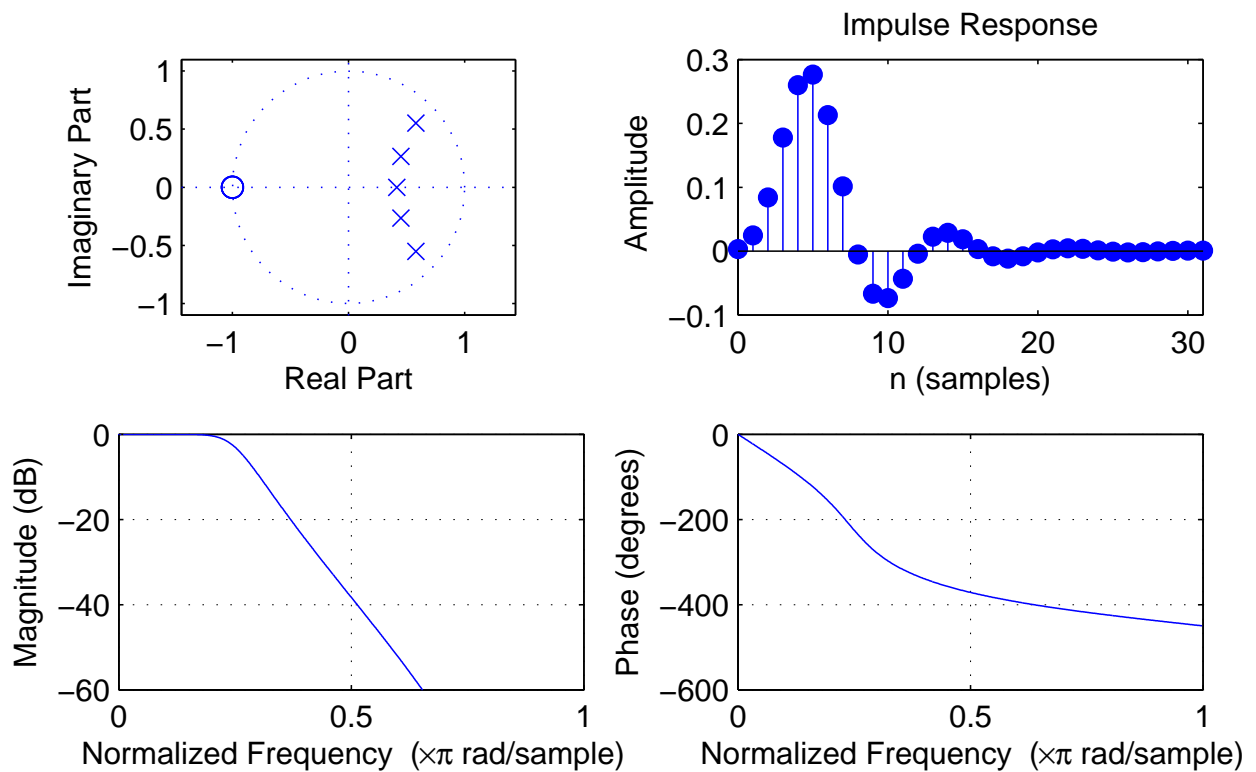
order: 5, cutoff frequency (-3 dB): $0.25 \times f_\mathsf{s}/2$

Chebyshev type I filter design example
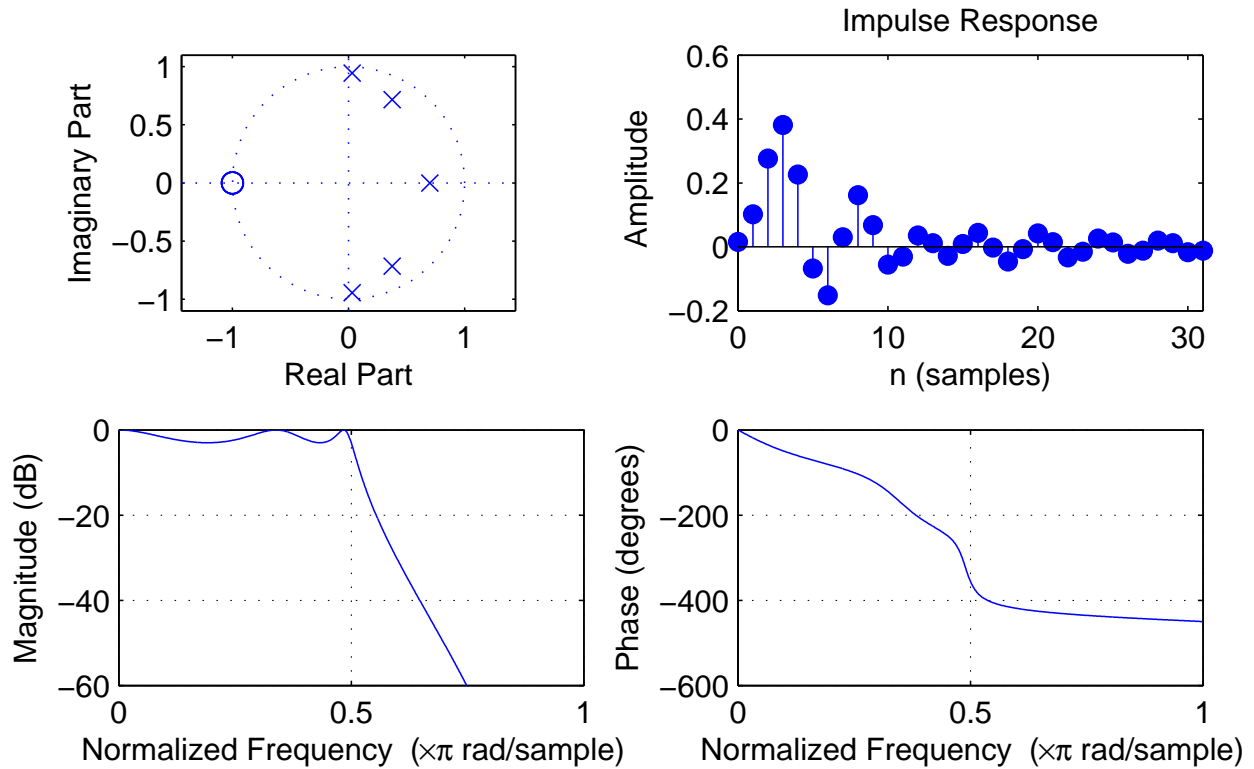
order: 5, cutoff frequency: $0.5 \times f_s/2$, pass-band ripple: -3 dB

Chebyshev type II filter design example
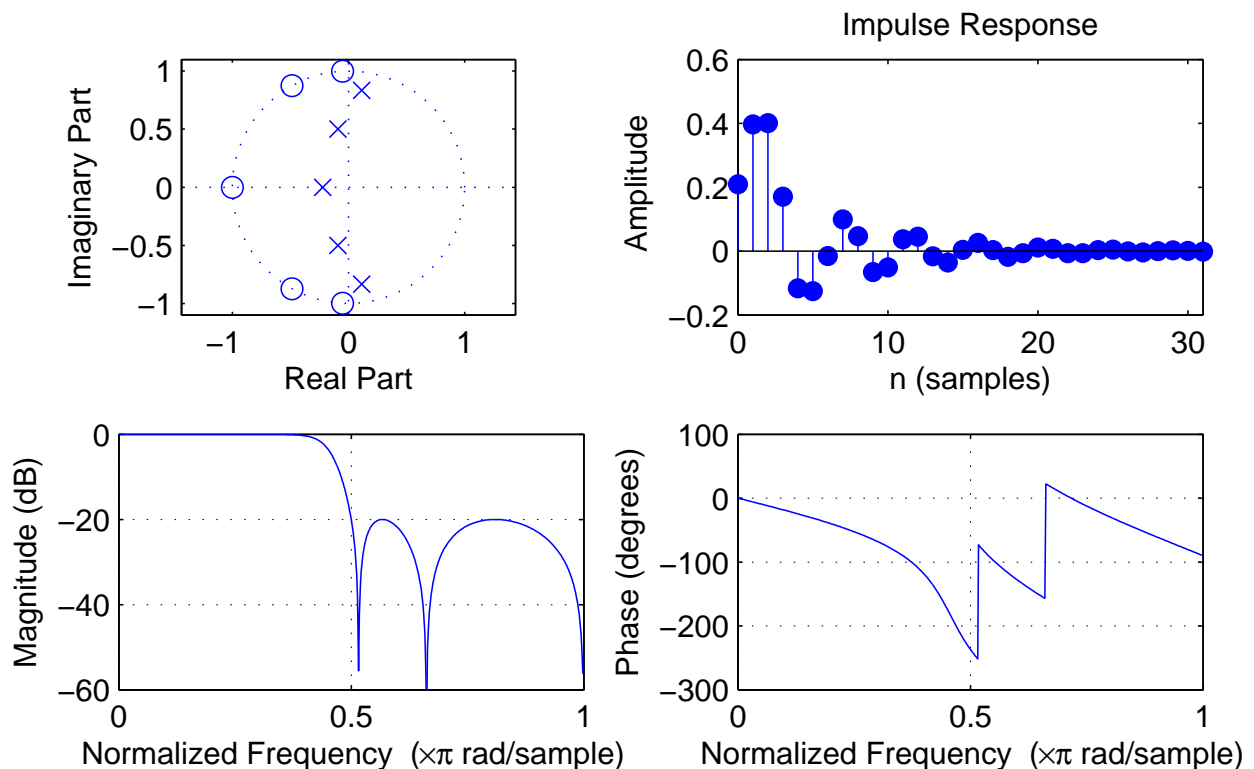
order: 5, cutoff frequency: $0.5 \times f_s/2$, stop-band ripple: -20 dB

Elliptic filter design example
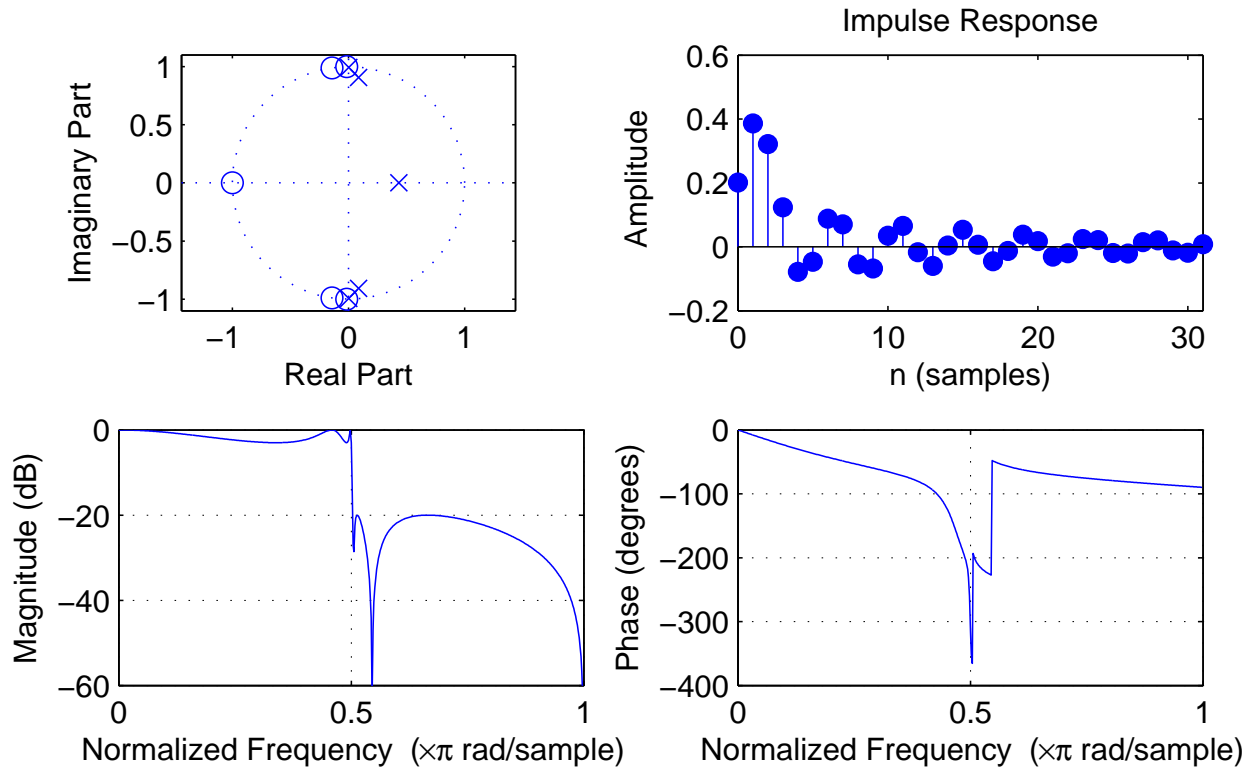
order: 5, cutoff frequency: $0.5 \times f_\text{s}/2$, pass-band ripple: -3 dB, stop-band ripple: -20 dB

