DIGITAL SIGNAL & IMAGE PROCESSING

B Option – 8 lectures

Stephen Roberts
sjrob@robots.ox.ac.uk

Lecture 1 – Foundations

1.1 Recommended books

- Lynn. An introduction to the analysis and processing of signals. Macmillan.
- Oppenhein & Shafer. Digital signal processing. Prentice Hall
- Orfanidis Introduction to Signal Processing. Prentice Hall.
- Proakis & Manolakis. Digital Signal Processing: Principles, Algorithms and Applications.
- Matlab Signal processing toolbox manual.

1.2 Introduction

This course is ultimately concerned with the problem of computation, inference, manipulation and decision making using 1- and 2-dimensional data streams ('signals' and 'images'). The course starts by considering the foundations of modern signal processing theory, defining terms and offering a simple but profound framework under which data may be manipulated. Although theory is very important in this subject area, an effort is made to provide examples of the major points throughout the course.

1.3 Summary/Revision of basic definitions

For the sake of brevity in the *nomenclature* definitions and theorems are often going to be introduced in their 1-d form (i.e. as signals) with the index variable t or x. Please note that such an indexed set of samples is just a 1-d case of a generic ordered set which could be 2-d or more, i.e. dependent on a set of indices or variables, i, j or x, y for example. I try to keep to the convention that $f(x)$ is a 1-d system with 1-d frequency support indexed by u and $f(x, y)$ is a 2-d system with 2-d frequency support indexed by u, v.

1.3.1 Linear Systems

A linear system may be defined as one which obeys the **Principle of Superposition**. If $f_1(x)$ and $f_2(x)$ are inputs to a linear system which gives rise to outputs $r_1(x)$ and $r_2(x)$ respectively, then the combined input $af_1(x) + bf_2(x)$ will give rise to an output $ar_1(x) + br_2(x)$, where a and b are arbitrary constants.

Notes

- If we represent an input signal by some support in a frequency domain, $\mathcal{F}_{in}$ (i.e. the set of frequencies present in the input) then no new frequency support will be required to model the output, i.e.

$$\mathcal{F}_{out} \subseteq \mathcal{F}_{in}$$

- Linear systems can be broken down into simpler sub-systems which can be re-arranged in any order, i.e.

$$f \longrightarrow g_1 \longrightarrow g_2 \equiv f \longrightarrow g_2 \longrightarrow g_1 \equiv f \longrightarrow g_{1,2}$$

1.3.2 Time or space Invariance

A time-invariant system is one whose properties do not vary with time (i.e. the input signals are treated the same way regardless of their time of arrival); for example, with discrete systems, if an input sequence $f(x)$ produces an output sequence $r(x)$, then the input sequence $f(x - x_o)$ will produce the output sequence $r(x - x_o)$ for all x_o. In the case of 2-d images, the system response does not depend on the position within the image, i.e. not on x_o, y_o.

1.4 Linear Processes

Some of the common signal processing functions are amplification (or attenuation), mixing (the addition of two or more signal waveforms) or un-mixing and filtering. Each of these can be represented by a linear time-invariant "block" with an input-output characteristic which can be defined by:

- The *impulse response* $g(x)$ in the *time domain*.
- The *transfer function* $G(u)$ in a *frequency domain*. We will see that the choice of frequency basis may be subtly different from time to time.

As we will see, there is (for many of the systems we examine in this course) an *invertable* mapping between the time (image index) and (spatial) frequency domain representations.

1.5 Convolution

Convolution allows the evaluation of the output signal from a LTI system, given its *impulse response* and input signal.

The input signal can be considered as being composed of a succession of impulse functions, each of which generates a weighted version of the impulse response at the output, as shown in 1.1.

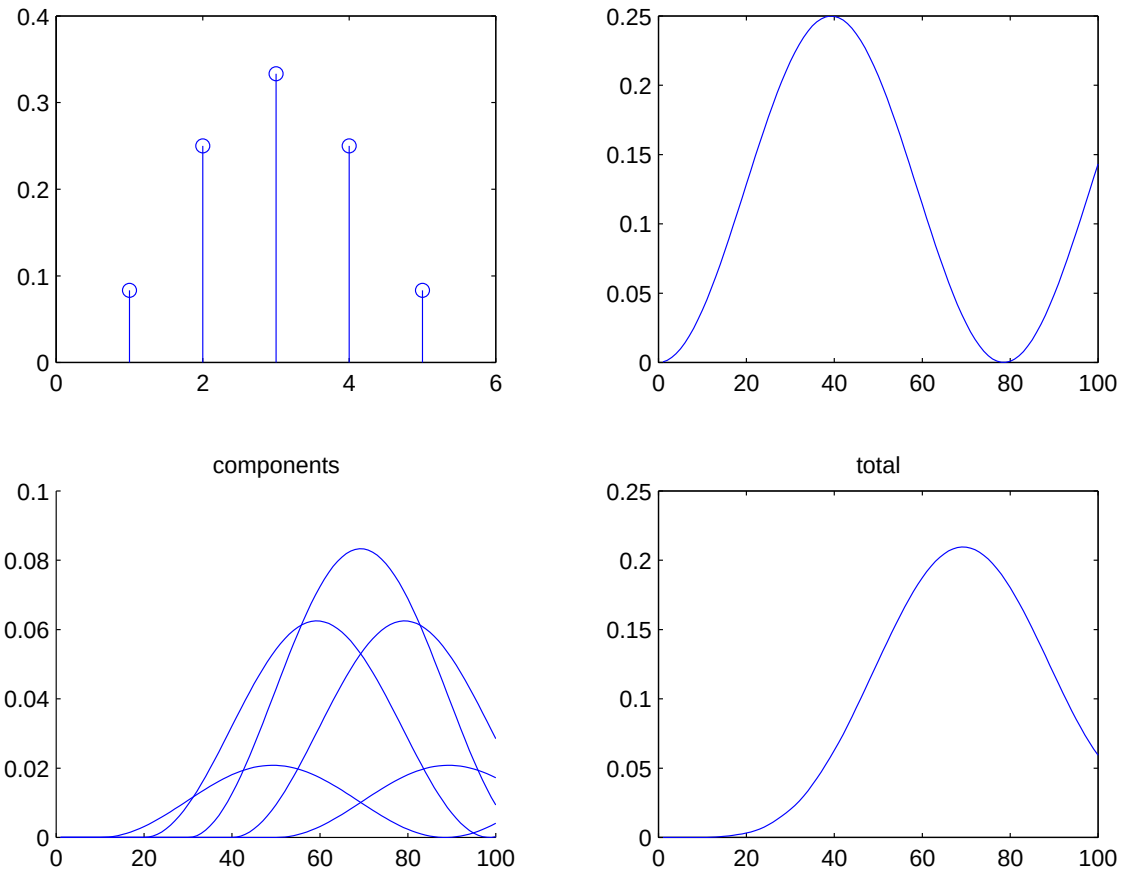


Figure 1.1: *Convolution as a summation over shifted impulse responses.*

The output at time x, $r(x)$, is obtained simply by adding the effect of each separate impulse function – this gives rise to the *convolution integral*:

$$r(x) = \sum_{\tau} \{f(x-\tau)d\tau\} g(\tau) \xrightarrow{d\tau \to 0} \int_0^{\infty} f(x-\tau)g(\tau)d\tau$$

τ is a dummy variable which represents time measured "back into the past" from the instant x at which the output $r(x)$ is to be calculated.

1.5.1 Notes

- Convolution is commutative. Thus:

$$r(x) \text{ is also } \int_0^{\infty} f(\tau)g(x-\tau)d\tau$$

- For discrete systems convolution is a summation operation:

$$r[n] = \sum_{k=0}^{\infty} f[k]g[n-k] = \sum_{k=0}^{\infty} f[n-k]g[k]$$

1.6 Frequency-Domain Analysis

Linear (time-invariant) systems, by definition, may be represented (in the continuous case) by linear differential equations (in the discrete case by linear *difference* equations). Consider the application of the linear differential operator, $\mathcal{D} = d/dx$, to the function $f(x) = e^{sx}$:

$$\mathcal{D}f(x) = sf(x)$$

An equation of this form means that $f(x)$ is the *eigenfunction* of $\mathcal{D}$. Just like the eigen analysis you know from matrix theory, this means that $f(x)$ and any linear operation on $f(x)$ may be represented using a set of functions of exponential form, and that this function may be chosen to be *orthogonal*. This naturally gives rise to the use of the *Laplace* and *Fourier* representations.

- **The Laplace transform**:

$$F(s) \longrightarrow \text{Transfer function } G(s) \longrightarrow R(s)$$

where,

$$F(s) = \int_0^\infty f(x)e^{-sx}dx \qquad \text{Laplace transform of } f(x)$$

$$R(s) = G(s)F(s)$$

where $G(s)$ can be expressed as a *pole-zero* representation of the form:

$$G(s) = \frac{A(s - z_1) \ldots (s - z_m)}{(s - p_1)(s - p_2) \ldots (s - p_n)}$$

- **The Fourier transform**:

$$F(u) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi ux} dx \quad \text{Fourier transform of } f(x)$$

and

$$R(iu) = G(iu)F(iu)$$

The output time function can be obtained by taking the inverse Fourier transform:

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(u) e^{i2\pi ux} du$$

Theorem

Convolution in the time domain is equivalent to multiplication in the frequency domain i.e.

$$r(x) = g(x) * f(x) \equiv \mathcal{F}^{-1}\{R(u) = G(u)F(u)\}$$

and

$$r(x) = g(x) * f(x) \equiv \mathcal{L}^{-1}\{R(s) = G(s)F(s)\}$$

Proof

Consider the general integral (Laplace) transform of a shifted function:

$$\begin{aligned} \mathcal{L}\{f(x-\tau)\} &= \int_x f(x-\tau)e^{-sx}dx \\ &= e^{-s\tau}\mathcal{L}\{f(x)\} \end{aligned}$$

Now consider the Laplace transform of the convolution integral

$$\begin{aligned} \mathcal{L}\{f(x) * g(x)\} &= \int_x \int_\tau f(x-\tau)g(\tau)d\tau e^{-sx}dx \\ &= \int_\tau g(\tau)e^{-s\tau}d\tau\mathcal{L}\{f(x)\} \\ &= \mathcal{L}\{g(x)\}\mathcal{L}\{f(x)\} \end{aligned}$$

By allowing $s \to iu$ we prove the result for the Fourier transform as well.

1.7 Simple filtering - basics

It is very useful in image and signal processing to view transformations as *input-output* relationships, specified by a *transfer function*.

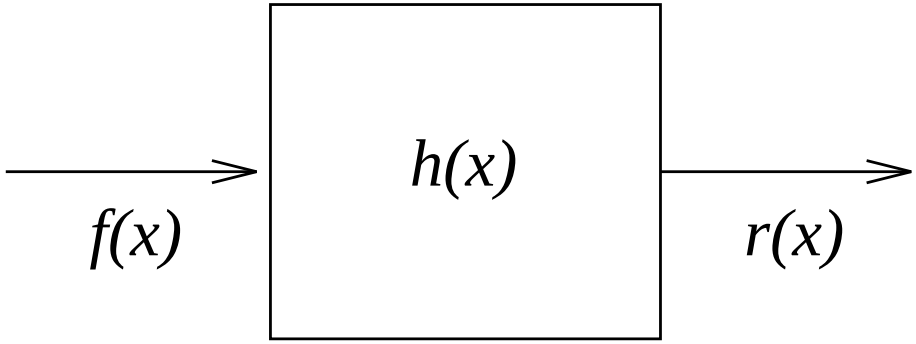


Figure 1.2: *Transfer function model.*

The transfer function is defined by the *impulse response* of the system. This specifies the response of the system to an impulse input (a dirac). The convolution theorem states that the response, $r(x)$, of such an impulse input, $i(x)$, may be given as

$$r(x) = i(x) * h(x)$$

where $h(x)$ is the impulse response function. And for an arbitrary input, $f(x)$

$$r(x) = f(x) * h(x)$$

or, by taking the FT

$$R(u) = F(u)H(u)$$

where $H(u)$ is the transfer function in the Fourier domain. As $F(u)$ represents the spectrum of $f(x)$, so multiplying by some function $H(u)$ may be regarded as modifying or *filtering* $F(u)$ and hence filtering the data sequence $f(x)$.

Key point : Filtering may be regarded as a *multiplication* in the spectral domain, or as a *convolution* in the image/signal domain.

The Ideal LPF (ILPF) : Consists of $H(u, v) = 1$ if $\sqrt{(u^2 + v^2)} \leq D$ and 0 otherwise. As the ILPF is symmetric about the origin in Fourier space, it requires that the FT of the image is also centred on the origin.

The Butterworth LPF (BLPF) : Consists of

$$H(u, v) = \left\{ 1 + \left[\frac{D(u, v)}{D} \right]^{2n} \right\}^{-1}$$

where $D(u, v) = \sqrt{(u^2 + v^2)}$ and D and n are filter settings **The IHPF** : Is the inverse transfer function of the ILPF, and the corresponding Butterworth filter, the **BHPF**, as the inverse of the BLPF.