Exercise 10 Draw the direct form II block diagrams of the causal infinite-impulse response filters described by the following z-transforms and write down a formula describing their time-domain impulse responses:

(a) $H(z) = \dfrac{1}{1 - \frac{1}{2}z^{-1}}$

(b) $H'(z) = \dfrac{1 - \frac{1}{4^4}z^{-4}}{1 - \frac{1}{4}z^{-1}}$

(c) $H''(z) = \dfrac{1}{2} + \dfrac{1}{4}z^{-1} + \dfrac{1}{2}z^{-2}$

Exercise 11 (a) Perform the polynomial division of the rational function given in exercise 10 (a) until you have found the coefficient of z^{-5} in the result.

(b) Perform the polynomial division of the rational function given in exercise 10 (b) until you have found the coefficient of z^{-10} in the result.

(c) Use its z-transform to show that the filter in exercise 10 (b) has actually a finite impulse response and draw the corresponding block diagram.

Exercise 12 Consider the system $h : \{x_n\} \rightarrow \{y_n\}$ with $y_n + y_{n-1} = x_n - x_{n-4}$.

(a) Draw the direct form I block diagram of a digital filter that realises h.

(b) What is the impulse response of h?

(c) What is the step response of h (i.e., $h * u$)?

(d) Apply the z-transform to (the impulse response of) h to express it as a rational function $H(z)$.

(e) Can you eliminate a common factor from nominator and denominator? What does this mean?

(f) For what values $z \in \mathbb{C}$ is $H(z) = 0$?

(g) How many poles does H have in the complex plane?

(h) Write H as a fraction using the position of its poles and zeros and draw their location in relation to the complex unit circle.

(i) If h is applied to a sound file with a sampling frequency of 8000 Hz, sine waves of what frequency will be eliminated and sine waves of what frequency will be quadrupled in their amplitude?

Random sequences and noise

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

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

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

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

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

$$P_{\mathbf{x}_n}(a) = \mathrm{Prob}(\mathbf{x}_n \leq a)$$

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

$$P_{\mathbf{x}_{n_1}, \ldots, \mathbf{x}_{n_k}}(a_1, \ldots, a_k) = \mathrm{Prob}(\mathbf{x}_{n_1} \leq a_1 \wedge \ldots \wedge \mathbf{x}_{n_k} \leq a_k)$$

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

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

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

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

$\longrightarrow$ *expected value* $\mathcal{E}(\mathbf{x}_n) = \int a p_{\mathbf{x}_n}(a)\, \mathrm{d}a$

$\longrightarrow$ *mean-square value* (average power) $\mathcal{E}(|\mathbf{x}_n|^2) = \int |a|^2 p_{\mathbf{x}_n}(a)\, \mathrm{d}a$

$\longrightarrow$ *variance* $\mathrm{Var}(\mathbf{x}_n) = \mathcal{E}[|\mathbf{x}_n - \mathcal{E}(\mathbf{x}_n)|^2] = \mathcal{E}(|\mathbf{x}_n|^2) - |\mathcal{E}(\mathbf{x}_n)|^2$

$\longrightarrow$ *correlation* $\mathrm{Cor}(\mathbf{x}_n, \mathbf{x}_m) = \mathcal{E}(\mathbf{x}_n \cdot \mathbf{x}_m^*)$

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

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

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

its variance

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

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

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

and its *autocovariance sequence*

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

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

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

and its *crosscovariance sequence*

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

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: $c_{xy}(k) = c_{yx}(-k)$.

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

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

109

In other words:

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

Similarly:

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

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:

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

110

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

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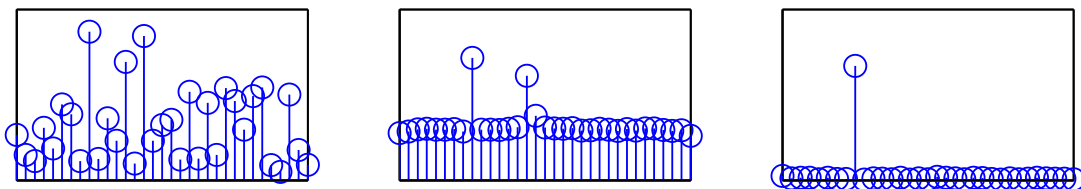
The above diagrams show different types of spectral estimates of a sequence $x_i = \sin(2\pi \mathsf{j} \times 8/64) + \sin(2\pi \mathsf{j} \times 14.32/64) + n_i$ with $\phi_{nn}(i) = 4\delta_i$.

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 $1/p$.

Image, video and audio compression

Structure of modern audiovisual communication systems:

| signal | → | sensor + sampling | → | perceptual coding | → | entropy coding | → | channel coding |

| | | | | | | noise | → | channel |

| human senses | ← | display | ← | perceptual decoding | ← | entropy decoding | ← | channel decoding |
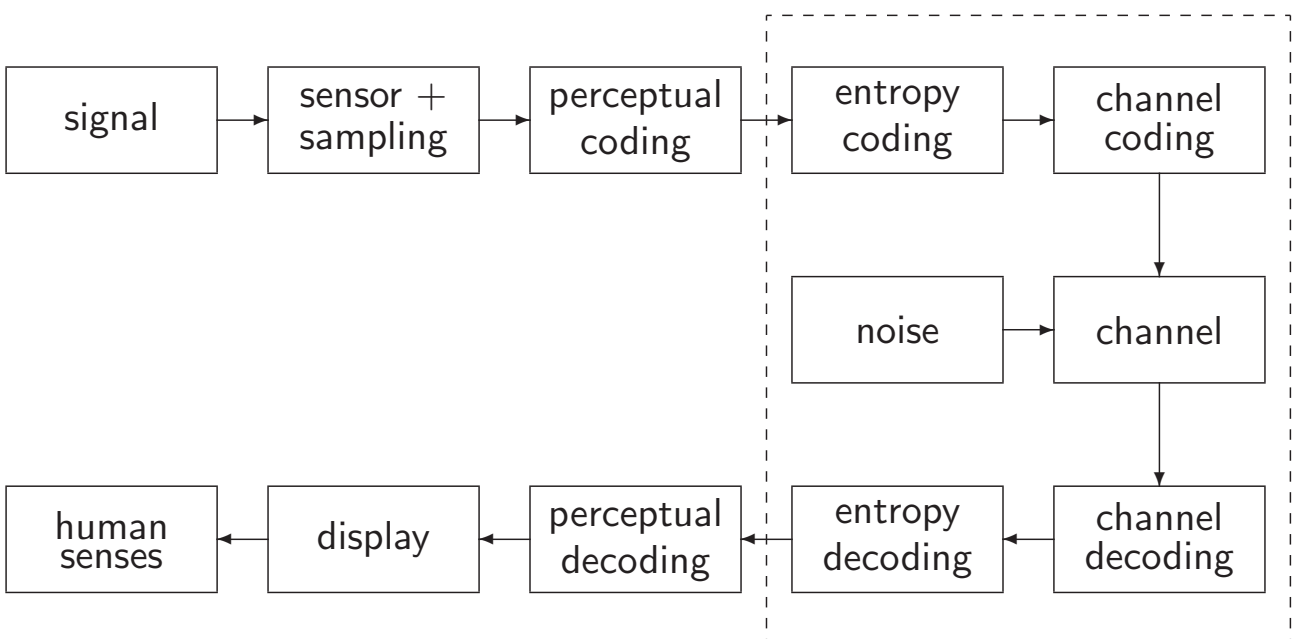
Audio-visual lossy coding today typically consists of these steps:

$\longrightarrow$ A *transducer* converts the original stimulus into a voltage.

$\longrightarrow$ 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.

$\longrightarrow$ 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.

$\longrightarrow$ This signal is *quantized* again, based on a *perceptual model* of what level of quantization-noise humans can still sense.

$\longrightarrow$ 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.

$\longrightarrow$ 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

$\longrightarrow$ Quick review of entropy coding

$\longrightarrow$ 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)

$\longrightarrow$ Introduction to some characteristics and limits of human senses

- perceptual scales and sensitivity limits

- colour vision

- human hearing limits, critical bands, audio masking

$\longrightarrow$ Quantization techniques to remove information that is irrelevant to human senses

$\longrightarrow$ Image and audio coding standards

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

Literature

$\longrightarrow$ D. Salomon: A guide to data compression methods. ISBN 0387952608, 2002.

$\longrightarrow$ L. Gulick, G. Gescheider, R. Frisina: Hearing. ISBN 0195043073, 1989.

$\longrightarrow$ 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:

0.0		0.35	0.55	0.75	0.9 0.95 1.0
	u	v	w	x	y z
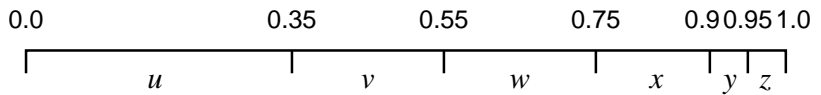
Encode text $wuvw\ldots$ as numeric value $(0.58\ldots)$ in nested intervals:
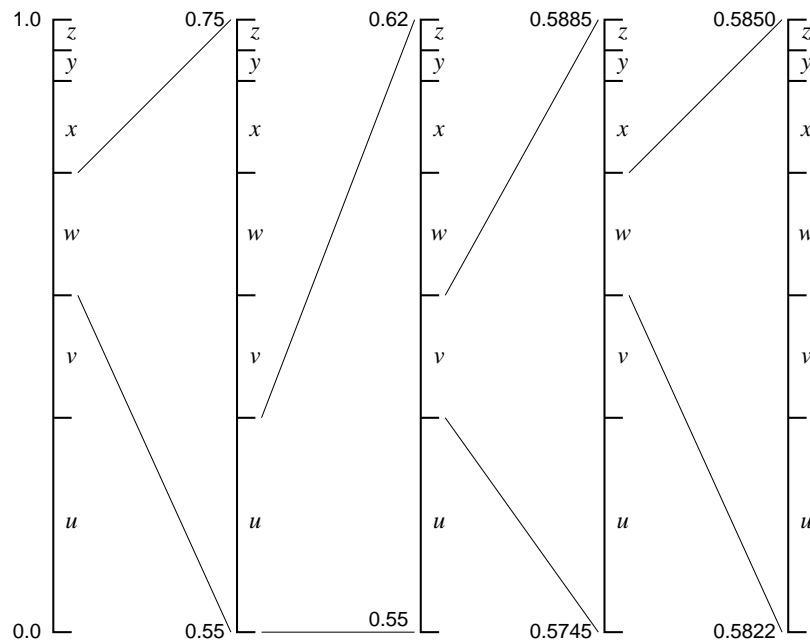
Arithmetic coding

Several advantages:

$\longrightarrow$ Length of output bitstring can approximate the theoretical information content of the input to within 1 bit.

$\longrightarrow$ Performs well with probabilities > 0.5, where the information per symbol is less than one bit.

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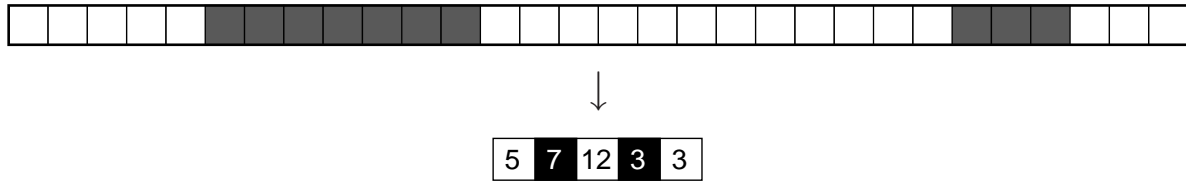
Predictive coding:
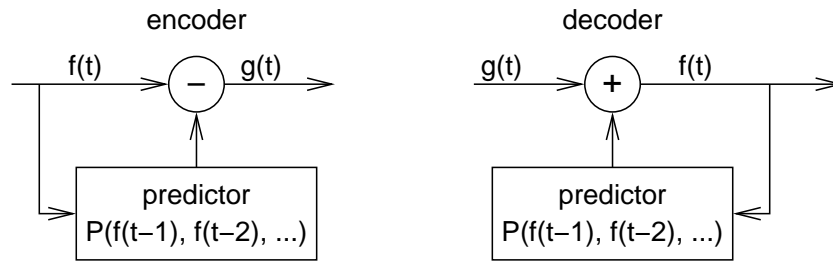
$$\text{Delta coding (DPCM):} \qquad P(x) \;=\; x$$

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

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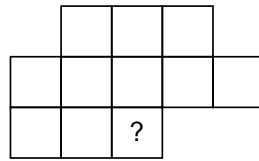
Based on the counted numbers n_{black} and n_{white} of how often each pixel value has been encountered so far in each of the 1024 contexts, the probability for the next pixel being black is estimated as

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

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

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

$$\Rightarrow \quad \exists x, y : \ \ P(X = x \,|\, Y = y) \neq P(X = x) \vee$$
$$P(Y = y \,|\, X = x) \neq P(Y = y)$$
$$\Rightarrow \quad H(X|Y) < H(X) \ \vee \ H(Y|X) < H(Y)$$

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 merely an array of $2^{10} = 1024$ 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

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

where $E(\cdots)$ 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 audiovisual data is a consequence of correlation, which is algorithmically much easier to exploit.

Dependent but not correlated:
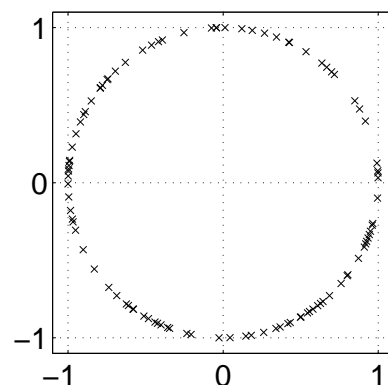
Positive correlation: higher $X \Leftrightarrow$ higher Y, lower $X \Leftrightarrow$ lower Y
Negative correlation: lower $X \Leftrightarrow$ higher Y, higher $X \Leftrightarrow$ lower Y

Correlation of neighbour pixels

Values of neighbour pixels at distance 1

Values of neighbour pixels at distance 2

Values of neighbour pixels at distance 4

Values of neighbour pixels at distance 8
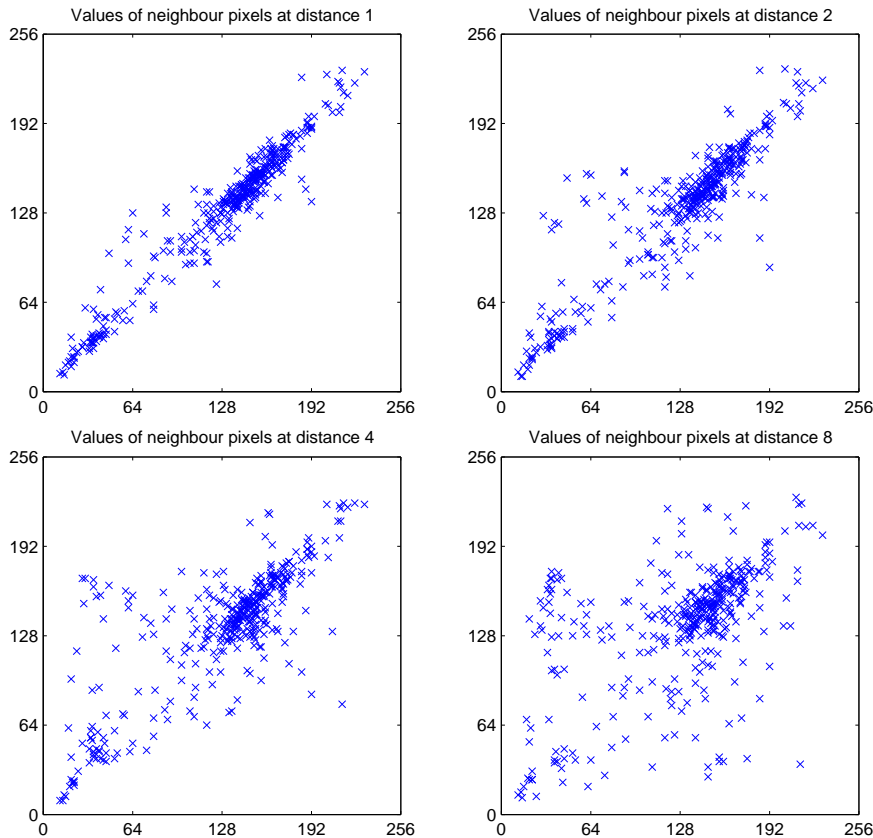
Covariance and correlation

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

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

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

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 $b = E(Y) - E(X)$.

Covariance Matrix

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

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

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

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

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

Decorrelation by coordinate transform

Neighbour–pixel value pairs

Decorrelated neighbour–pixel value pairs

Probability distribution and entropy
- correlated value pair (H = 13.90 bit)
- decorrelated value 1 (H = 7.12 bit)
- decorrelated value 2 (H = 4.75 bit)
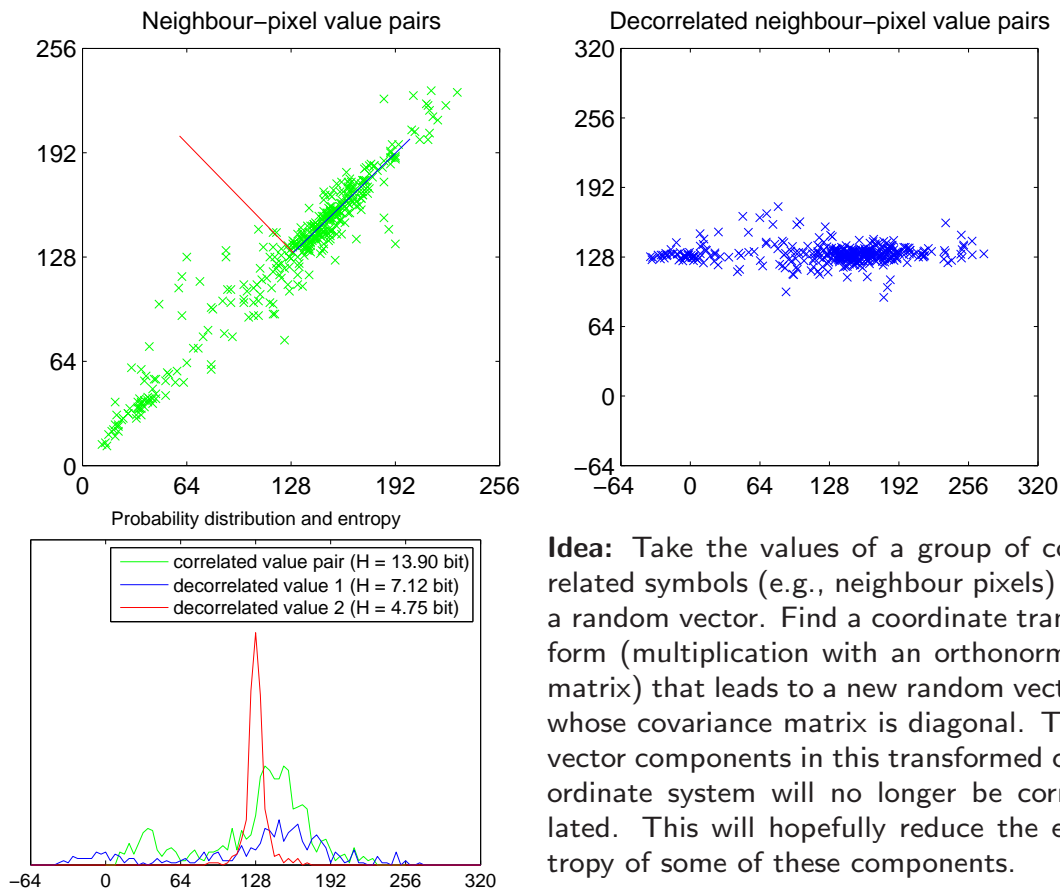
Idea: Take the values of a group of correlated symbols (e.g., neighbour pixels) as a random vector. Find a coordinate transform (multiplication with an orthonormal matrix) that leads to a new random vector whose covariance matrix is diagonal. The vector components in this transformed coordinate system 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

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

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

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

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 $\Lambda = \mathsf{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ 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 $\text{Cov}(\mathbf{X})$.

How can we find a transform matrix A that decorrelates $\mathbf{X}$, i.e. that turns $\text{Cov}(A\mathbf{X}) = A \cdot \text{Cov}(\mathbf{X}) \cdot A^{\mathsf{T}}$ 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 $\text{Cov}(\mathbf{X})$ 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 $b_1, b_2, \ldots, b_n$ of length 1 with

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

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

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

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

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

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

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

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

and therefore shows with

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

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

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

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

135

[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

$$\begin{pmatrix} 0.0328 & 0.0256 & 0.0160 \\ 0.0256 & 0.0216 & 0.0140 \\ 0.0160 & 0.0140 & 0.0109 \end{pmatrix} \begin{pmatrix} 0.7167 \\ 0.5833 \\ 0.3822 \end{pmatrix} = 0.0622 \begin{pmatrix} 0.7167 \\ 0.5833 \\ 0.3822 \end{pmatrix}$$

$$\begin{pmatrix} 0.0328 & 0.0256 & 0.0160 \\ 0.0256 & 0.0216 & 0.0140 \\ 0.0160 & 0.0140 & 0.0109 \end{pmatrix} \begin{pmatrix} -0.5509 \\ 0.1373 \\ 0.8232 \end{pmatrix} = 0.0025 \begin{pmatrix} -0.5509 \\ 0.1373 \\ 0.8232 \end{pmatrix}$$

$$\begin{pmatrix} 0.0328 & 0.0256 & 0.0160 \\ 0.0256 & 0.0216 & 0.0140 \\ 0.0160 & 0.0140 & 0.0109 \end{pmatrix} \begin{pmatrix} -0.4277 \\ 0.8005 \\ -0.4198 \end{pmatrix} = 0.0006 \begin{pmatrix} -0.4277 \\ 0.8005 \\ -0.4198 \end{pmatrix}$$

and can therefore be diagonalized as

$$\begin{pmatrix} 0.0328 & 0.0256 & 0.0160 \\ 0.0256 & 0.0216 & 0.0140 \\ 0.0160 & 0.0140 & 0.0109 \end{pmatrix} = C = U \cdot D \cdot U^\mathsf{T} =$$

$$\begin{pmatrix} 0.7167 & -0.5509 & -0.4277 \\ 0.5833 & 0.1373 & 0.8005 \\ 0.3822 & 0.8232 & -0.4198 \end{pmatrix} \begin{pmatrix} 0.0622 & 0 & 0 \\ 0 & 0.0025 & 0 \\ 0 & 0 & 0.0006 \end{pmatrix} \begin{pmatrix} 0.7167 & 0.5833 & 0.3822 \\ -0.5509 & 0.1373 & 0.8232 \\ -0.4277 & 0.8005 & -0.4198 \end{pmatrix}$$