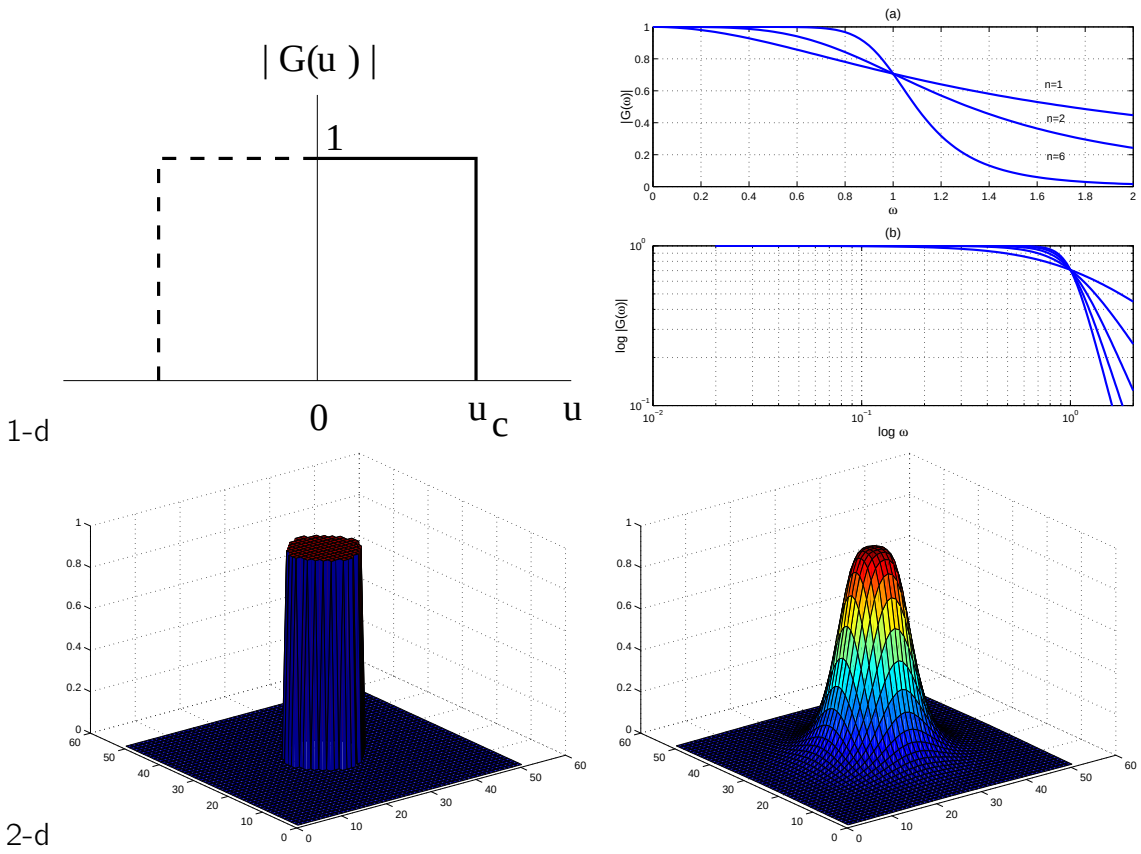


Figure 1.3: *The ILPF and the BLPF*

Lecture 2 – The Fourier Domain & Digital Filters

2.1 The Fourier transform

Consider again the 1-D case of a signal $f(x)$, the FT is defined as

$$F(u) = \int_{-\infty}^{+\infty} f(x) \exp[-i2\pi ux] dx$$

and the inverse as

$$f(x) = \int_{-\infty}^{\infty} F(u) e^{i2\pi ux} du$$

which form a *Fourier transform pair*. We note that the FT is, in general, a complex function of the form $F(u) = \mathbb{R}(u) + i\mathbb{I}(u)$. We call $|F(u)|$ the *Fourier spectrum* of $f(x)$ and $\phi(u) = \tan^{-1}[\mathbb{I}(u)/\mathbb{R}(u)]$ the *phase spectrum*.

2.1.1 2-D Fourier transform

There is no inherent change in theory for the 2-dimensional case, where $f(x, y)$ exists, so the FT, $F(u, v)$ is given as

$$F(u, v) = \int \int_{-\infty}^{+\infty} f(x, y) \exp[-i2\pi(ux + vy)] dx dy$$

and the inverse as

$$f(x, y) = \int \int_{-\infty}^{+\infty} F(u, v) \exp[i2\pi(ux + vy)] du dv$$

We note that the above are *separable*

2.1.2 Some basic theorems

Here are some basic (and useful) theorems related to the FT. They are shown for a 1-D system, for ease of reading and notation, and directly translate into higher dimensions as above.

- *Similarity theorem* : if $f(x) \rightarrow F(u)$ then $f(ax) \rightarrow \frac{1}{|a|} F(u/a)$
- *Addition theorem* : if $f(x), g(x) \rightarrow F(u), G(u)$ then $af(x) + bg(x) \rightarrow aF(u) + bG(u)$

- *Shift or twist theorem* : if $f(x) \rightarrow F(u)$ then $f(x-a) \rightarrow \exp[-i2\pi ua]F(u)$
- *Convolution theorem* : if

$$f(x) * g(x) = \int f(\tau)g(x-\tau)d\tau$$

  then $FT[f(x) * g(x)] = F(u)G(u)$. Note that this is of great use in filtering
- *Power theorem* :

$$\int |f(x)|^2 dx = \int |F(u)|^2 du$$

  i.e. a statement about conservation of energy.
- *Derivative theorem* : if $f(x) \rightarrow F(u)$ then $f'(x) = d/dx[f(x)] \rightarrow iuF(u)$

2.1.3 The discrete FT (DFT)

We sample the continuous (start with 1-D) function, $f(x)$, at M points spaced Δx apart. We now describe the function as

$$f(x) = f(x_o + x\Delta x)$$

where x now describes an *index*, with this transformation, u the Fourier varable paired to x is discretised into M points. We thus obtain :

$$F(u) = \frac{1}{M} \sum_{x=0}^{M-1} f(x) \exp[-i2\pi ux/M]$$

and

$$f(x) = \sum_{u=0}^{M-1} F(u) \exp[i2\pi ux/M]$$

and, for 2-D systems

$$F(u,v) = \frac{1}{M}\frac{1}{N} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) \exp[-i2\pi(ux/M + vy/N)]$$

and

$$f(x,y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v) \exp[i2\pi(ux/M + vy/N)]$$

if y is sampled evenly at N sample points.

The sampling in the space domain, $\Delta x, \Delta y$ corresponds to a sampling in the 'frequency' domain of

$$\Delta u = \frac{1}{M\Delta x} \qquad \Delta v = \frac{1}{M\Delta y}$$

2.1.4 Some useful results using the DFT

- The total width of the samples in the x, y directions determines the lowest spatial frequency we can resolve, $u_{min} = 1/(M\Delta x)$
- The sample interval, $\Delta x, \Delta y$, dictates the highest spatial frequency we can resolve, $u_{max} = 1/(2\Delta x)$
- The number of samples, M, N, dictates the number of spatial frequency 'bins' that can be resolved.
- *Addition & linearity* : as with continuous functions
- *Shift theorem* : : if $f(x) \rightarrow F(u)$ then $f(x-a) \rightarrow \exp[-i2\pi ua/M]F(u)$, hence

$$f(x,y)\exp[i2\pi(u_o x + v_o y)/M] \rightarrow F(u-u_o, v-v_o)$$

and

$$f(x-x_o, y-y_o) \rightarrow F(u,v)\exp[-i2\pi(ux_o + vy_o)/M]$$

if we let $u_o = v_o = M/2$ then we can shift the frequency space to the centre of the frequency square

$$f(x,y)(-1)^{x+y} \rightarrow F(u-M/2, v-M/2)$$

- *Discrete convolution* :

$$f(x) * g(x) = \sum_{m=0}^{M-1} f(m)g(x-m)$$

and

$$DFT[f(x) * g(x)] = MF(u)G(u)$$

- *Power theorem* :

$$\sum_{x=0}^{M-1} |f(x)|^2 = M \sum_{u=0}^{M-1} |F(u)|^2$$

- *Periodicity* :

$$F(u,v) = F(u+M,v) = F(u,v+M) = F(u+M,v+M)$$

Note that this leads us to deduce the *aliasing theorem*

- *Rotation* :

$$f(r,\theta+\theta_o) \rightarrow F(\omega,\phi+\theta_o)$$

- *Average value* : If we define the average value of the 2-D function as

$$\overline{f}(x,y) = \frac{1}{M^2} \sum_{x=0}^{M-1} \sum_{y=0}^{M-1} f(x,y) \qquad \text{then} \qquad \overline{f}(x,y) = F(0,0)$$

- *Laplacian* : The Laplacian of a 2-D variable is defined as

$$\nabla^2 f(x, y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

The DFT of the above is hence

$$-(2\pi)^2(u^2 + v^2)F(u, v)$$

(each time we take a derivative we get a factor of $i2\pi$)

Trick : Plots of $|F(u, v)|$ often decay very rapidly from a central peak, so it is good to display on a log scale. Often the transform, $F'(u, v) = \log[1+|F(u, v)|]$ is used.

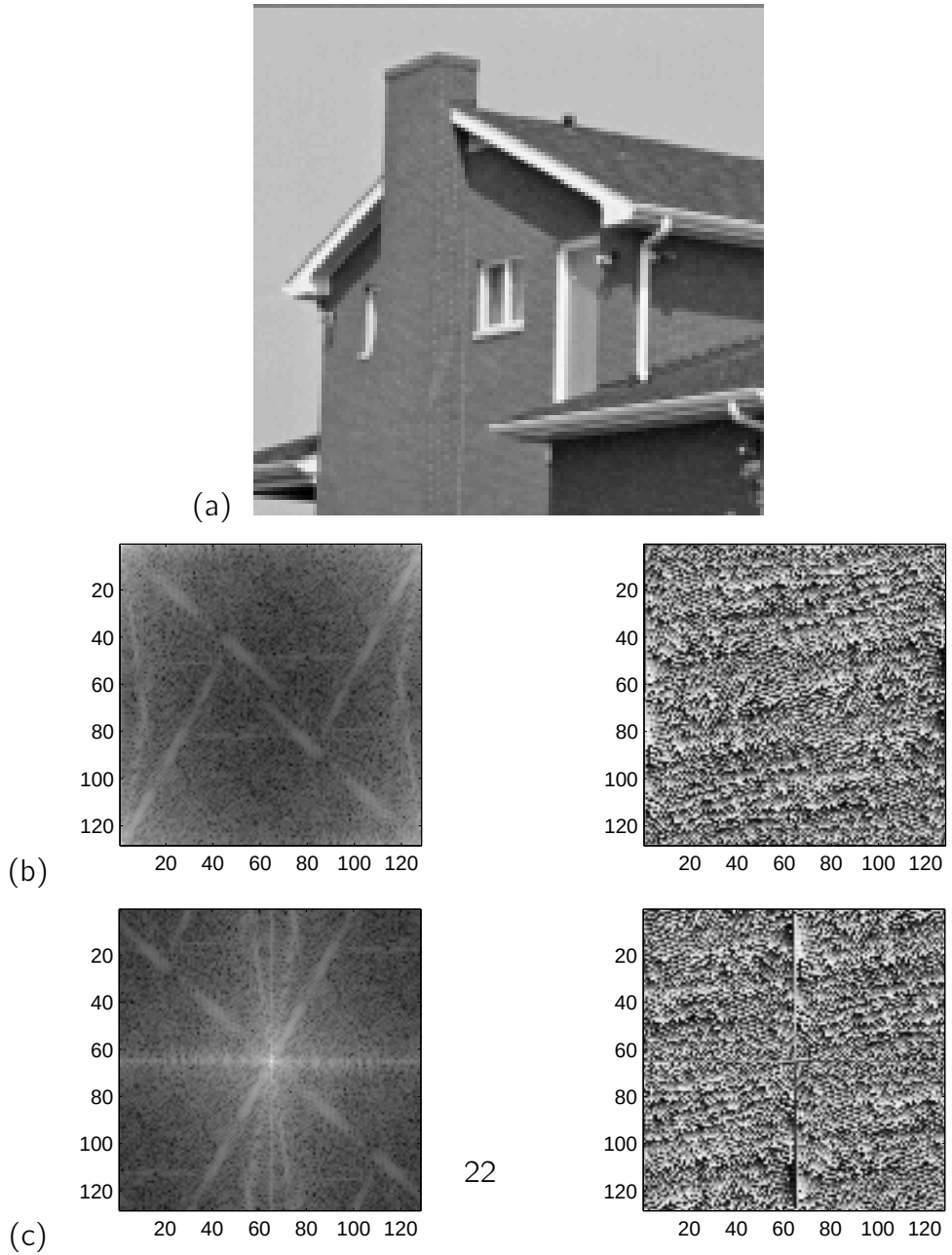


Figure 2.1: *Image (a), Fourier spectrum (b) and shifted Fourier spectrum (c).*

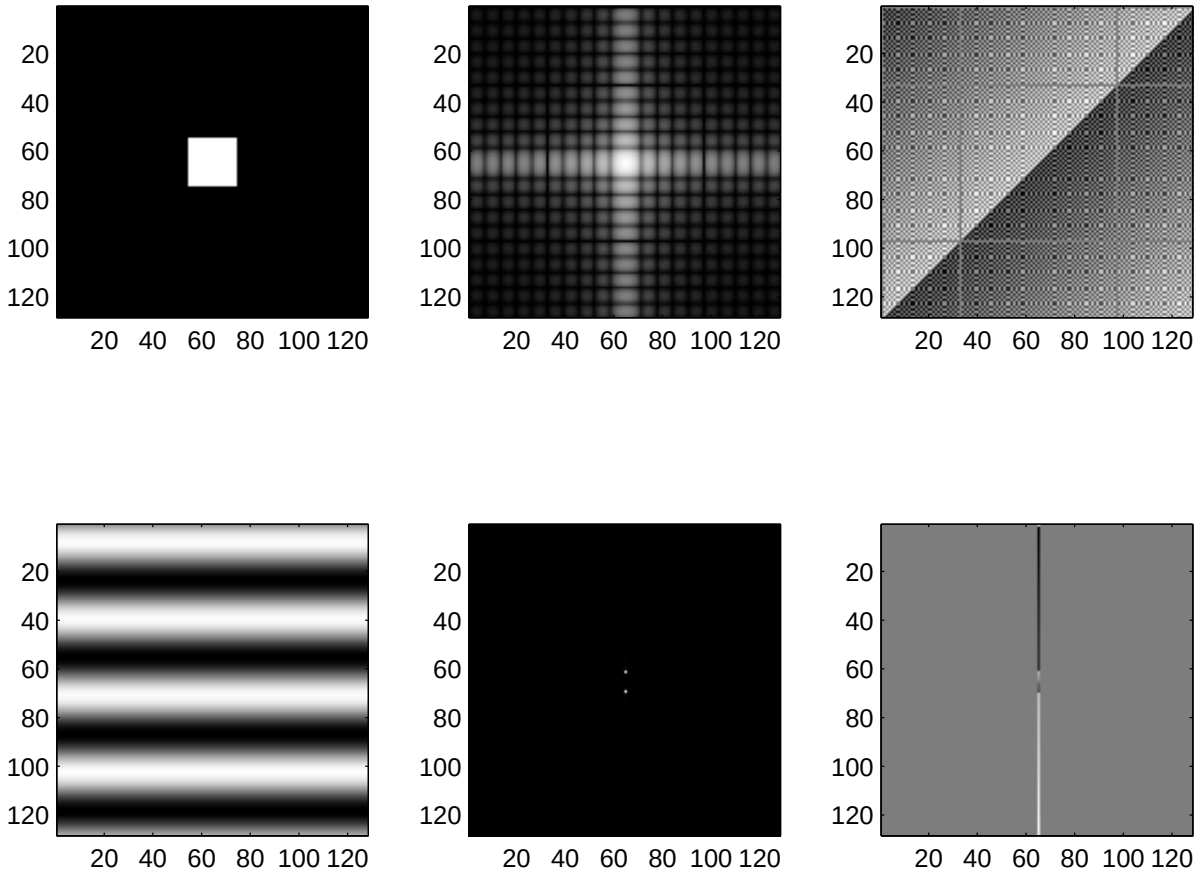


Figure 2.2: *Some 2-D functions and their resultant DFTs*

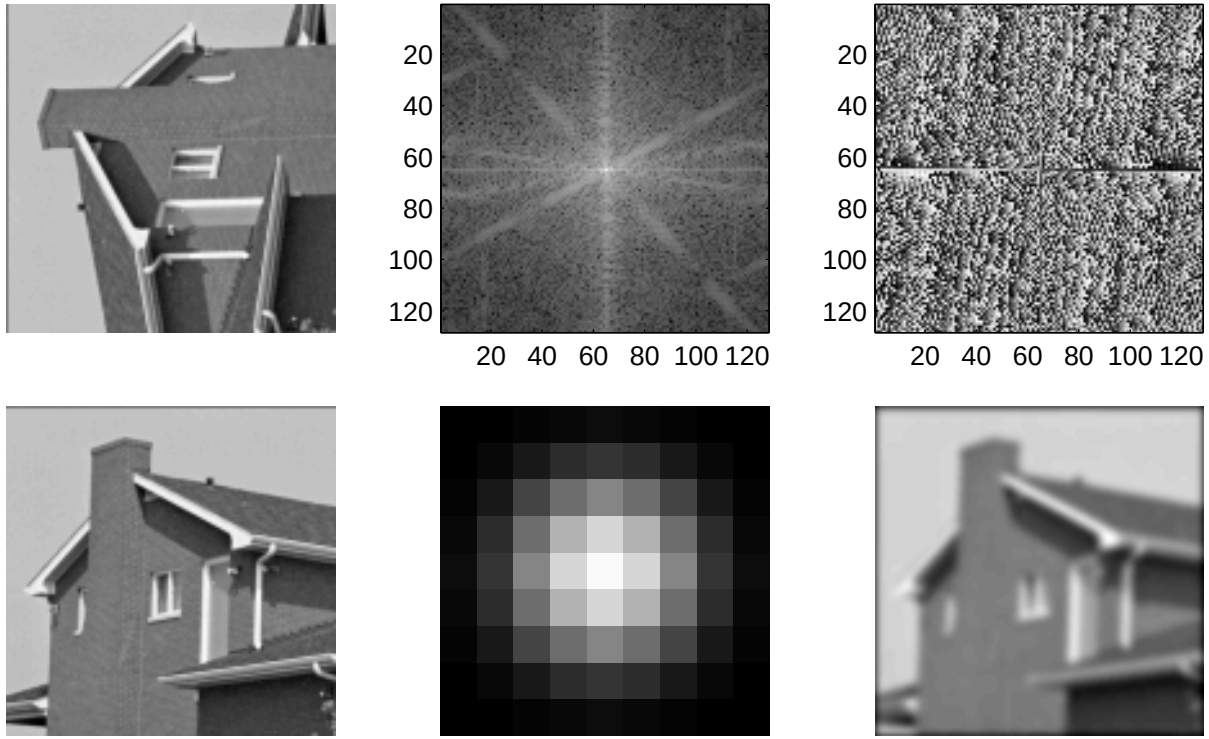


Figure 2.3: *Rotation in FT space (top) and 2-D Convolution (bottom).*

2.2 Other transforms

The FT represents a specific case in a more general transform theory. In the FT the image is decomposed into a series of harmonic functions (sines and cosines). These have the property of being *orthogonal* functions and form a *complete basis* set. They are not the only such functions, however.

- FT kernel basis
- Walsh
- Hadamard
- Discrete cosine transform (DCT)
- Wavelets - localised functions.

We concentrate later in the course on the use of the DCT as this is the most widely used of the above and forms the basis of JPEG compression. We also look at wavelets as these form the basis of the JPEG-2000 compression scheme.

2.3 Digital filtering

2.3.1 The sampling process

This is performed by an *analogue to digital converter* (ADC) in which the continuous function $f(x)$ is replaced by a "discrete function" $f[k]$, which is defined only at $x = kT$, with $k = 0, 1, 2$. We thence only need consider the digitised sample set $f[k]$ and the sample interval T. A simple generalisation allows for a sampled set over the 2-D plane, with samples at $u\Delta M$, $v\Delta N$ so that u, v indexes the image pixels.

Aliasing

Consider $f(x) = \cos(\frac{\pi}{2}\frac{t}{T})$ (one cycle every 4 samples) and also $f(t) = \cos(\frac{3\pi}{2}\frac{x}{T})$ (3 cycles every 4 samples) as shown in the Figure. Note that the resultant samples are the same. This result is referred to as aliasing.

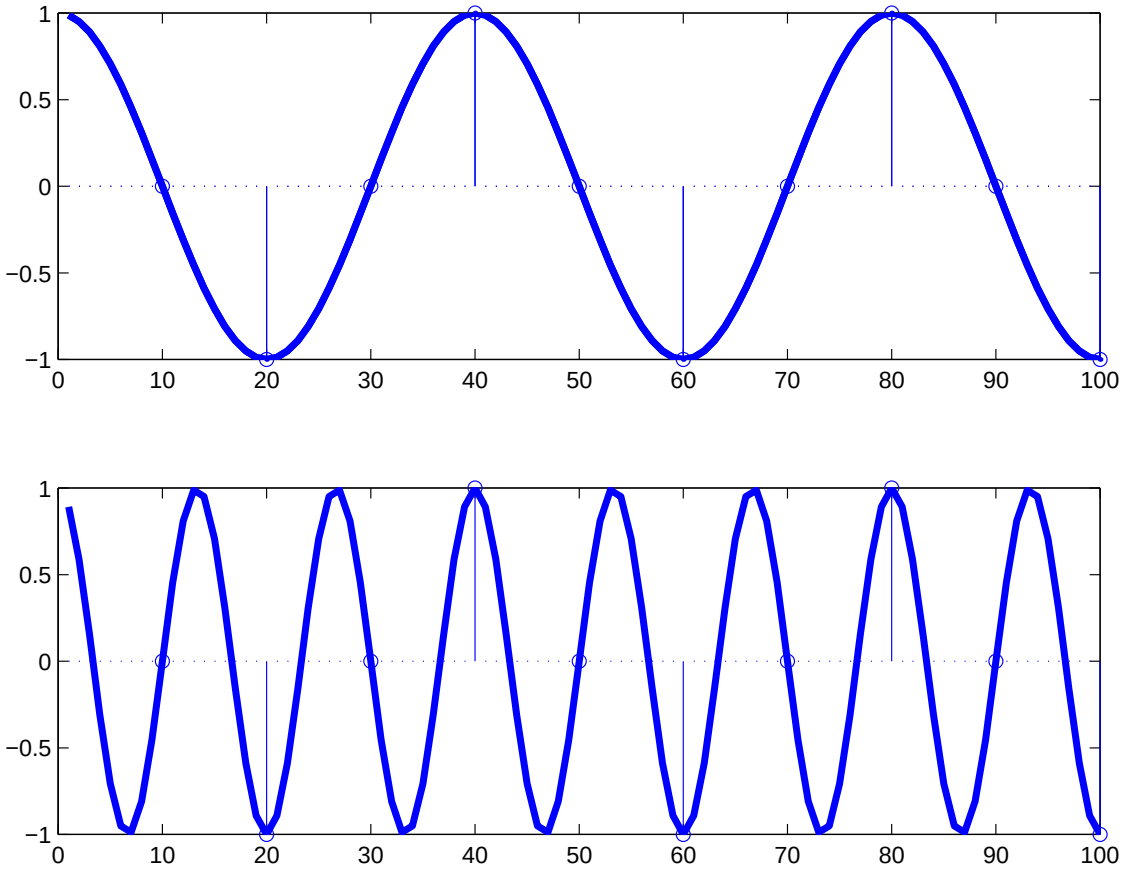


Figure 2.4: *Aliasing*.