(e.g. using MATLAB's singular-value decomposition function svd).
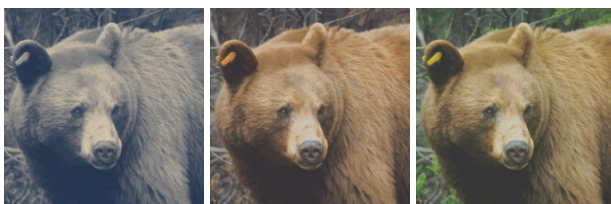
136

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

. . .
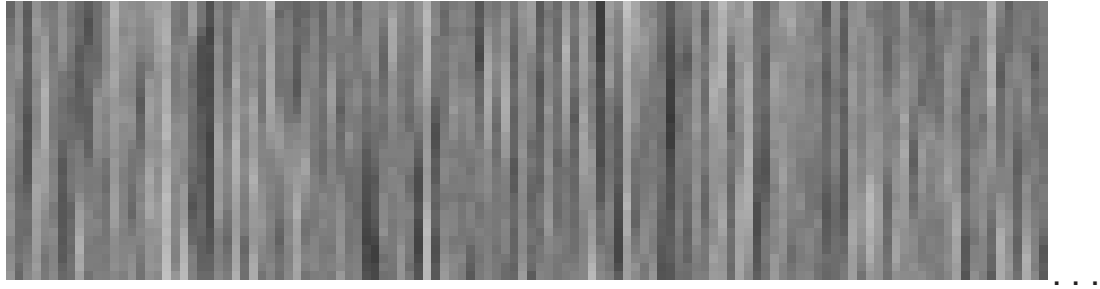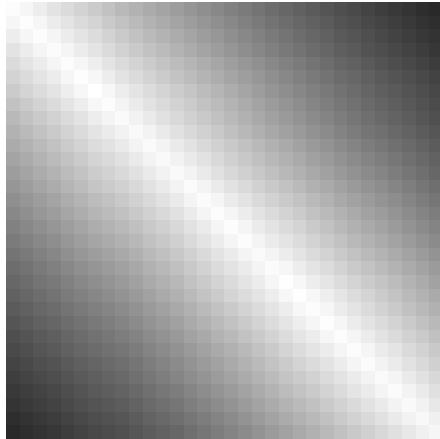
Covariance matrix C Matrix U with eigenvector columns
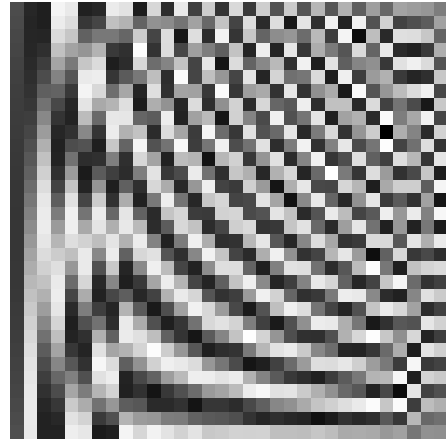
Matrix U' with normalised KLT eigenvector columns   Matrix with Discrete Cosine Transform base vector columns
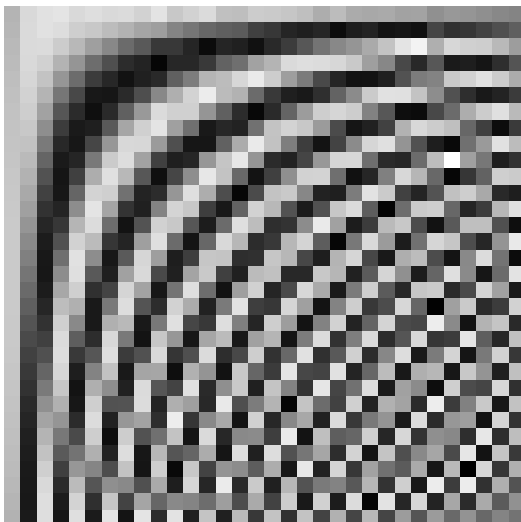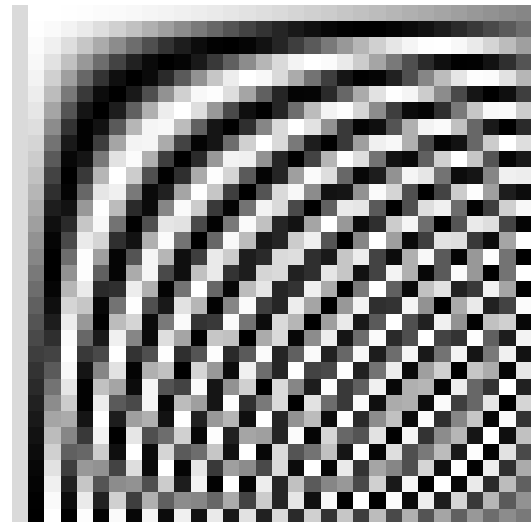
Breakthrough: Ahmed/Natarajan/Rao discovered the DCT as an excellent approximation of the KLT for typical photographic images, but far more efficient to calculate.

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

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

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

with

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

is an orthonormal transform:

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

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

2D Discrete Cosine Transform (log10)
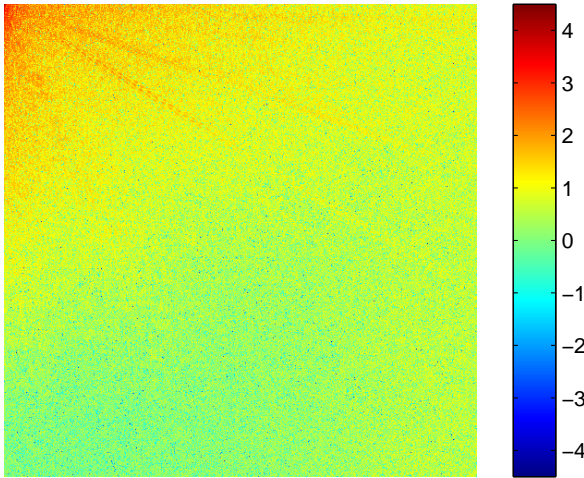
Original image

Whole-image DCT, 80% coefficient cutoff

80% truncated 2D DCT (log10)
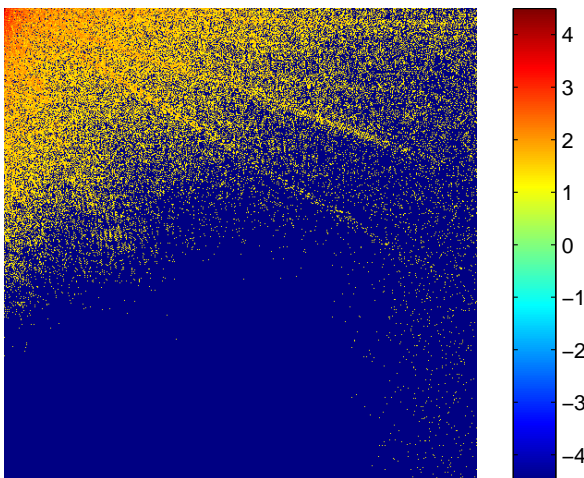
80% truncated DCT: reconstructed image

Whole-image DCT, 90% coefficient cutoff

90% truncated 2D DCT (log10)
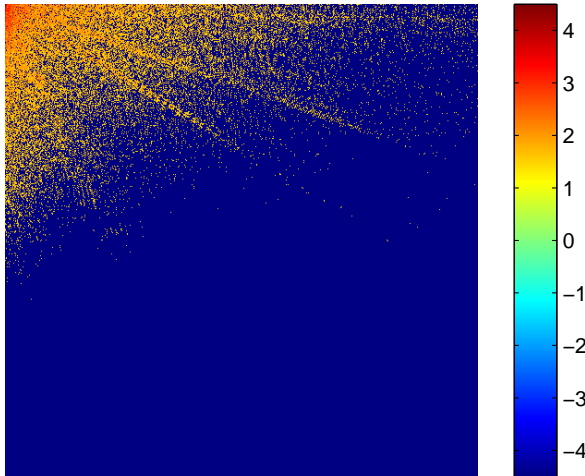
90% truncated DCT: reconstructed image

Whole-image DCT, 95% coefficient cutoff

95% truncated 2D DCT (log10)
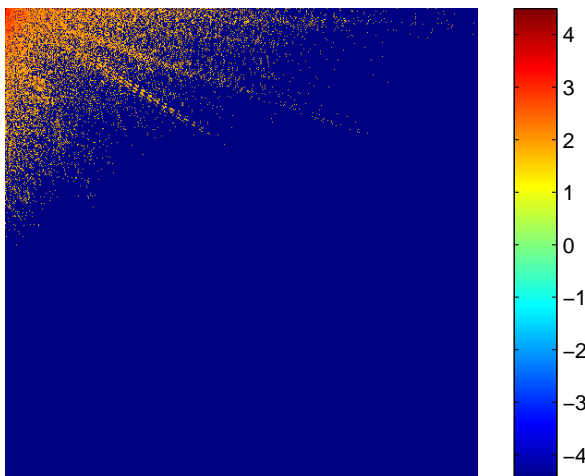
95% truncated DCT: reconstructed image

Whole-image DCT, 99% coefficient cutoff

99% truncated 2D DCT (log10)