2.4 Introduction to the principles of digital filtering

We can see that the numerical processing is at the heart of the digital filtering process. How can the arithmetic manipulation of a set of numbers produce a "filtered" version of that set? Consider the noisy signal of figure 2.5, together with its sampled version:

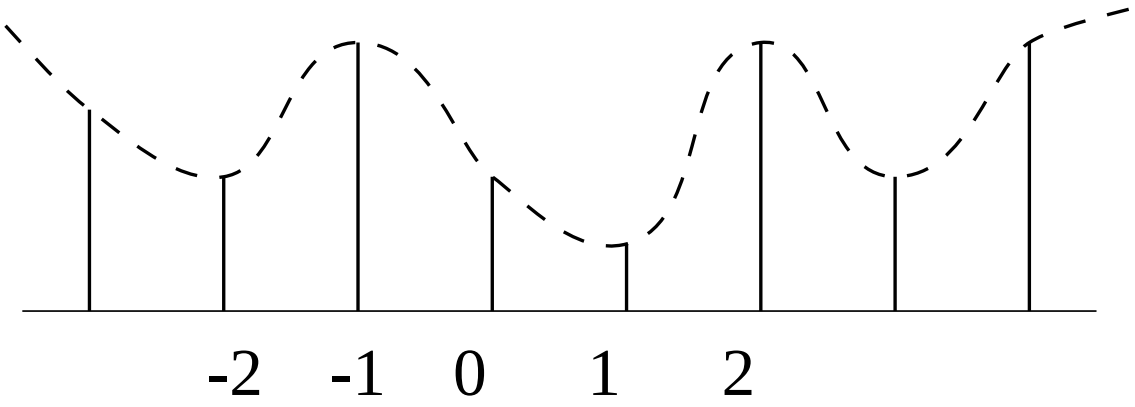


Figure 2.5: *Noisy data.*

One way to (e.g) reduce the noise might be to try and *smooth* the data. For example, we could try a polynomial fit using a least-squares criterion. If we choose, say, to fit a parabola to every group of 5 points in the sequence, then, for every point, we will make a parabolic approximation to that point using the value of the sample at that point together with the values of the 4 nearest samples (this forms a *parabolic filter*), as in Fig. 2.6

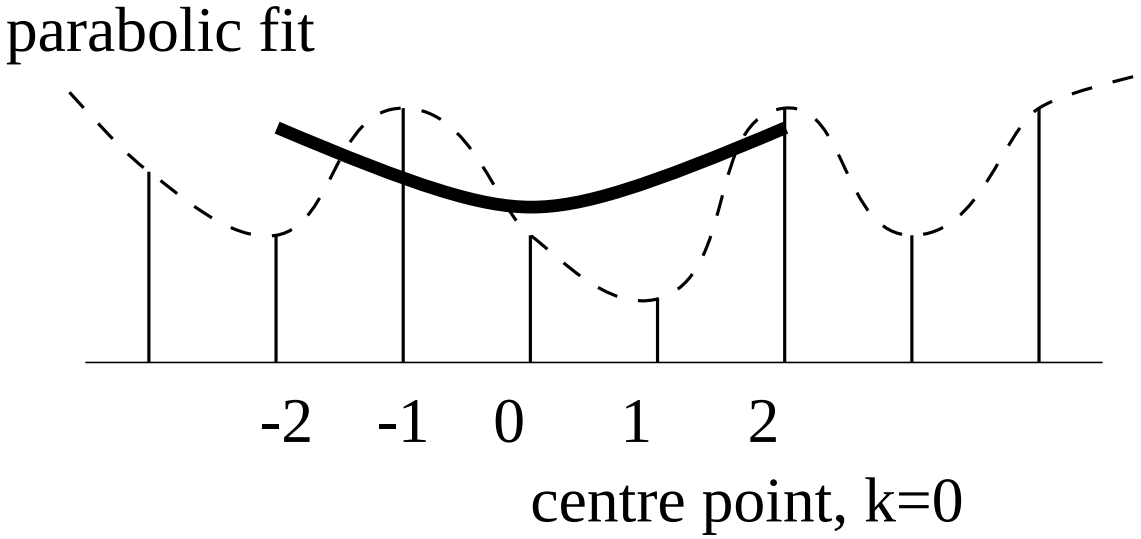


Figure 2.6: *Parabolic fit.*

$$p[k] = s_0 + ks_1 + k^2 s_2$$

where $p[k]$ = value of parabola at each of the 5 possible values of $k = \{-2, -1, 0, 1, 2\}$ and s_0, s_1, s_2 are the variables used to fit each of the parabolae to 5 input data points.

We obtain a fit by finding a parabola (coefficients s_0, s_1 and s_2) which best approximates the 5 data points as measured by the least-squares error E:

$$E(s_0, s_1, s_2) = \sum_{k=-2}^{2} (x[k] - [s_0 + ks_1 + k^2 s_2])^2$$

Minimizing the least-squares error gives:

$$\frac{\partial E}{\partial s_0} = 0, \quad \frac{\partial E}{\partial s_1} = 0, \quad \text{and } \frac{\partial E}{\partial s_2} = 0$$

and thus:

$$5s_0 + 10s_2 = \sum_{k=-2}^{k=2} x[k]$$

$$10s_1 = \sum_{k=-2}^{k=2} kx[k]$$

$$10s_0 + 34s_2 = \sum_{k=-2}^{k=2} k^2 x[k]$$

which therefore gives:

$$s_0 = \frac{1}{35}(-3x[-2] + 12x[-1] + 17x[0] + 12x[1] - 3x[2])$$

$$s_1 = \frac{1}{10}(-2x[-2] - x[-1] + x[1] + 2x[2])$$

$$s_2 = \frac{1}{14}(2x[-2] - x[-1] - 2x[0] - x[1] + 2x[2])$$

The centre point of the parabola is given by:

$$p[k] \mid_{k=0} = s_0 + ks_1 + k^2 s_2 \mid_{k=0} = s_0$$

Thus, the parabola coefficient s_0 given above is the output sequence number calculated from a set of 5 input sequences points. The output sequence so obtained is similar to the input sequence, but with less noise (i.e. low-pass filtered) because the parabolic filtering provides a smoothed approximation to each set of five data points in the sequence. Fig. 2.7 shows this filtering effect.

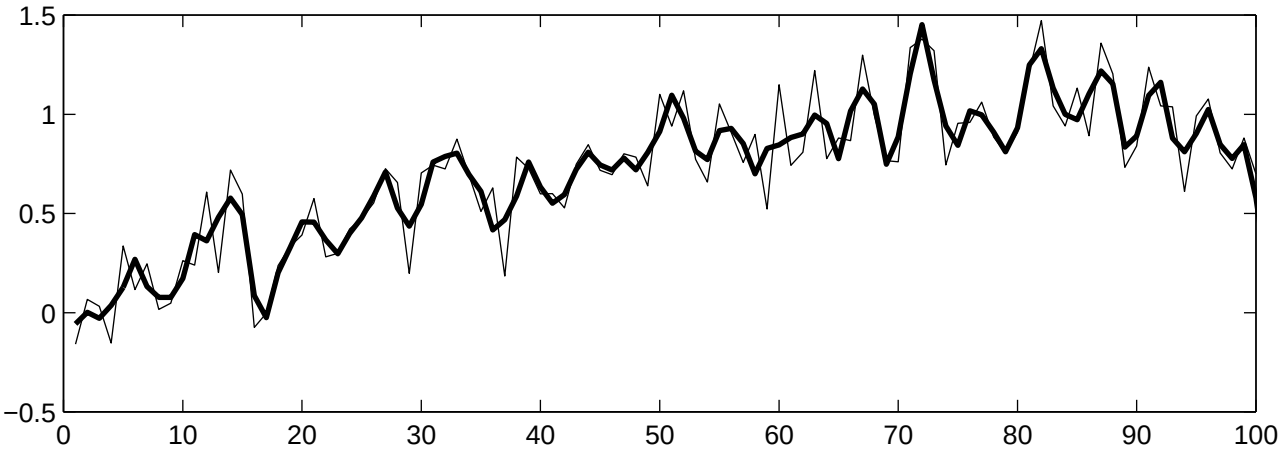


Figure 2.7: *Noisy data (thin line) and 5-point parabolic filtered (thick line).*

The magnitude response (which we will re-consider later) for the 5-point parabolic filter is shown below in Fig. 2.8.

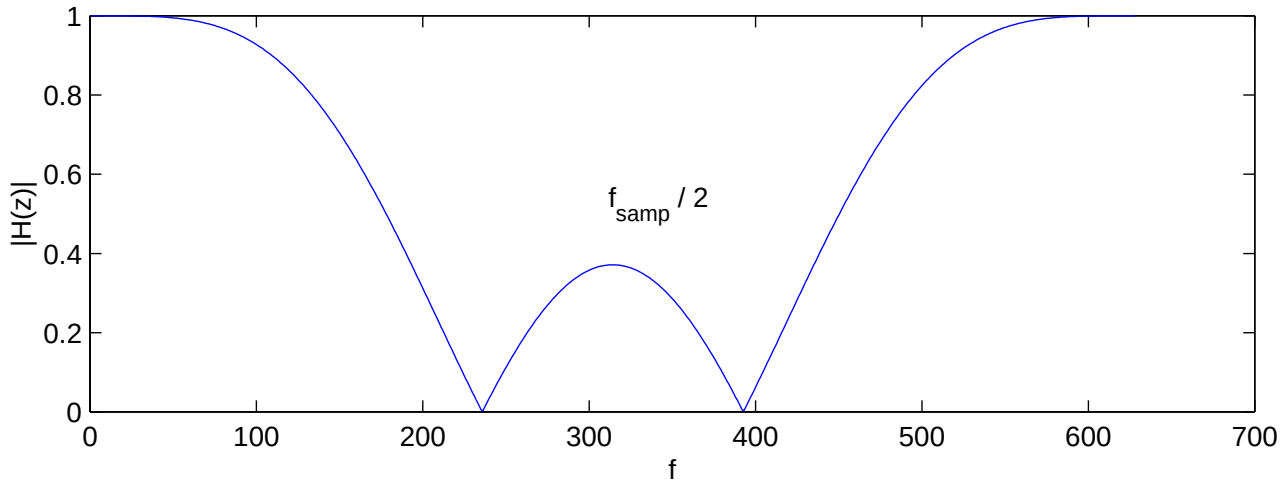


Figure 2.8: *Frequency response of 5-point parabolic filter.*

The filter which has just been described is an example of a *non- recursive* digital filter, which are defined by the following relationship (known as a *difference equation*):

$$r[k] = \sum_{i=0}^{N} a_i f[k-i]$$

where the a_i coefficients determine the filter characteristics. The difference equation for the 5-point smoothing filter, therefore, is:

$$r[k] = \frac{1}{35}(-3f[k+2] + 12f[k+1] + 17f[k] + 12f[k-1] - 3f[k-2])$$

This is a *non-causal* filter since a given output value $r[k]$ depends not only on previous inputs, but also on the current input $f[k]$, the input $f[k+1]$ and the input $f[k+2]$. The problem is solved by delaying the calculation of the output value $f[k]$ (the centre point of the parabola) until all the 5 input values have been sampled (i.e. a delay of $2T$ where $T =$ sampling period), ie:

$$r[k] = \frac{1}{35}(-3f[k] + 12f[k-1] + 17f[k-2] + 12f[k-3] - 3f[k-4])$$

It is of importance to note that the equation $r[k] = \sum a_i f[k-i]$ represents a *discrete convolution* of the input data with the filter coefficients; hence these

coefficients constitute the *impulse response* of the filter.

Proof:

Let $f[k] = 0$, except at $k = 0$, where $f[0] = 1$. Then $r[k] = \sum_i a_i f[k-i] = a_k f[0]$ (all terms zero except when $i = k$). This is equal to a_k since $f[0] = 1$. Therefore $r[0] = a_0$; $r[1] = a_1$; etc .... As there is a finite number of a's, the impulse response is finite. For this reason, non-recursive filters are also called **Finite-Impulse Response** (FIR) filters.

As we will see, we may also formulate a digital filter as a recursive filter; in which, the output $r[k]$ is also a function of previous outputs:

$$r[k] = \sum_{i=0}^{N} a_i f[k-i] + \sum_{i=1}^{M} b_i r[k-i]$$

Before we can describe methods for the design of both types of filter, we need to review the concept of the z-transform.

2.5 The z-transform

The z-transform is important in digital filtering because it describes the sampling process and plays a role in the digital domain similar to that of the Laplace transform in analogue filtering.

The Laplace transform of a unit impulse occurring at time $x = kT$ is e^{-kTs}. Consider the discrete function $f[k]$ to be a succession of impulses, for example of area $f(0)$ occurring at $x = 0$, $f(1)$ occurring at $x = T$, etc .... The Laplace transform of the whole sequence would be:

$$F_d(s) = f(0) + f(1)e^{-Ts} + f(2)e^{-2Ts} + \ldots + f[k]e^{-kTs}$$

The suffix d denotes the transform of the *discrete* sequence, not of the continuous $f(t)$.

Let us replace e^{Ts} by a new variable z, and rename $F_d(s)$ as $F(z)$:

$$F(z) = f(0) + f(1)z^{-1} + f(2)z^{-2} + \ldots f[k]z^{-k}$$

For many functions, the infinite series can be represented in "closed form", in general as the ratio of two polynomials in z^{-1}.

2.5.1 The Pulse Transfer Function

This is the name for (z-transform of output)/(z-transform of input).

Let the impulse response, for example of an FIR filter, be a_0 at $t = 0$, a_1 at $x = T$, $\ldots a_n$ at $x = nT$ with $n = 0$ to N.

Let $G(z)$ be the z-transform of this sequence:

$$G(z) = a_0 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_i z^{-i} + \ldots a_N z^{-N}$$

Let $F(z)$ be an input expressed in the z-domain as:

$$F(z) = f[0] + f[1]z^{-1} + f[2]z^{-2} + \ldots + f[k]z^{-k} + \ldots$$

The product $G(z)F(z)$ is:

$$G(z)F(z) = (a_0 + a_1 z^{-1} + \ldots a_n z^{-n} + \ldots a_N z^{-N})(f[0] + f[1]z^{-1} + \ldots f[k]z^{-k})$$

in which the coefficient of z^{-k} is:

$$a_0 f[k] + a_1 f[k-1] + \ldots a_n f[k-n] + \ldots a_N f[k-N]$$

This is nothing else than the value of the output sample at $x = kT$. Hence the whole sequence is the z-transform of the output, say $R(z)$, where $R(z) = G(z)F(z)$. Hence the pulse transfer function, $G(z)$, is the z-transform of the impulse response.

For non-recursive filters:

$$G(z) = \sum_{n=0}^{N} a_n z^{-n}$$

For recursive filters:

$$R(z) = \sum_{n=0}^{N} a_i z^{-i} F(z) + \sum_{i=m}^{M} b_i z^{-i} R(z)$$

$$G(z) = \frac{R(z)}{F(z)} = \frac{\sum_n a_n z^{-n}}{1 - \sum_m b_m z^{-m}}$$

2.5.2 z-plane pole-zero plot

Let $z = e^{sT}$, where T = sampling period. Since $s = \sigma + i2\pi u$, we have:

$$z = e^{\sigma T} e^{i2\pi uT}$$

If $\sigma = 0$, then $\mid z \mid = 1$ and $z = e^{i2\pi uT} = \cos 2\pi uT + i \sin 2\pi uT$, i.e. the equation of a circle of unit radius (the *unit circle*) in the z-plane.

Thus, the imaginary axis in the s-plane ($\sigma = 0$) maps onto the unit circle in the z-plane and the left half of the s-plane ($\sigma < 0$) onto the *interior* of the unit circle.

We know that all the poles of $G(s)$ must be in the left half of the s-plane for a continuous filter to be stable. We can therefore state the equivalent rule for stability in the z-plane:

For stability all poles in the z-plane must be inside the unit circle.

2.6 Frequency response of a digital filter

This can be obtained by evaluating the (pulse) transfer function on the unit circle (i.e. $z = e^{2\pi i u T}$).

Proof

Consider the general filter

$$r[k] = \sum_{n=0}^{\infty} a_n f[k-n]$$

NB: A recursive type can always be expressed as an infinite sum by dividing out:

$$\text{eg., for} \quad G(z) = \frac{a_0}{1 - b_1 z^{-1}}, \qquad \text{we have} \quad r[k] = \sum_{n=0}^{\infty} a_0 . b_1^n f[k-n]$$

Let input before sampling be $\cos(2\pi ut + \theta)$, sampled at $t = 0, T, \ldots, kT$.
Therefore $f[k] = \cos(2\pi ukT + \theta) = \frac{1}{2}\{e^{i(2\pi ukT+\theta)} + e^{-i(2\pi ukT+\theta)}\}$

$$\text{ie.} r[k] = \frac{1}{2}\sum_{n=0}^{\infty} a_n e^{i\{2\pi u[k-n]T+\theta\}} + \frac{1}{2}\sum_{n=0}^{\infty} a_n e^{-i2\pi\{u[k-n]T+\theta\}}$$

$$= \frac{1}{2}e^{i(2\pi ukT+\theta)}\sum_{n=0}^{\infty} a_n e^{-i2\pi unT} + \frac{1}{2}e^{-i2\pi(ukT+\theta)}\sum_{n=0}^{\infty} a_n e^{i2\pi unT}$$

$$\text{Now } \sum_{n=0}^{\infty} a_n e^{-i2\pi unT} = \sum_{n=0}^{\infty} a_n (e^{i2\pi uT})^{-n}$$

$$\text{But } G(z) \text{ for this filter is } \quad \sum_{n=0}^{\infty} a_n z^{-n}$$

$$\text{and so } \quad \sum_{n=0}^{\infty} a_n e^{-i2\pi unT} = G(z)_z = e^{i2\pi uT}$$

Let $G(z)_z = e^{i2\pi uT} = Ae^{i\phi}$.
Then

$$\sum_{n=0}^{\infty} a_n e^{i2\pi unT} = Ae^{-i\phi} \quad \text{(complex conjugate)}$$

Hence $r[k] = \frac{1}{2}e^{i(2\pi ukT+\theta)}Ae^{i\phi} + \frac{1}{2}e^{-i(2\pi ukT+\theta)}Ae^{-i\phi}$

$$\text{or} \quad r[k] = A\cos(2\pi ukT + \theta + \phi) \qquad \text{when } f[k] = \cos(2\pi ukT + \theta)$$

Thus A and ϕ represent the gain and phase of the frequency response. i.e. the frequency response (as a complex quantity) is

$$G(z)|_{z=e^{i2\pi uT}}$$

Lecture 3 – Restoration : global filters

3.1 Noise removal

3.1.1 Signal in additive noise

Consider a signal $f(x)$ corrupted by noise $n(x)$ such that

$$s(x) = f(x) + n(x)$$

If the freqency support of the signal and the noise is different then we may recover the signal by (typically) low-pass filtering.

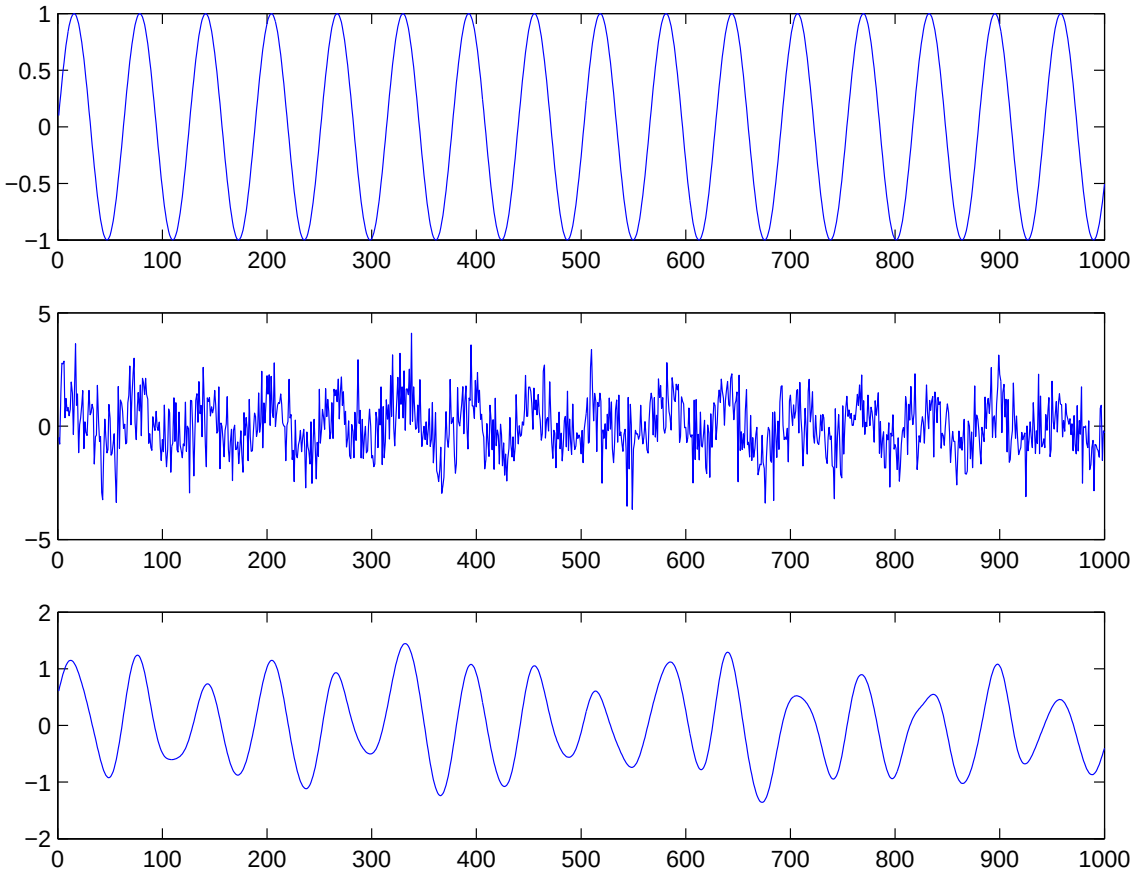


Figure 3.1: *Simple low-pass filtering of additive noise*

3.1.2 Signal in multiplicative noise

Consider our signal corrupted by noise of the form:

$$s(x) = f(x)n(x)$$

One way we can work with this is to use the idea of *homomorphic filtering*. Without loss of generality we can make $s(x) > 0$ for all x (ie just a DC shift). Then we can take logs,

$$\log s(x) = \log f(x) + \log n(x)$$

which is just an additive noise system. We can then (low-pass) filter this system and invert the log (ie exponentiate) to obtain an enhanced signal or image.

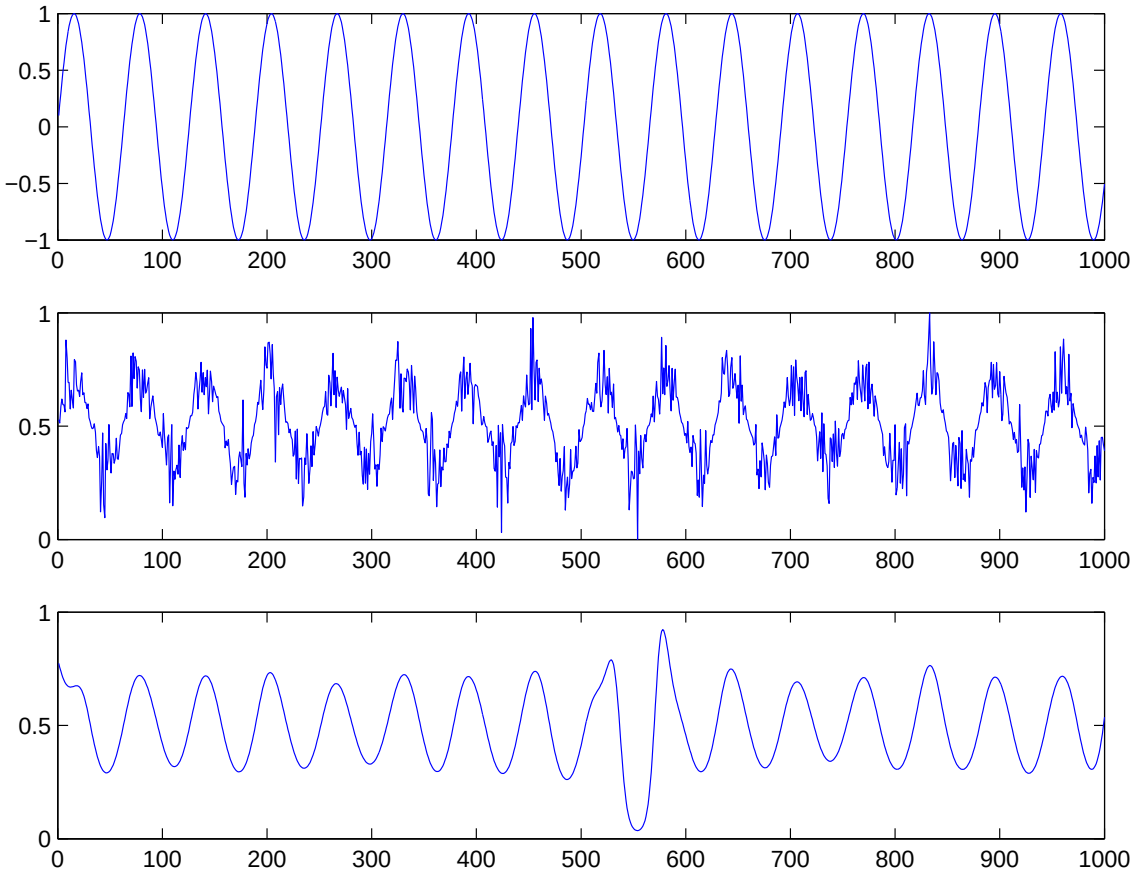


Figure 3.2: *Homomorphic filtering of multiplicative noise.*

3.2 Inverse filters

In general, we will have something more to contend with than just a noise process. Consider a camera with a blur. Consider taking a picture of a moving car, consider a microphone with a non-perfect transfer function.

In general then, the signal or image we obtain will be (considering here the additive noise case),

$$s(x) = b(x) * f(x) + n(x)$$

where $n(x)$ is our noise process and $b(x)$ is a blur or convolution kernel. How do we proceed to obtain our enhanced, restored $\hat{f}(x)$?