99% truncated DCT: reconstructed image
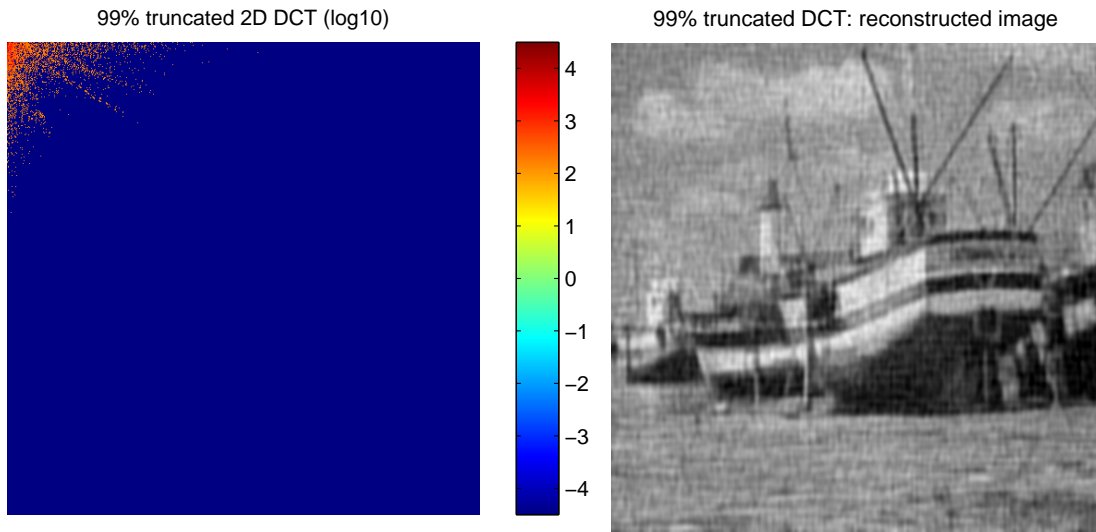
Base vectors of 8×8 DCT
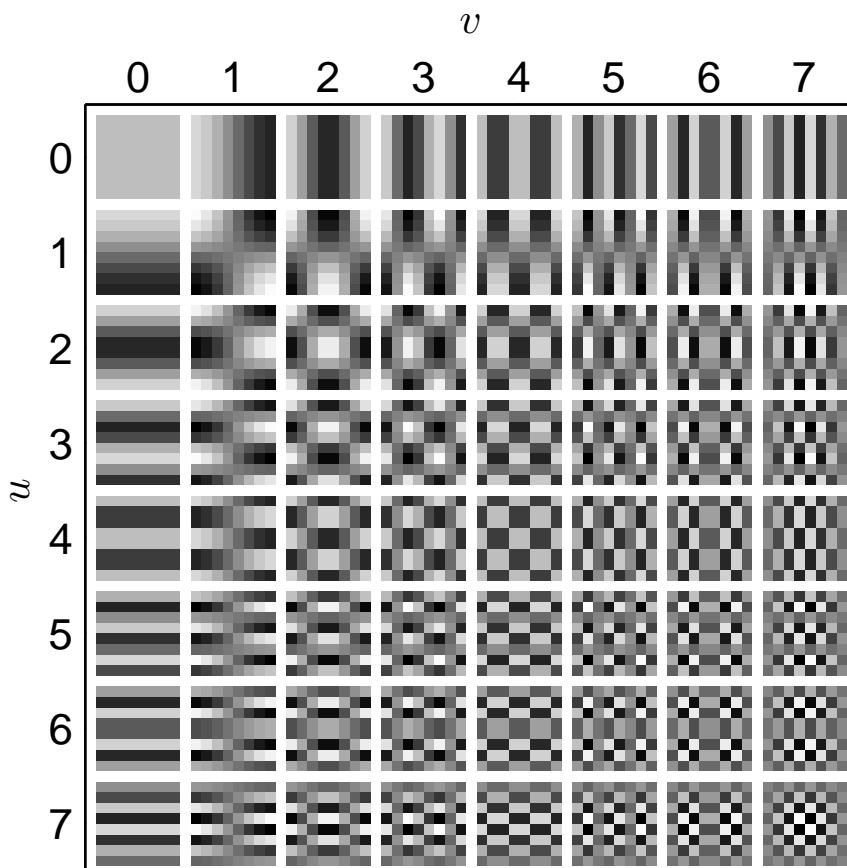
Discrete Wavelet Transform
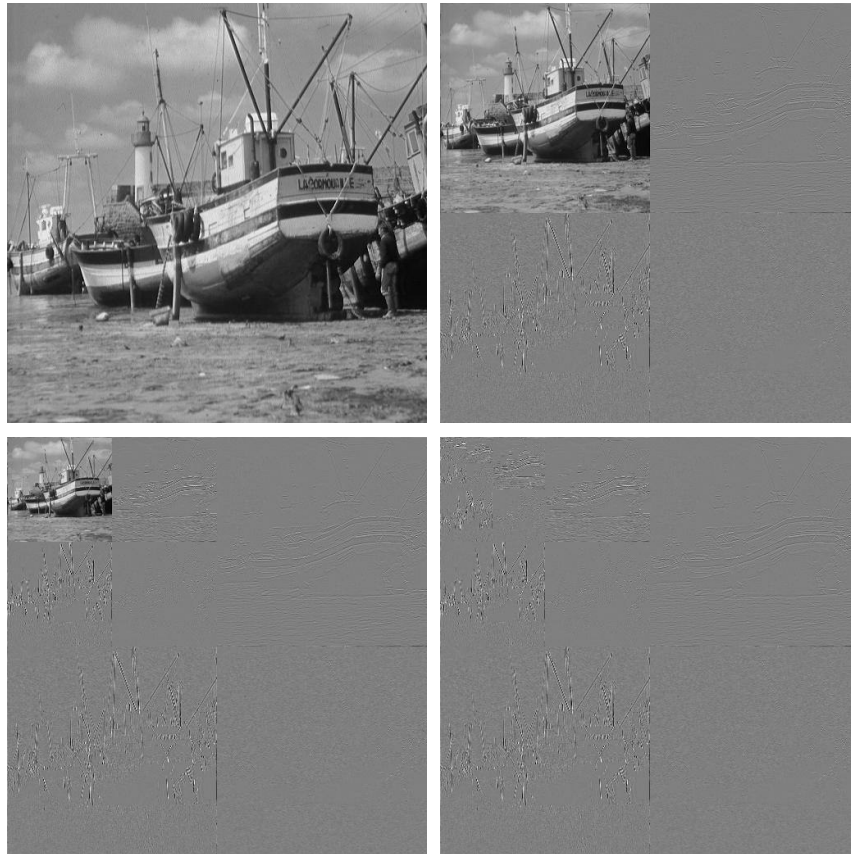
The n-point Discrete Fourier Transform (DFT) can be viewed as a device that sends an input signal through a bank of n non-overlapping band-pass filters, each reducing the bandwidth of the signal to $1/n$ of its original bandwidth.

According to the sampling theorem, after a reduction of the bandwidth by $1/n$, the number of samples needed to reconstruct the original signal can equally be reduced by $1/n$. The DFT splits a wide-band signal represented by n input signals into n separate narrow-band samples, each represented by a single sample.

A Discrete Wavelet Transform (DWT) can equally be viewed as such a frequency-band splitting device. However, with the DWT, the bandwidth of each output signal is proportional to the highest input frequency that it contains. High-frequency components are represented in output signals with a high bandwidth, and therefore a large number of samples. Low-frequency signals end up in output signals with low bandwidth, and are correspondingly represented with a low number of samples. As a result, high-frequency information is preserved with higher spatial resolution than low-frequency information.

Both the DFT and the DWT are linear orthogonal transforms that preserve all input information in their output without adding anything redundant.

As with the DFT, the 1-dimensional DWT can be extended to 2-D images by transforming both rows and columns (the order of which happens first is not relevant).

A DWT is defined by a combination of a low-pass filter, which smoothes a signal by allowing only the bottom half of all frequencies to pass through, and a high-pass filter, which preserves only the upper half of the spectrum. These two filters must be chosen to be "orthogonal" to each other, in the sense that together they split up the information content of their input signal without any mutual information in their outputs.

A widely used 1-D filter pair is DAUB4 (by Ingrid Daubechies). The low-pass filter convolves a signal with the 4-point sequence c_0, c_1, c_2, c_3, and the matching high-pass filter convolves with $c_3, -c_2, c_1, -c_0$. Written as a transformation matrix, DAUB4 has the form

$$
\begin{pmatrix}
c_0 & c_1 & c_2 & c_3 \\
c_3 & -c_2 & c_1 & -c_0 \\
 & & c_0 & c_1 & c_2 & c_3 \\
 & & c_3 & -c_2 & c_1 & -c_0 \\
\vdots & \vdots & & & & & \ddots \\
 & & & & & & c_0 & c_1 & c_2 & c_3 \\
 & & & & & & c_3 & -c_2 & c_1 & -c_0 \\
c_2 & c_3 & & & & & & & c_0 & c_1 \\
c_1 & -c_0 & & & & & & & c_3 & -c_2
\end{pmatrix}
$$

An orthogonal matrix multiplied with itself transposed is the identity matrix, which is fulfilled for the above one when

$$
\begin{aligned}
c_0^2 + c_1^2 + c_2^2 + c_3^2 &= 1 \\
c_2 c_0 + c_3 c_1 &= 0
\end{aligned}
$$

To determine four unknown variables we need four equations, therefore we demand that the high-pass filter will not pass through any information about polynomials of degree 1:

$$
\begin{aligned}
c_3 - c_2 + c_1 - c_0 &= 0 \\
0 c_3 - 1 c_2 + 2 c_1 - 3 c_0 &= 0
\end{aligned}
$$

This leads to the solution

$$
c_0 = (1 + \sqrt{3})/(4\sqrt{2}), \quad c_1 = (3 + \sqrt{3})/(4\sqrt{2})
$$
$$
c_2 = (3 - \sqrt{3})/(4\sqrt{2}), \quad c_3 = (1 - \sqrt{3})/(4\sqrt{2})
$$

Daubechies tabulated also similar filters with more coefficients.

In an n-point DWT, an input vector is convolved separately with a low-pass and a high-pass filter. The result are two output sequences of n numbers. But as each sequence has now only half the input bandwidth, each second value is redundant, can be reconstructed by interpolation with the same filter, and can therefore be discarded.

The remaining output values of the high-pass filter ("detail") are part of the final output of the DFT. The remaining values of the low-pass filter ("approximation") are recursively treated the same way, until they consist – after $\log_2 n$ steps – of only a single value, namely the average of the entire input.

Like with the DFT and DCT, for many real-world input signals, the DWT accumulates most energy into only a fraction of its output values. A commonly used approach for wavelet-based compression of signals is to replace all coefficients below an adjustable threshold with zero and encode only the values and positions of the remaining ones.

Discrete Wavelet Transform compression

80% truncated 2D DAUB8 DWT   90% truncated 2D DAUB8 DWT

95% truncated 2D DAUB8 DWT   99% truncated 2D DAUB8 DWT

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 ($\approx$150 BC).

Stevens' 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. Stevens observed that such rational scales ψ follow a power law:

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

Example coefficients a: temperature 1.6, weight 1.45, loudness 0.6, brightness 0.33.

Decibel

Communications engineers often use logarithmic units:

$\longrightarrow$ Quantities often vary over many orders of magnitude $\rightarrow$ difficult to agree on a common SI prefix

$\longrightarrow$ Quotient of quantities (amplification/attenuation) usually more interesting than difference

$\longrightarrow$ Signal strength usefully expressed as field quantity (voltage, current, pressure, etc.) or power, but quadratic relationship between these two ($P = U^2/R = I^2 R$) rather inconvenient

$\longrightarrow$ Weber/Fechner: perception is logarithmic

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.

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.

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 an additional letter after the "dB":

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

3 dB = double power, 6 dB = double pressure/voltage/etc.
10 dB = 10$\times$ power, 20 dB = 10$\times$ pressure/voltage/etc.

W.H. Martin: Decibel – the new name for the transmission unit. Bell System Technical Journal, January 1929.

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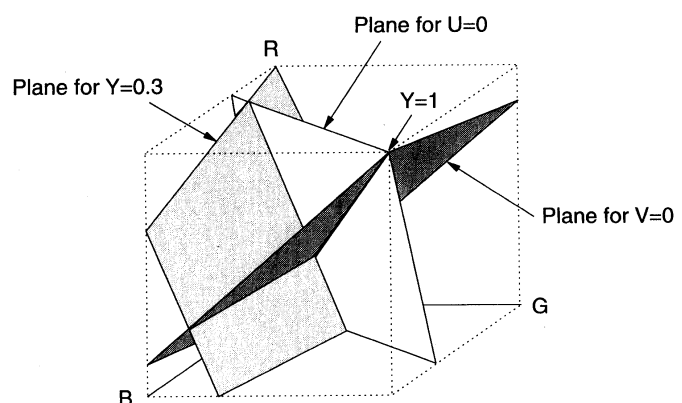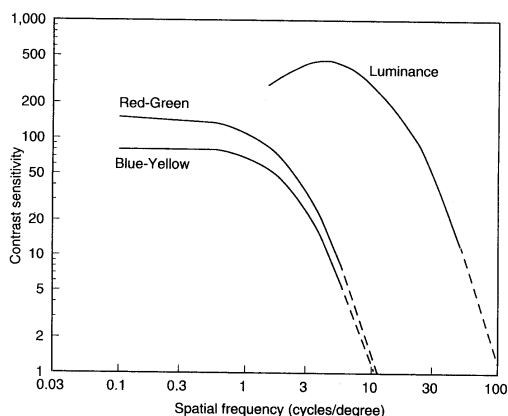
The human eye processes colour and luminosity at different resolutions. To exploit this phenomenon, many image transmission systems use a colour space with a luminance coordinate