If we take a Fourier Transform of $s(x)$ we obtain

$$S(u) = B(u)F(u) + N(u)$$

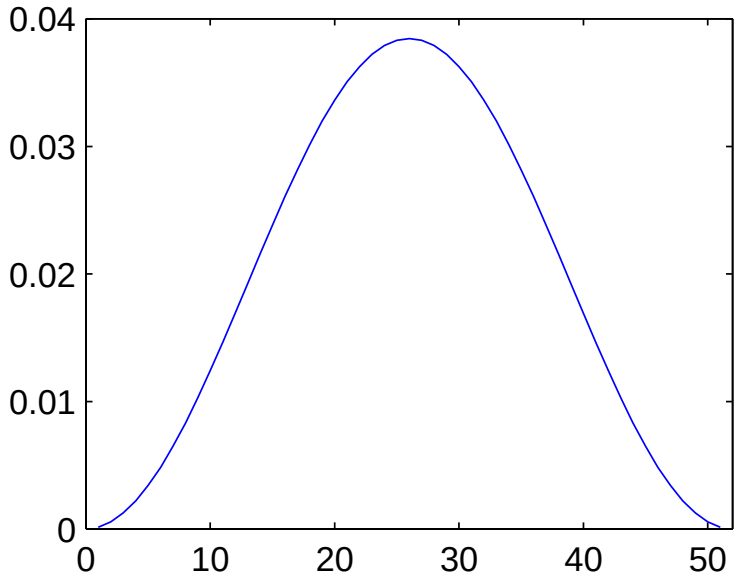
which yields

$$F(u) = B^{-1}(u)\left(S(u) - N(u)\right)$$

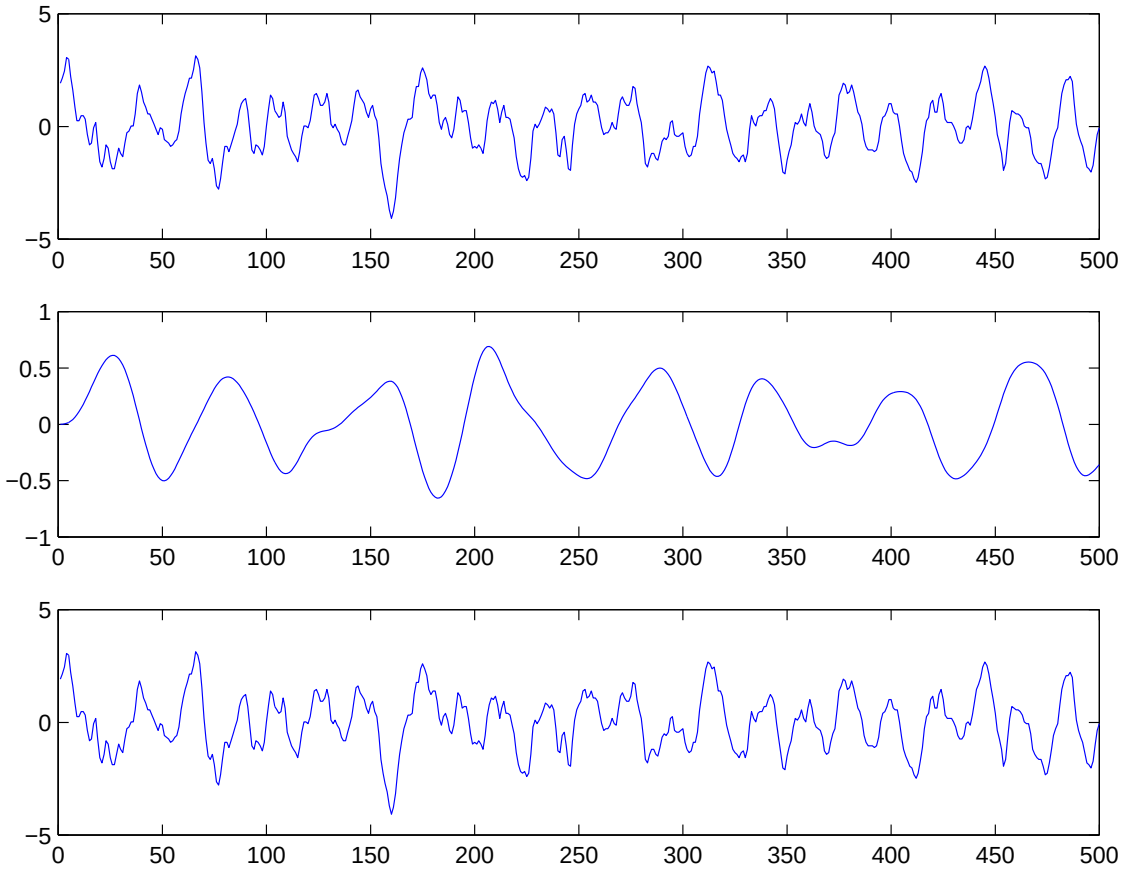
or, if we can filter out the effects of $n(x)$ easily (using a low-pass filter, for example)

$$F(u) = B^{-1}(u)S(u)$$

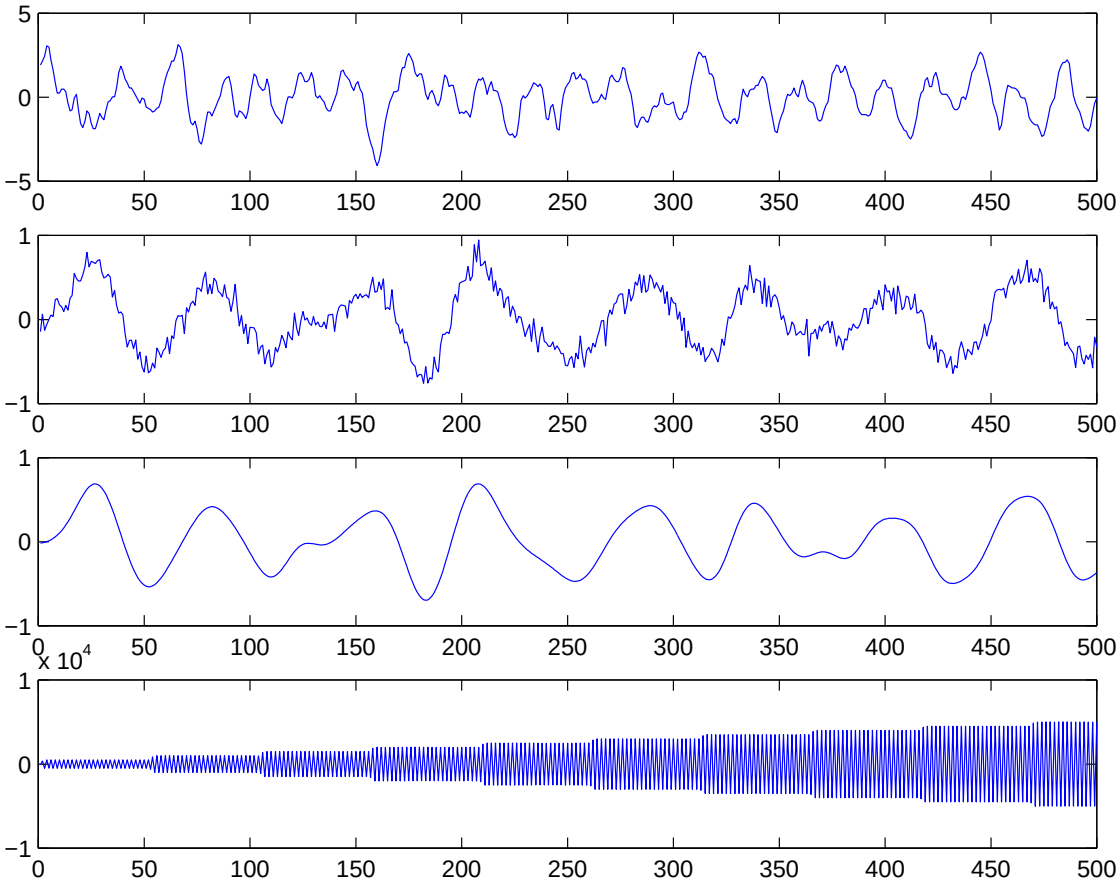
where $B^{-1}(u)$ is the *inverse filter* which corrects for the *deterministic* corruption of the original. Consider a simple 'raised cosine' blur, shown below...



When we know the convolution kernel, $b(x)$, we can achieve almost perfect results...



The example above was all in the absence of the noise, $n(x)$.



Noisy deconvolution is a messy business! Also, in all the examples looked at above we assume we know the convolution kernel, $b(x)$. This may be reasonable if we measure the spread of a camera lens, e.g. but for arbitrary images and signals we don't always know this. There are methods, so called *blind deconvolution*

approaches, which can solve this but they are outside the scope of this course.

3.2.1 Motion blur

Let the true image be $f(x, y)$. Let the displacement in the x, y directions be $\alpha(t), \beta(t)$. Hence, over a time period $-T/2, T/2$ the resultant image is given by

$$g(x, y) = \int_{-T/2}^{T/2} f(x - \alpha(t), y - \beta(t)) dt$$

Taking the FT gives

$$G(u, v) = \int_{-T/2}^{T/2} \int \int f(x - \alpha(t), y - \beta(t)) \exp\{-i2\pi(ux + vy)\} dx dy dt$$

Letting $\xi = x - \alpha(t)$ and $\nu = y - \beta(t)$ so

$$G(u, v) = \int_{T/2}^{T/2} \left(\int \int f(\xi, \nu) e^{-i2\pi(u\xi + v\nu)} d\xi d\nu \right) \exp\{-i[u\alpha(t) + v\beta(t)]\} dt$$

The term in the brackets is just the FT of $f(x, y)$ ie $F(u, v)$ which is independent of t, so

$$G(u, v) = F(u, v) \int_{T/2}^{T/2} \exp\{-i2\pi[u\alpha(t) + v\beta(t)]\} dt$$

which is of the form

$$G(u,v) = F(u,v)H(u,v)$$

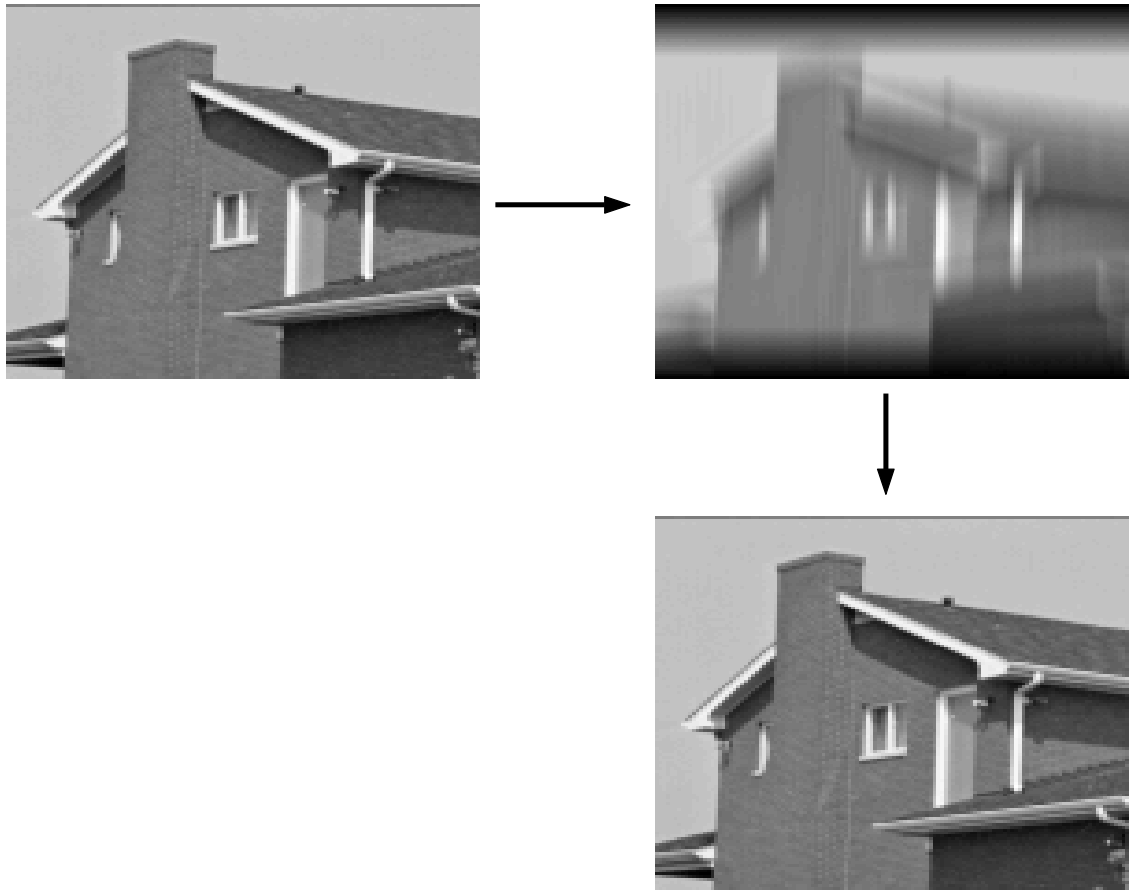
or

$$g(x,y) = f(x,y) * h(x,y)$$

If the motion is constant, i.e $\alpha(t) = Kt$ and $\beta(t) = Lt$ then

$$\begin{aligned} H(u,v) &= \int_{T/2}^{T/2} \exp\{-i2\pi[uK+vL]t\}dt \\ &= \frac{\sin(\pi[uK+vL]T)}{\pi[uK+vL]} \end{aligned}$$

i.e. a sinc function.



3.2.2 Other noise

The noise looked at above was additive or multiplication noise in which each datum of our signal was corrupted. Other noise exists, for example we may have a certain probability of corruption, such that most pixels or samples are fine, but once in a while we get a huge noise spike. We will look at this kind of noise, that 'replaces' the true sample with an outlying one, later in this course.

3.3 The Wiener filter

The Wiener Filter is a noise filter based on Fourier iteration. its main advantage is the short computational time it takes to find a solution.