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

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).
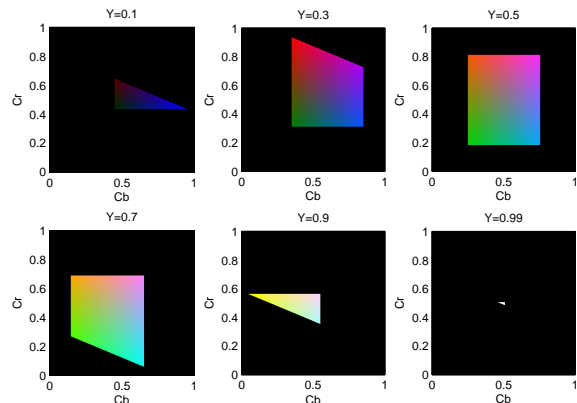
YCrCb video colour coordinates

Since $-0.7 \leq V \leq 0.7$ and $-0.9 \leq U \leq 0.9$, 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

$\longrightarrow$ frequency range 20–16000 Hz (babies: 20 kHz)

$\longrightarrow$ sound pressure range 0–140 dB$_{\text{SPL}}$ (about 10^{-5}–10^2 pascal)

$\longrightarrow$ mechanical filter bank (cochlea) splits input into frequency components, physiological equivalent of Fourier transform

$\longrightarrow$ most signal processing happens in the frequency domain where phase information is lost

$\longrightarrow$ some time-domain processing below 500 Hz and for directional hearing

$\longrightarrow$ sensitivity and difference limit are frequency dependent

Equiloudness curves and the unit "phon"
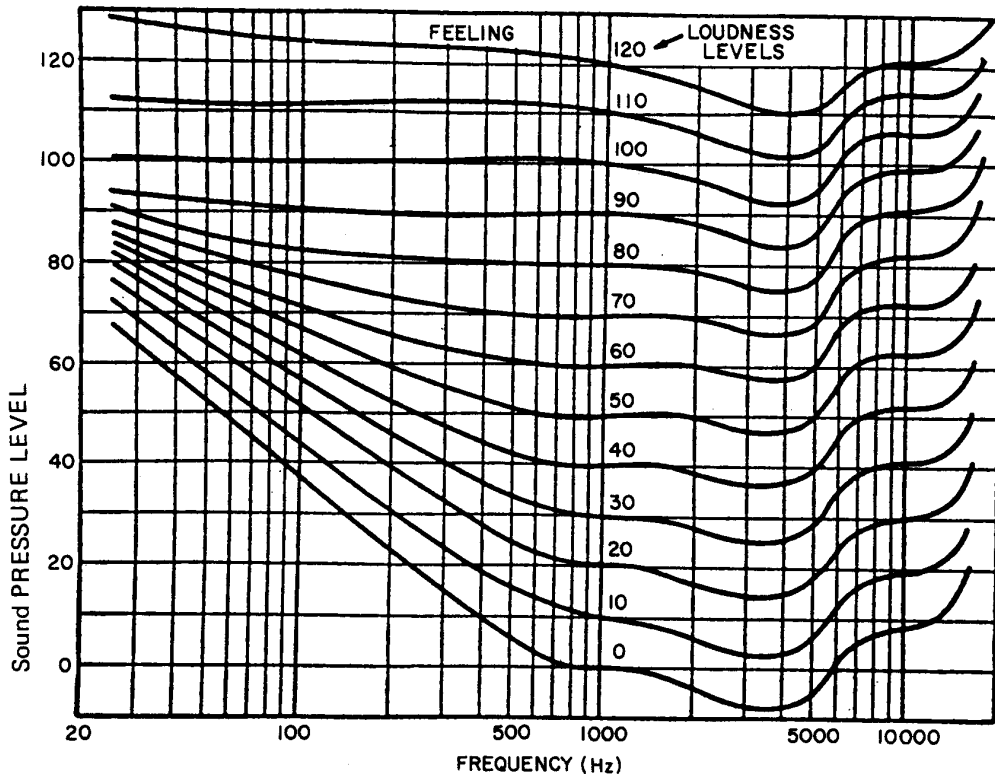
Each curve represents a loudness level in phon. At 1 kHz, the loudness unit phon is identical to dB_{SPL} and 0 phon is the sensation limit.
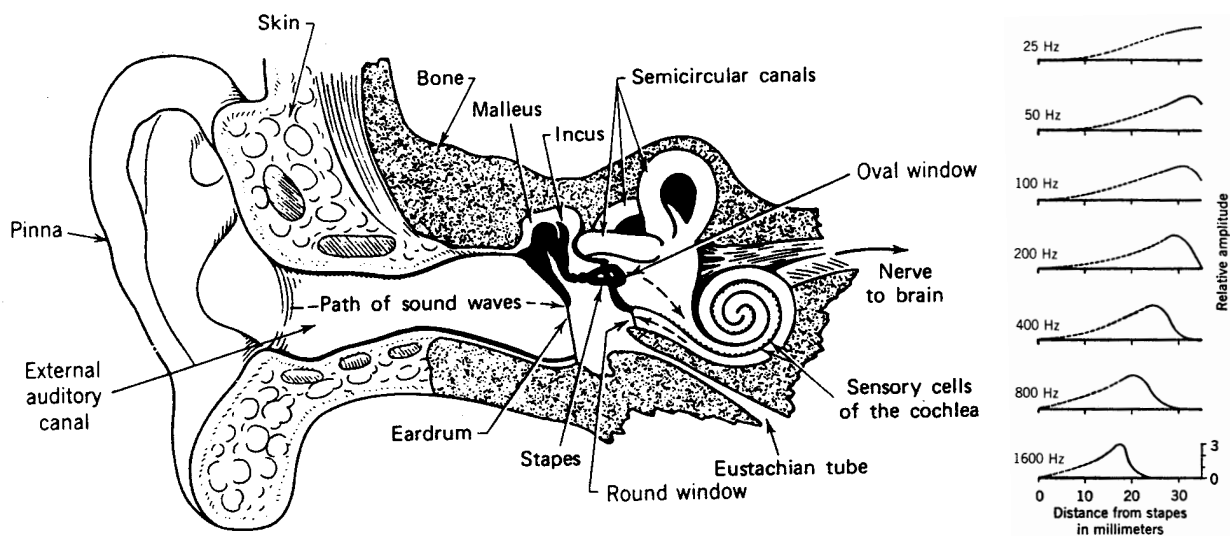
Sound waves cause vibration in the eardrum. The three smallest human bones in the middle ear (malleus, incus, stapes) provide an "impedance match" between air and liquid and conduct the sound via a second membrane, the oval window, to the cochlea. Its three chambers are rolled up into a spiral. The basilar 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

$\longrightarrow$ Louder tones increase the sensation limit for nearby frequencies and suppress the perception of quieter tones.

$\longrightarrow$ This increase is not symmetric. It extends about 3 barks to lower frequencies and 8 barks to higher ones.

$\longrightarrow$ 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.

$\longrightarrow$ Temporal masking: SL rises shortly before and after a masker.
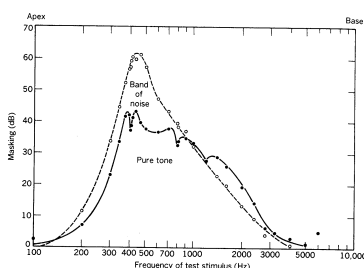
Level of masking in decibels for test tones of various frequencies presented with a masker of either an 80-dB SPL 400 Hz tone or an 80-dB SPL narrow-band noise (90 Hz) with a center frequency of 410 Hz. From Egan and Hake (11).
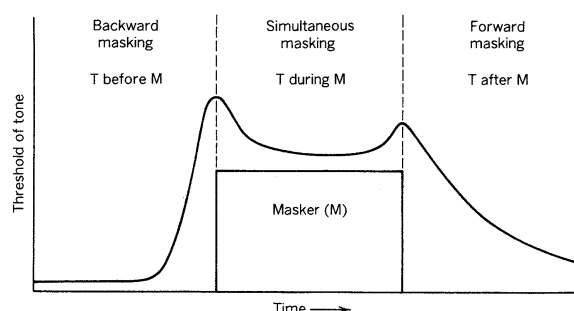
Audio demo: loudness and masking

loudness.wav

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

$\longrightarrow$ Sequence 1: tones have equal amplitude

$\longrightarrow$ 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
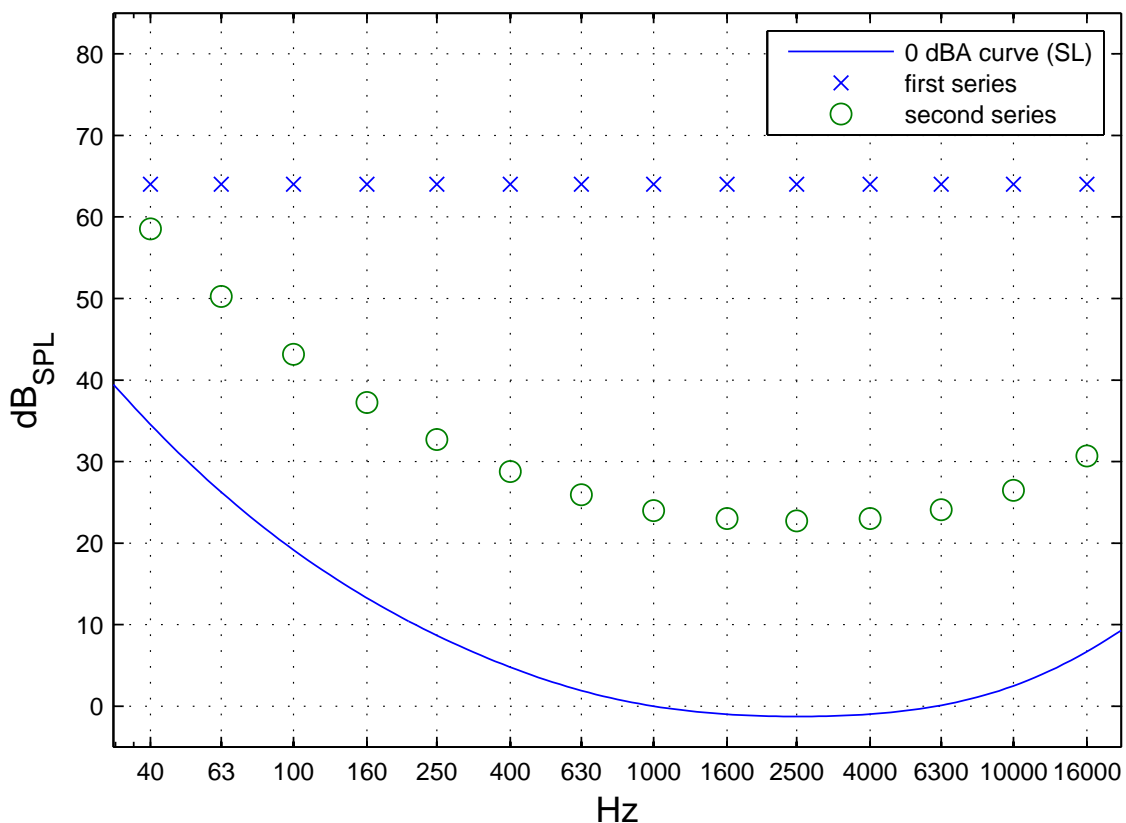
Audio demo: masking.wav
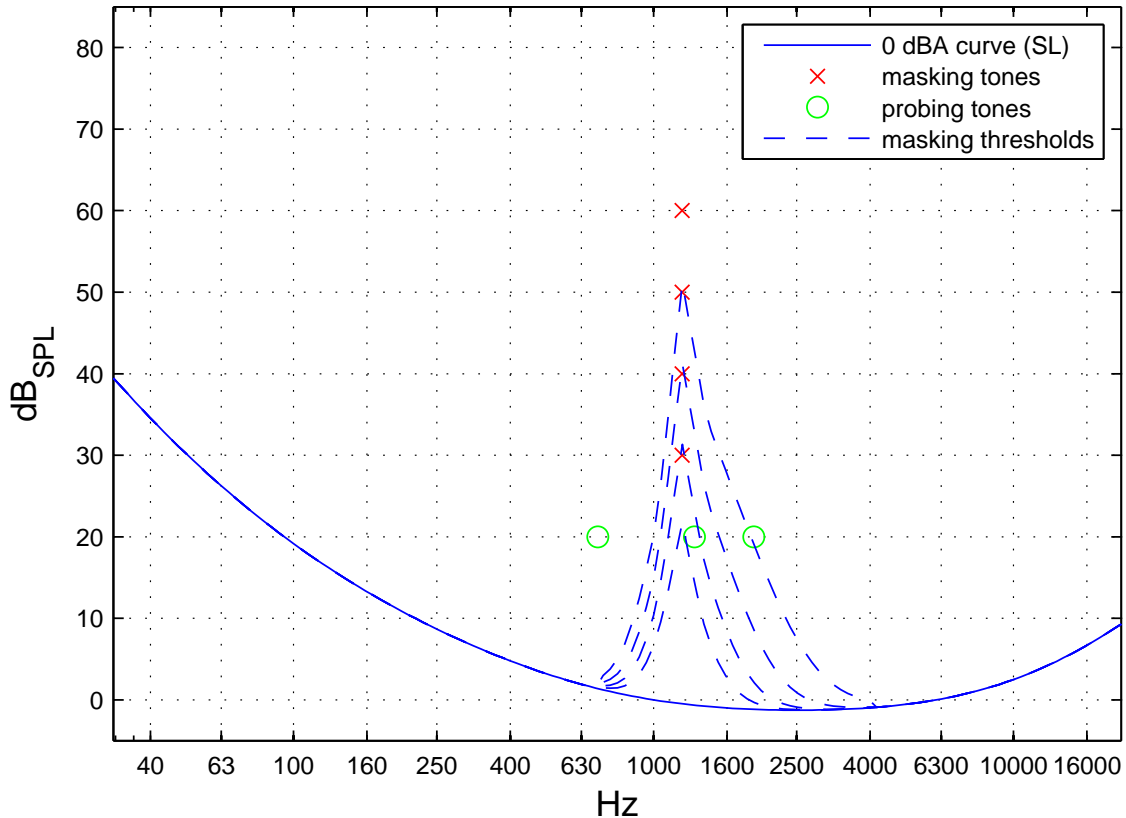
Quantization

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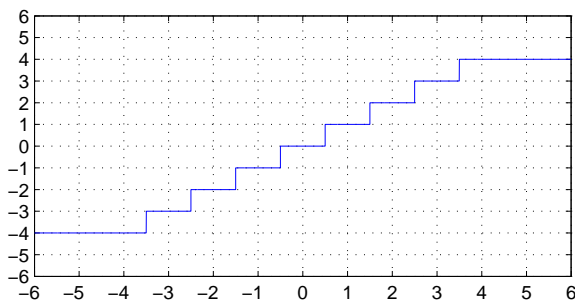
Non-uniform quantization:
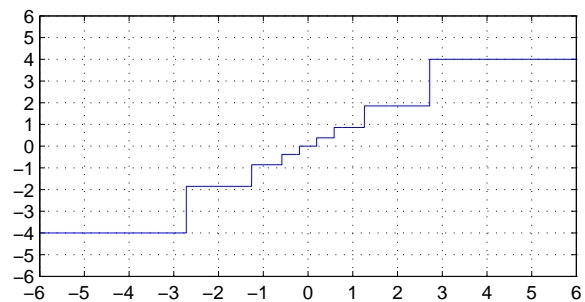
Quantization is the mapping from a continuous or large set of values (e.g., analog voltage, floating-point number) to a smaller set of (typically 2^8, 2^{12} or 2^{16}) values.

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.

173

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]

174
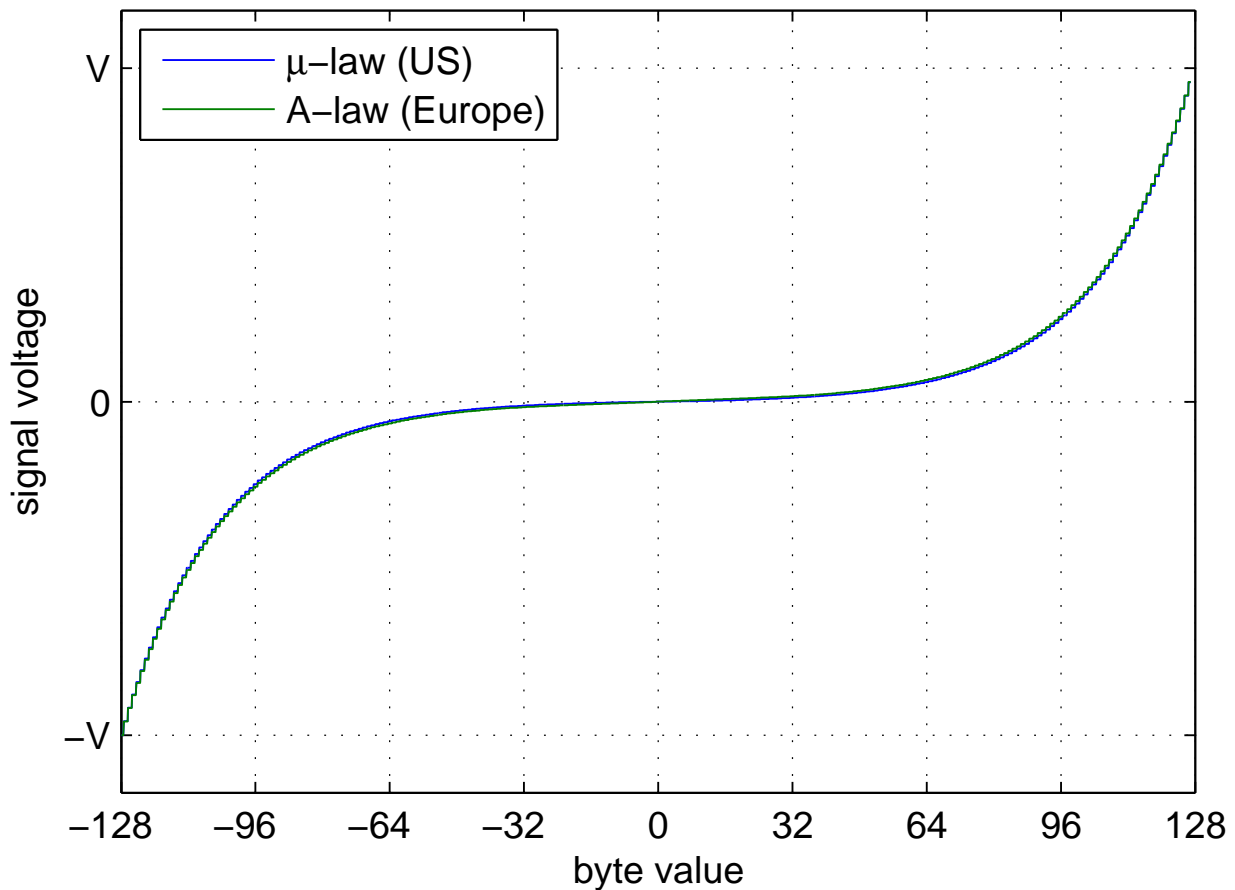
Joint Photographic Experts Group – JPEG

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:

$\longrightarrow$ continuous tone gray-scale and colour images

$\longrightarrow$ recognizable images at 0.083 bit/pixel

$\longrightarrow$ useful images at 0.25 bit/pixel

$\longrightarrow$ excellent image quality at 0.75 bit/pixel

$\longrightarrow$ indistinguishable images at 2.25 bit/pixel

$\longrightarrow$ feasibility of 64 kbit/s (ISDN fax) compression with late 1980s hardware (16 MHz Intel 80386).

$\longrightarrow$ 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:

$\longrightarrow$ Colour component transform: 8-bit RGB $\to$ 8-bit YCrCb

$\longrightarrow$ Reduce resolution of Cr and Cb by a factor 2

$\longrightarrow$ 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.

$\longrightarrow$ 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.

$\longrightarrow$ Apply the 8×8 forward DCT on each block

On unsigned 8-bit input, the resulting DCT coefficients will be signed 11-bit integers.

$\longrightarrow$ 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		17	18	24	47	99	99	99	99
12	12	14	19	26	58	60	55		18	21	26	66	99	99	99	99
14	13	16	24	40	57	69	56		24	26	56	99	99	99	99	99
14	17	22	29	51	87	80	62		47	66	99	99	99	99	99	99
18	22	37	56	68	109	103	77		99	99	99	99	99	99	99	99
24	35	55	64	81	104	113	92		99	99	99	99	99	99	99	99
49	64	78	87	103	121	120	101		99	99	99	99	99	99	99	99
72	92	95	98	112	100	103	99		99	99	99	99	99	99	99	99

$\longrightarrow$ apply DPCM coding to quantized DC coefficients from DCT

$\longrightarrow$ read remaining quantized values from DCT in zigzag pattern

$\longrightarrow$ locate sequences of zero coefficients (run-length coding)

$\longrightarrow$ apply Huffman coding on zero run-lengths and magnitude of AC values

$\longrightarrow$ add standard header with compression parameters

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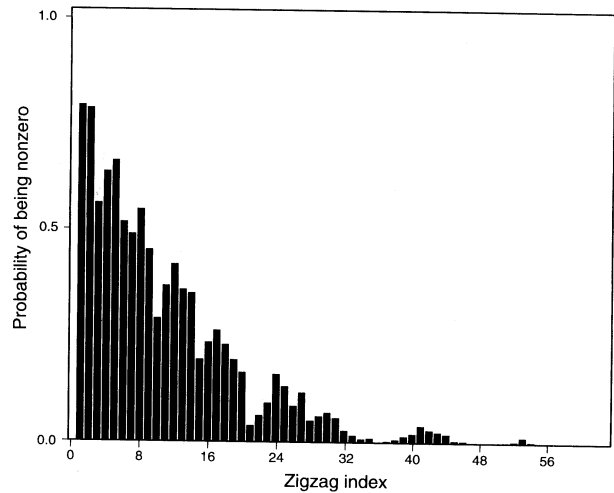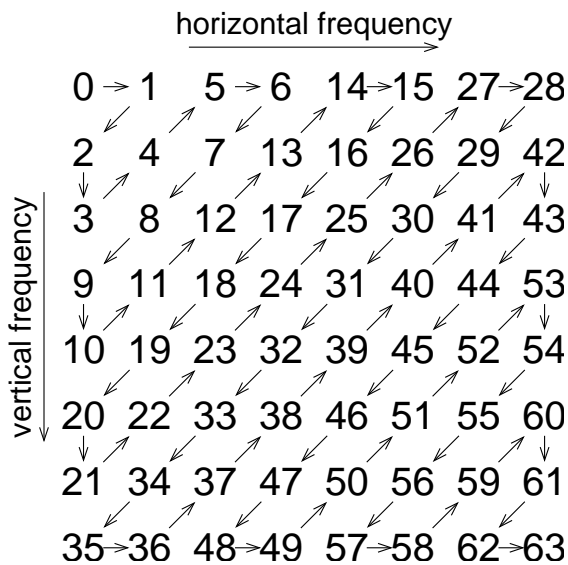
After the 8×8 coefficients produced by the discrete cosine transform have been quantized, the values are processed in the above zigzag order by a run-length encoding step.