Consider a situation such that there is some underlying, uncorrupted signal $f(t)$ that is required to measure. Error occur in the measurement due to imperfection in equipments, thus the output signal is corrupted. There are two ways the signal can be corrupted: First, the equipment can convolve, or 'smear' the signal. This occurs when the equipment doesn't have a perfect delta function response to the signal. Let $r(t)$ be the smeared signal and $g(t)$ be the (known) response that caused the convoltion. Then $r(t)$ is related to $f(t)$ by:

$$r(t) = f(t) * g(t)$$

or

$$R(u) = F(u)G(u)$$

where R, F, G are Fourier Transform of r, f, g. The second source of signal corruption is the unknown background noise $n(t)$. Therefore the measured signal $c(t)$ is a sum of $r(t)$ and $n(t)$:

$$c(t) = r(t) + n(t)$$

To deconvolve r to find f, simply divide $R(u)$ by $G(u)$ i.e.

$$F(u) = \frac{R(u)}{G(u)}$$

in the absense of noise n. To deconvolve c where n is present then one needs to find an optimum filter function $\phi(t)$ or $\Phi(u)$ which filters out the noise and gives a signal by:

$$\tilde{F}(u) = \frac{C(u)\Phi(u)}{G(u)}$$

where $\tilde{f}(t)$ is as close to the original signal as possible. For $\tilde{f}(t)$ to be similar to $f(t)$, their squared difference is as close to zero as possible, i.e.

$$\int |f(t) - \tilde{f}(t)|^2 dt$$

or

$$\int |F(u) - \tilde{F}(u)|^2 du$$

is minimised. After substitution and using the fact that the expectation of cross terms between $R(u)$ and $N(u)$ are zero we require the following to be minimised

$$\int |G(u)|^{-2} \left(|R(u)|^2 |1 - \Phi(u)|^2 + |N(u)|^2 |\Phi(u)|^2 \right) du$$

The best filter is one where the above integral is a minimum at every value of u. This is when,

$$\Phi(u) = \frac{|R(u)|^2}{|R(u)|^2 + |N(u)|^2}$$

Now we can make the approximation

$$|R(u)|^2 + |N(u)|^2 \approx |C(u)|^2$$

hence

$$\Phi(u) \approx \frac{|R(u)|^2}{|C(u)|^2}$$

From the above theory, it can be seen that a program can be written to Wiener Filter signal from noise using Fourier Transform.

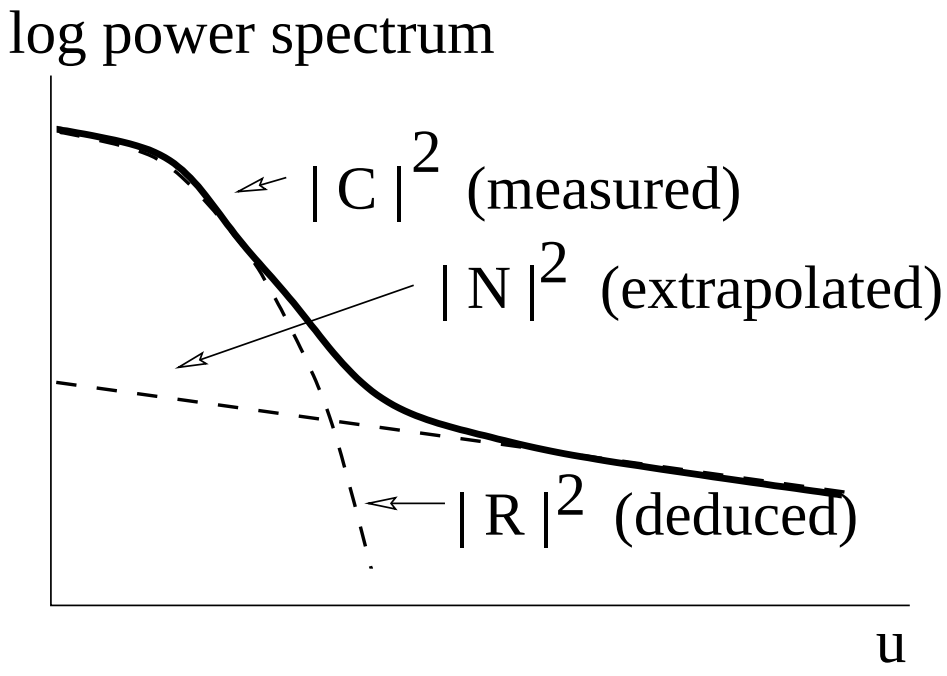


Figure 3.3: *Extracting Fourier terms needed in Wiener filter.*

There is another way to Wiener filtering a signal but this time without Fourier Transforming the data – this is the Mean-Squared Method.

The Mean-squared Method uses the fact that the Wiener Filter is one that is based on the least-squared principle, i.e. the filter minimises the square error between the actual output and the desired output. To do this, first the variance of the data matrix is to be found. Then, a box window is moved over the image matrix, moving one pixel at a time. In every box window, the local mean and variance is found. The filtered value of each datum (at index x, y say) is found by the following formula:

$$\tilde{f}_{x,y} = \mu_{x,y} + \frac{\sigma^2_{x,y} - s^2}{\sigma^2_{x,y}}(c_{x,y} - \mu_{x,y})$$

where $\tilde{f}_{x,y}$ is the filtered signal, $\mu_{x,y}$ is the local mean, $\sigma^2_{x,y}$ is the local variance, s^2 is the noise variance of the entire data matrix, and $c_{x,y}$ is the original signal value. From the above formula, it can be see that if the original signal is similar to the local mean then the filtered value will be the local mean, and if the original signal is very different from the local mean, then it will be filtered to give a higher/lower intensity signal depending on the differences. Also, if the local variance is similar to the matrix variance, which is around 1 (i.e. only noise exists in the box) then

the filtered signal will be that of the local mean, which should be close to zero. But if the local variance is much bigger than the matrix variance (i.e. when the box is at the actual signal), then the signal will be amplified. As the box moves through the entire matrix, it will calculate the solution to each pixel using the above formula, thus filtering the data.

The Wiener filter has been here introduced in its 2-d guise, but it is trivial to go to 1-d.

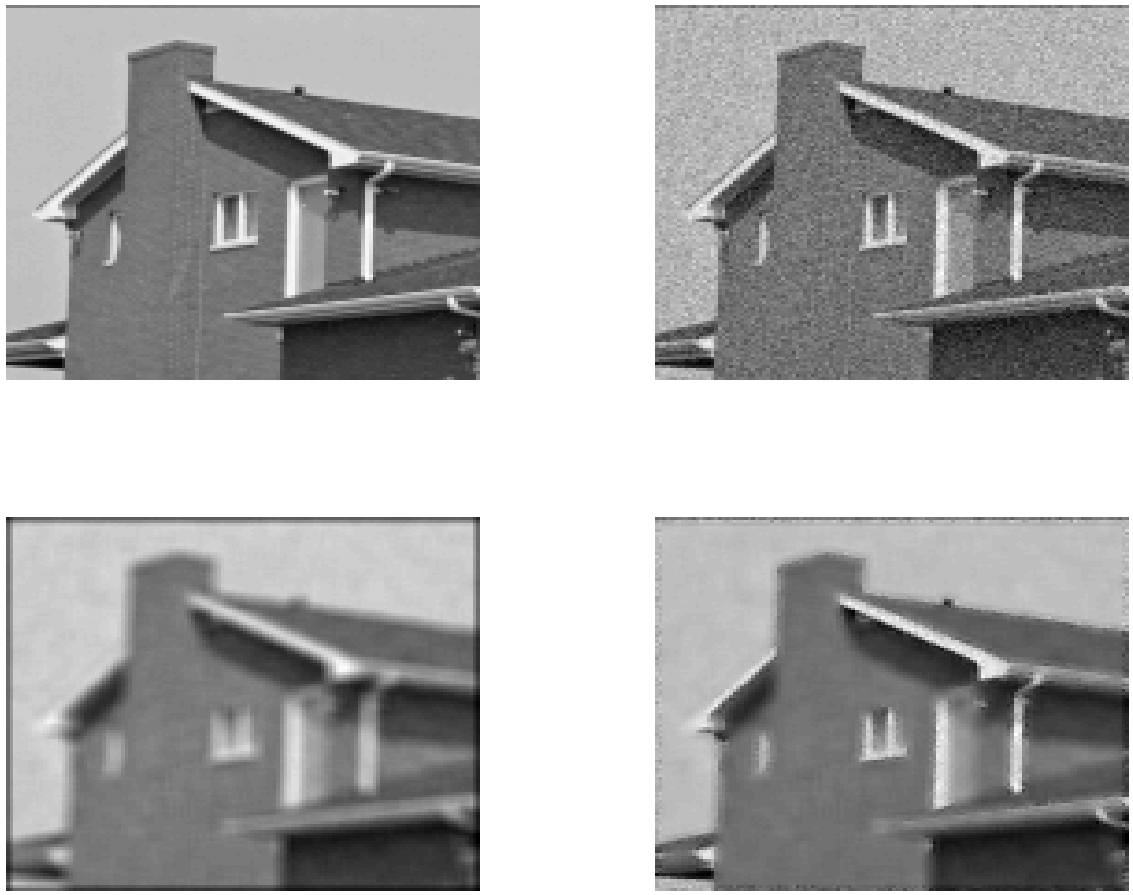


Figure 3.4: *Wiener filter (bottom right) improves over flat average filter (bottom left). 5 pixel square used in both cases.*

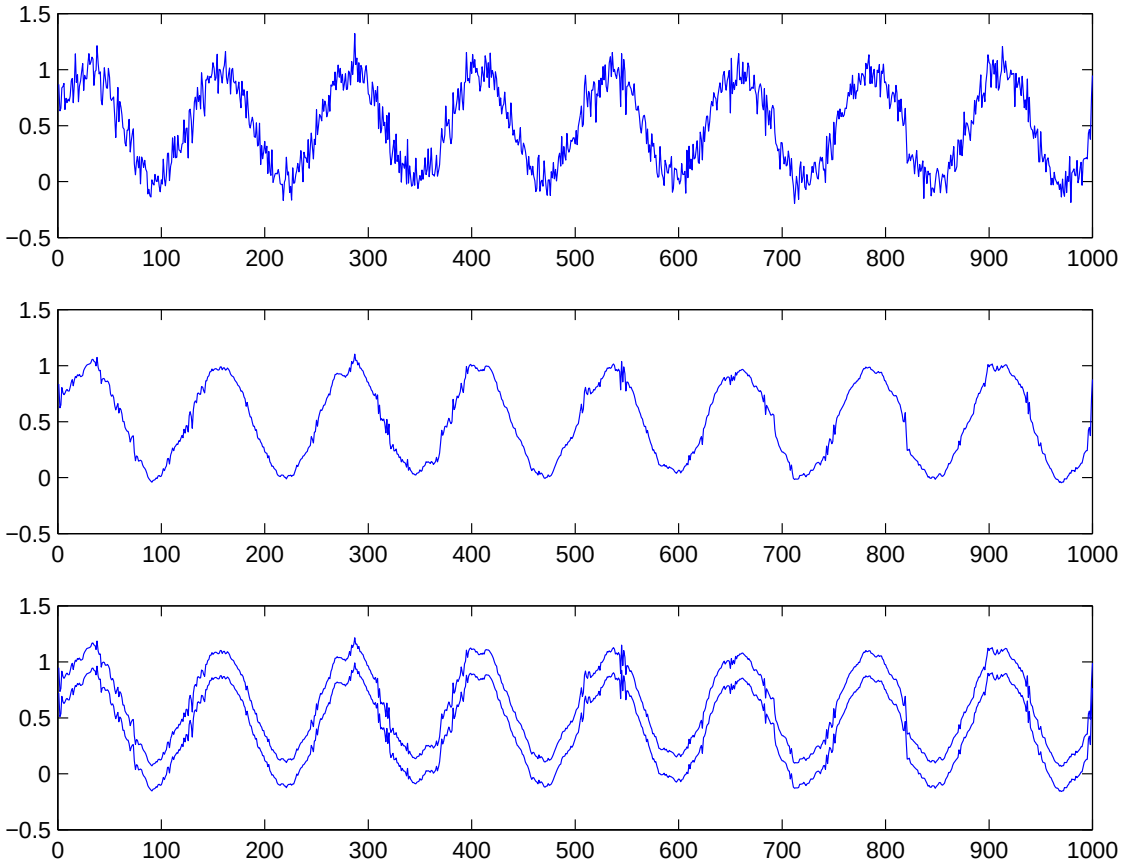


Figure 3.5: *Wiener filter adaptively removing noise - estimation of mean and one σ.*

Lecture 4 – Local derivative filters

4.1 Estimating derivatives in signals & images

In many applications it is useful to be able to estimate simple measures of derivatives of signals & images, to detect for example level changes or edges.