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:

$$
\begin{aligned}
1: &\quad x = a \\
2: &\quad x = b \\
3: &\quad x = c \\
4: &\quad x = a + b - c \\
5: &\quad x = a + (b - c)/2 \\
6: &\quad x = b + (a - c)/2 \\
7: &\quad x = (a + b)/2
\end{aligned}
$$
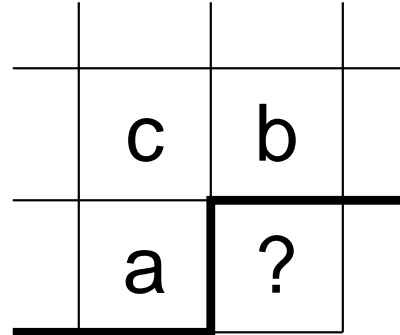
Predictor 1 is used for the top row, predictor 2 for the left-most row. The predictor used for the rest of the image is chosen in a header. The difference between the predicted and actual value is fed into either a Huffman or arithmetic coder.

Advanced JPEG features

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

$\longrightarrow$ 8 or 12 bits per pixel input resolution for DCT modes

$\longrightarrow$ 2–16 bits per pixel for lossless mode

$\longrightarrow$ 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

$\longrightarrow$ the transmission order of colour components, lines, as well as DCT coefficients and their bits can be interleaved in many ways

$\longrightarrow$ 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.

JPEG-2000 (JP2)

Processing steps:

$\longrightarrow$ Preprocessing: If pixel values are unsigned, subtract half of the maximum value $\rightarrow$ symmetric value range.

$\longrightarrow$ Colour transform: In lossy mode, use RGB $\leftrightarrow$ YCrCb.
In lossless mode, use RGB $\leftrightarrow$ YUV with integer approximation $Y = \lfloor (R + 2G + B)/4 \rfloor$.

$\longrightarrow$ Cut each colour plane of the image into tiles (optional), to be compressed independently, symmetric extension at edges.

$\longrightarrow$ Apply discrete wavelength transform to each tile, via recursive application (typically six steps) of a 2-channel uniformly maximally-decimated filter bank.
In lossy mode, use a 9-tap/7-tap real-valued filter (Daubechies), in lossless mode, use a 5-tap/3-tap integer-arithmetic filter.

$\longrightarrow$ Quantization of DWT coefficients (lossy mode only), same quantization step per subband.

$\longrightarrow$ Each subband is subdivided into rectangles (code blocks). These are split into bit planes and encoded with an adaptive arithmetic encoder (probability estimates based on 9 contexts).

For details of this complex multi-pass process, see D. Taubman: High-performance scalable image compression with EBCOT. IEEE Trans. Image Processing 9(7)1158–1170, July 2000. (On http://ieeexplore.ieee.org/)

$\longrightarrow$ The bit streams for the independently encoded code blocks are then truncated (lossy mode only), to achieve the required compression rate.

Features:

$\longrightarrow$ progressive recovery by fidelity or resolution

$\longrightarrow$ lower compression for specified region-of-interest

$\longrightarrow$ CrCb subsampling can be handled via DWT quantization

ISO 15444-1, example implementation: http://www.ece.uvic.ca/~mdadams/jasper/

JPEG examples (baseline DCT)

1:5 (1.6 bit/pixel)

1:10 (0.8 bit/pixel)

JPEG2000 examples (DWT)

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):

JPEG2000 examples (DWT)

1:20 (0.4 bit/pixel)

1:50 (0.16 bit/pixel)

Moving Pictures Experts Group – MPEG

$\longrightarrow$ MPEG-1: Coding of video and audio optimized for 1.5 Mbit/s ($1\times$ CD-ROM). ISO 11172 (1993).

$\longrightarrow$ 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).

$\longrightarrow$ MPEG-4: Adds algorithmic or segmented description of audio-visual objects for very-low bitrate applications. ISO 14496 (2001).

$\longrightarrow$ System layer multiplexes several audio and video streams, time stamp synchronization, buffer control.

$\longrightarrow$ Standard defines decoder semantics.

$\longrightarrow$ Asymmetric workload: Encoder needs significantly more computational power than decoder (for bit-rate adjustment, motion estimation, perceptual modeling, etc.)

http://www.chiariglione.org/mpeg/

MPEG video coding

$\longrightarrow$ Uses YCrCb colour transform, 8×8-pixel DCT, quantization, zigzag scan, run-length and Huffman encoding, similar to JPEG

$\longrightarrow$ the zigzag scan pattern is adapted to handle interlaced fields

$\longrightarrow$ Huffman coding with fixed code tables defined in the standard
MPEG has no arithmetic coder option.

$\longrightarrow$ adaptive quantization

$\longrightarrow$ SNR and spatially scalable coding (enables separate transmission of a moderate-quality video signal and an enhancement signal to reduce noise or improve resolution)

$\longrightarrow$ 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

time

backward
reference picture

current picture

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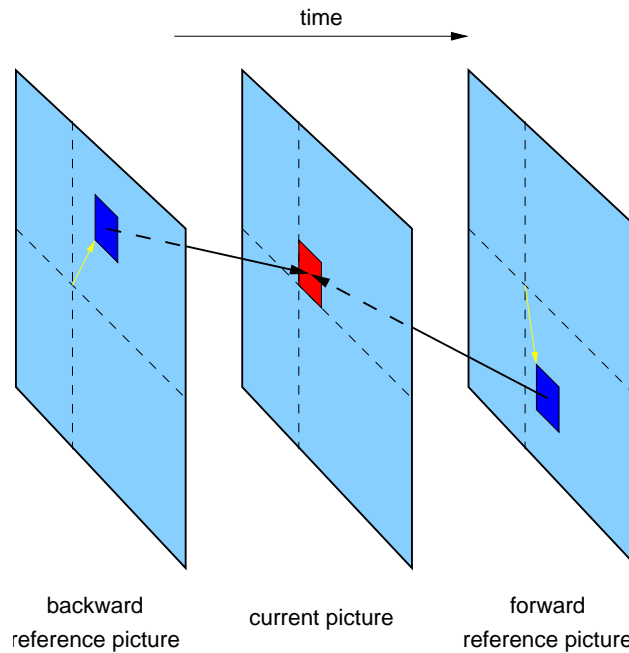
Each MPEG image is split into 16×16-pixel large *macroblocks*. The predictor forms a linear combination of the content of one or two other blocks of the same size in a preceding (and following) reference image. The relative positions of these reference blocks are encoded along with the differences.

MPEG reordering of reference images

Display order of frames:

time

I B B B P B B B P B B B P

Coding order:

time

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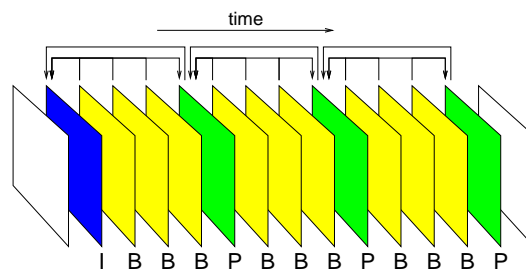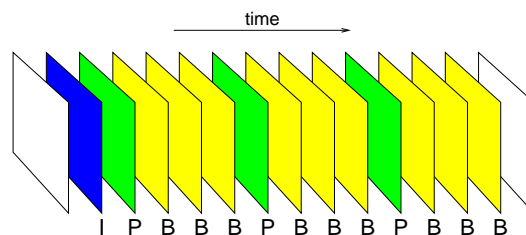
MPEG distinguishes between I-frames that encode an image independent of any others, P-frames that encode differences to a previous P- or I-frame, and B-frames that interpolate between the two neighbouring B- and/or I-frames. A frame has to be transmitted before the first B-frame that makes a forward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

MPEG system layer: buffer management

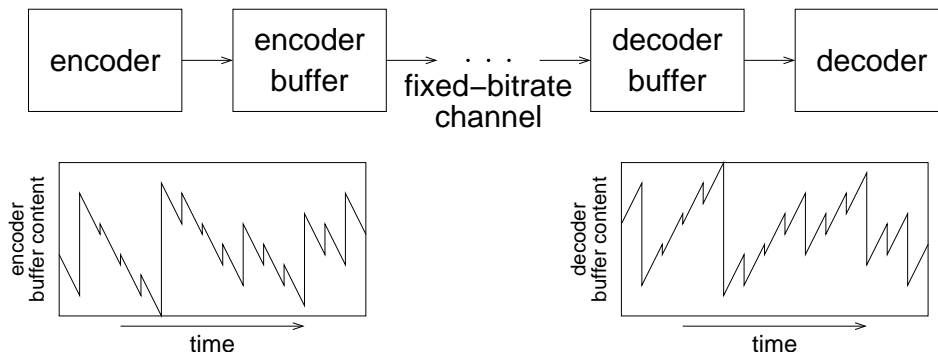
MPEG can be used both with variable-bitrate (e.g., file, DVD) and fixed-bitrate (e.g., ISDN) channels. The bitrate of the compressed data stream varies with the complexity of the input data and the current quantization values. Buffers match the short-term variability of the encoder bitrate with the channel bitrate. A control loop continuously adjusts the average bitrate via the quantization values to prevent under- or overflow of the buffer.

The MPEG system layer can interleave many audio and video streams in a single data stream. Buffers match the bitrate required by the codecs with the bitrate available in the multiplex and encoders can dynamically redistribute bitrate among different streams.

MPEG encoders implement a 27 MHz clock counter as a timing reference and add its value as a *system clock reference (SCR)* several times per second to the data stream. Decoders synchronize with a phase-locked loop their own 27 MHz clock with the incoming SCRs.

Each compressed frame is annotated with a *presentation time stamp (PTS)* that determines when its samples need to be output. *Decoding timestamps* specify when data needs to be available to the decoder.

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

$\longrightarrow$ Waveforms are split into segments of 384 samples each (8 ms at 48 kHz).

$\longrightarrow$ 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.

$\longrightarrow$ Each subband is then sampled at 1.5 kHz (for 48 kHz).

12 samples per window → again 384 samples for all 32 bands

$\longrightarrow$ 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.

$\longrightarrow$ 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

$\longrightarrow$ Modified DCT step decomposes subbands further into 18 or 6 frequencies

$\longrightarrow$ 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)

$\longrightarrow$ non-uniform quantization

$\longrightarrow$ Huffman entropy coding

$\longrightarrow$ buffer with short-term variable bitrate

$\longrightarrow$ 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:

$\longrightarrow$ Fourier transform for spectral analysis

$\longrightarrow$ Group the resulting frequencies into "critical bands" within which masking effects will not vary significantly

$\longrightarrow$ Distinguish tonal and non-tonal (noise-like) components

$\longrightarrow$ Apply masking function

$\longrightarrow$ Calculate threshold per subband

$\longrightarrow$ 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.

Exercise 13 Compare the quantization techniques used in the digital telephone network and in audio compact disks. Which factors to you think led to the choice of different techniques and parameters here?

Exercise 14 Which steps of the JPEG (DCT baseline) algorithm cause a loss of information? Distinguish between accidental loss due to rounding errors and information that is removed for a purpose.

Exercise 15 How can you rotate by multiples of $\pm 90°$ or mirror a DCT-JPEG compressed image without losing any further information. Why might the resulting JPEG file not have the exact same file length?

Exercise 16 Decompress this G3-fax encoded line:
1101011011110111011001101000000000000001

Exercise 17 You adjust the volume of your 16-bit linearly quantizing sound-card, such that you can just about hear a 1 kHz sine wave with a peak amplitude of 200. What peak amplitude do you expect will a 90 Hz sine wave need to have, to appear equally loud (assuming ideal headphones)?

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:

$\longrightarrow$ multirate systems

$\longrightarrow$ effects of rounding errors

$\longrightarrow$ adaptive filters

$\longrightarrow$ DSP hardware architectures

$\longrightarrow$ modulation and symbol detection techniques

$\longrightarrow$ sound effects

Your feedback on the course is very much appreciated:

http://www.cl.cam.ac.uk/cgi-bin/lr/login

If you find a typo or mistake in these lecture notes, please notify 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