Consider a simple 1-d signal, $f(t)$. A simple backwards-difference (first order) estimate of the gradient may be written as

$$f'(t) = \frac{1}{\delta t}(f(t) - f(t-1))$$

We can also represent this as

$$f'(t) = f(t) * g(t)/\delta t$$

where $g(t) = [-1, 1]$ is a filter kernel (i.e. a simple linear filter). We must remember that such a filter has error of order $O(\delta t)$ whereas the central difference

estimate of the gradient is better, with $O(\delta t^2)$. Letting $\delta t = 1$ for ease of nomenclature gives

$$f'(t) = \frac{1}{2}(f(t+1) - f(t-1)) = f(t) * g(t)$$

where now $g(t) = \frac{1}{2}[-1, 0, 1]$. So we have a local filter. This general approach can, of course, be extended to higher-order derivatives. The second derivative, $f''(t)$ may be estimated as

$$f''(t) = \frac{d}{dt}f'(t) = f(t) * (g(t) * g(t))$$

where the filter kernel that does the work is $(g(t) * g(t))$. Letting $g(t) = [-1, 1]$ then we get a filter for the 2nd derivative as

$$g_2(t) = [1, -2, 1]$$

which you will have seen before. To make sure that the signal power out equals that in we can normalise with a factor of $|1| + |-2| + |1| = 4$ so now $g_2(t) = 1/4[1, -2, 1]$. The figures show this working on a signal.

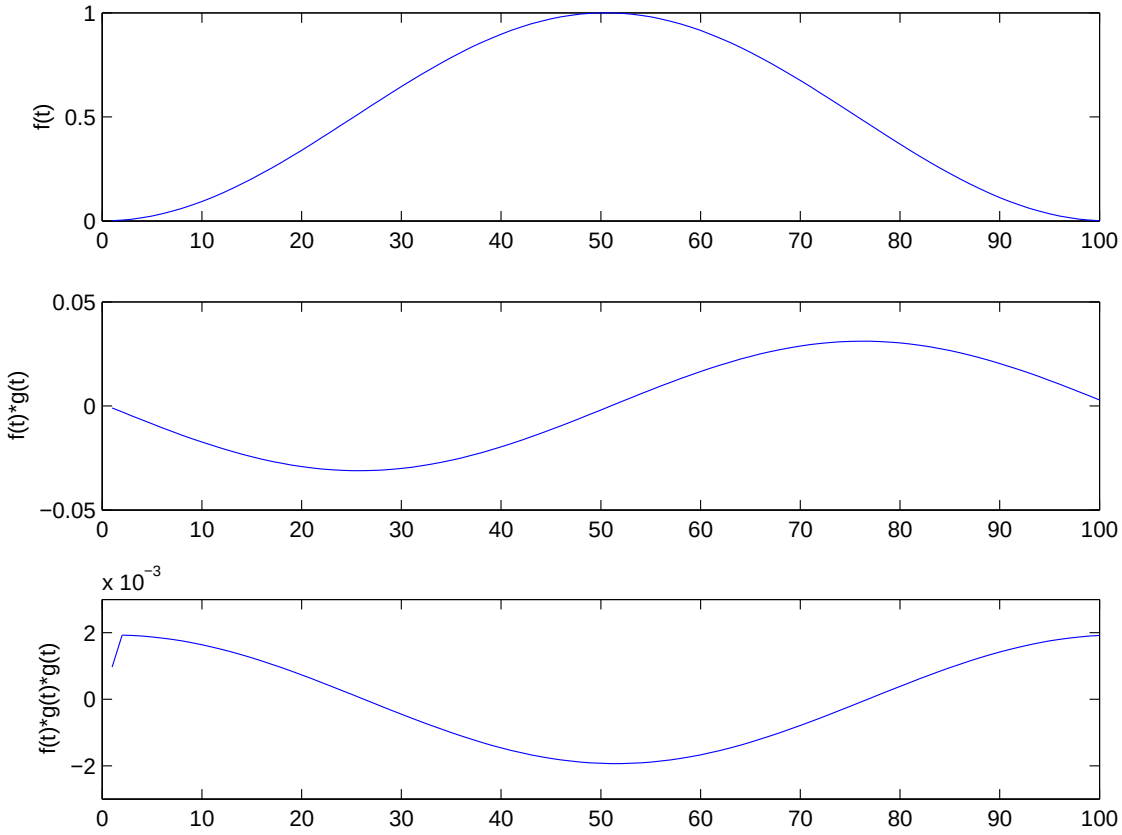


Figure 4.1: Derivative estimates

We can, of course, simply extend this to images by running over the rows and columns using one of the above filters, but normally we will extend these 1-d filters to a 2-d mask, typically making the mask square. For example, the 1-d central difference filter has a kernel (ignoring the factor of $1/2$ for a moment) $g(t) = [-1, 0, 1]$. In its 2-d guise, operating along rows (x-direction), a 2d mask may be made

$$g_x(x, y) = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

and similarly in the y-direction

$$g_y(x, y) = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

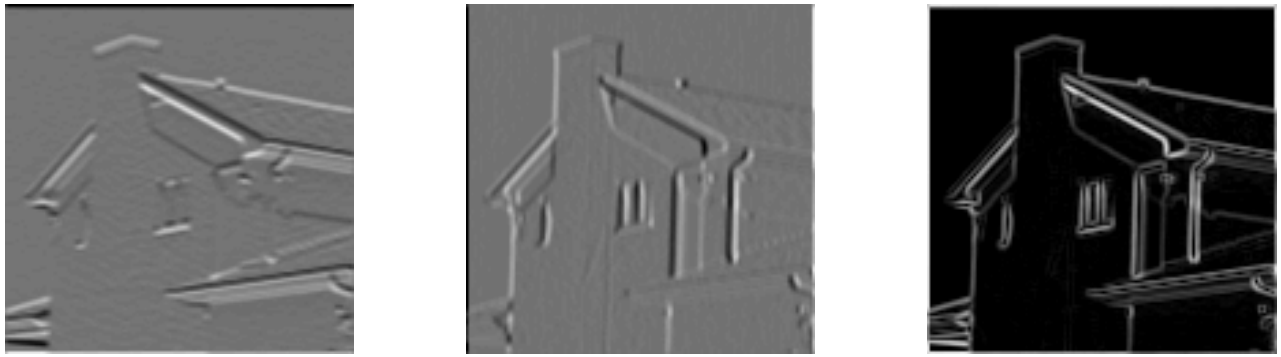


Figure 4.2: Derivative estimates on a simple image in y, x directions and magnitude.

We may formulate the total gradient in a 2-d setting via

$$g(x, y) = \sqrt{g_x^2 + g_y^2}$$

which is shown in the right-hand subplot of the figure.

The 2-d second derivatives may also easily be inferred via

$$g2_x(x, y) = \begin{bmatrix} 1 & -2 & 1 \\ 1 & -2 & 1 \\ 1 & -2 & 1 \end{bmatrix}$$

and similarly in the y-direction

$$g2_y(x, y) = \begin{bmatrix} 1 & 1 & 1 \\ -2 & -2 & -2 \\ 1 & 1 & 1 \end{bmatrix}$$

and hence the 2-d Laplacian may be given as

$$\nabla^2(x, y) = g2_x(x, y) + g2_y(x, y) = \begin{bmatrix} 2 & -1 & 2 \\ -1 & -4 & -1 \\ 2 & -1 & 2 \end{bmatrix}$$

and these give edge information, as valid potential edge points have $\nabla^2(x, y) = 0$. The Figure shows $|\nabla^2(x, y)|$ applied to the 'house' image.

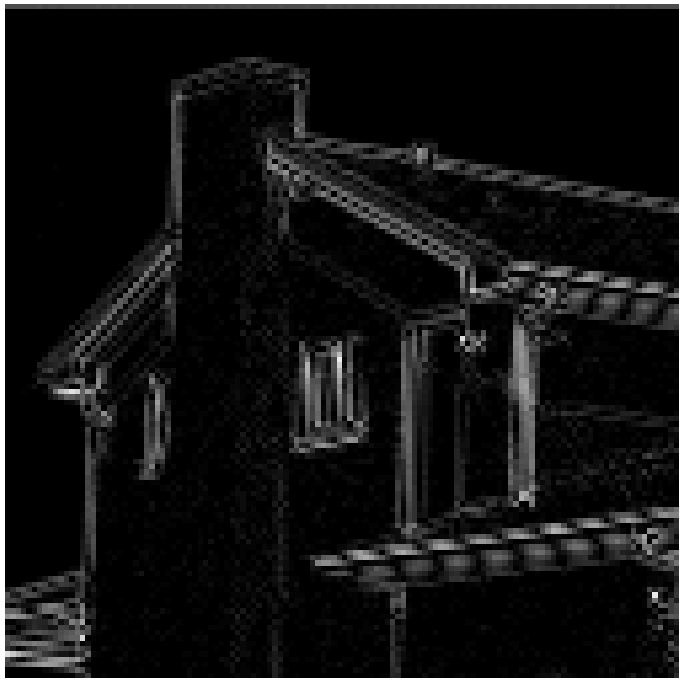


Figure 4.3: Laplacian estimates on a simple image.

The major problems with such simple gradient estimators, in both 1- and 2-d, lie in their lack of robustness in the presence of noise. The way around this is to use simple parametric functions, such as polynomials (as covered in the A1 course) or *splines* and then take derivatives of the fitted functions. The Figure shows the profound improvement for a simple signal using this approach.

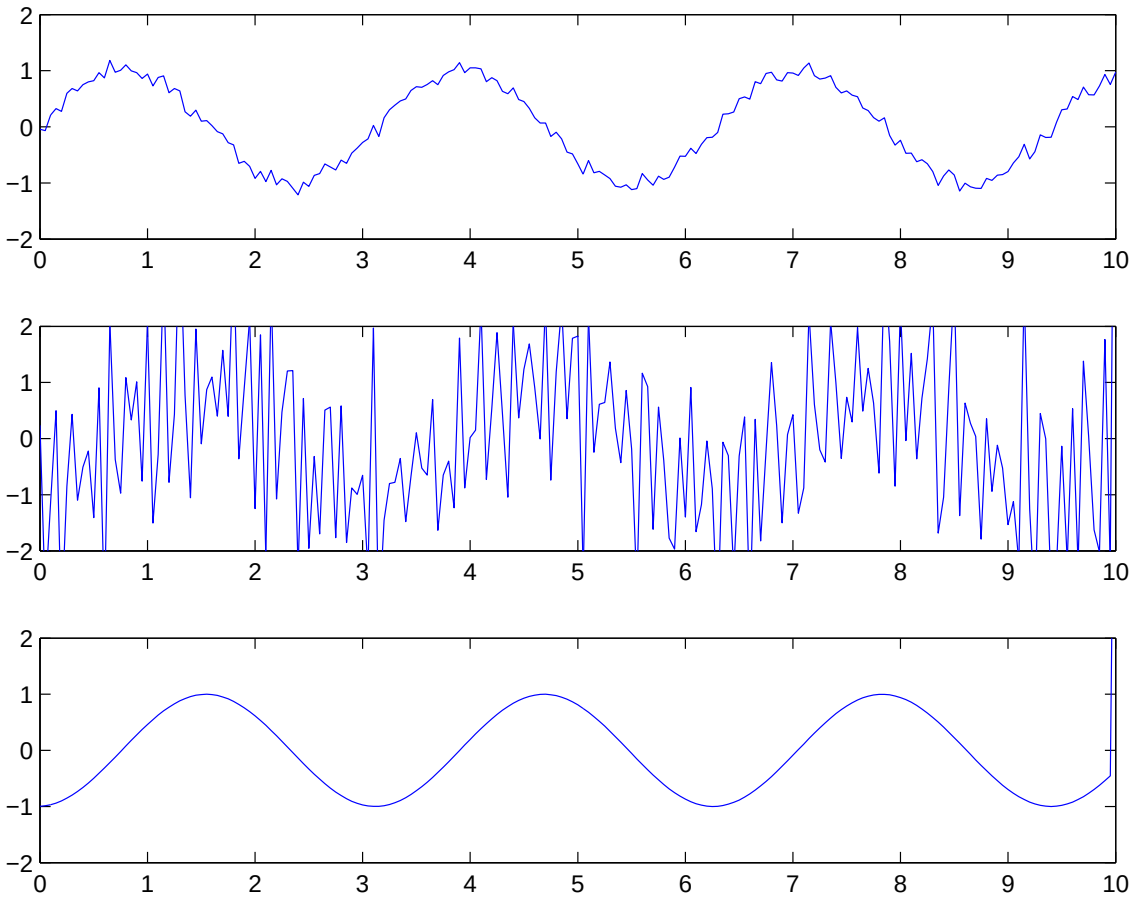


Figure 4.4: Gradient estimates on a simple signal. Top: noisy signal, middle: simple gradient filter, bottom: gradient of polynomial fit.

4.2 Motion analysis

In many machine applications we would like to estimate the motion of objects - which can be caused by

- Moving objects
- Moving camera, e.g. on a robot arm or an autonomous robot
- Moving objects and camera

In essence we ask the question: is there a difference in frames? We may use a simple difference operation, either globally or over a small window

$$\delta f(x, y, t) = \sum_{x,y} |f(x, y, t) - f(x, y, t-1)|$$

however this only tells us about total pixel intensity changes - we would like to obtain motion *vectors* on a pixel by pixel basis.

4.3 The motion field

A 2-D representation of a 3-D motion is called the *motion field*. Each pixel in the image has a velocity vector $\mathbf{v} = (v_x, v_y)^T$. This is the motion that we wish

to estimate and use for moving object evaluation.

We first consider some different motions.

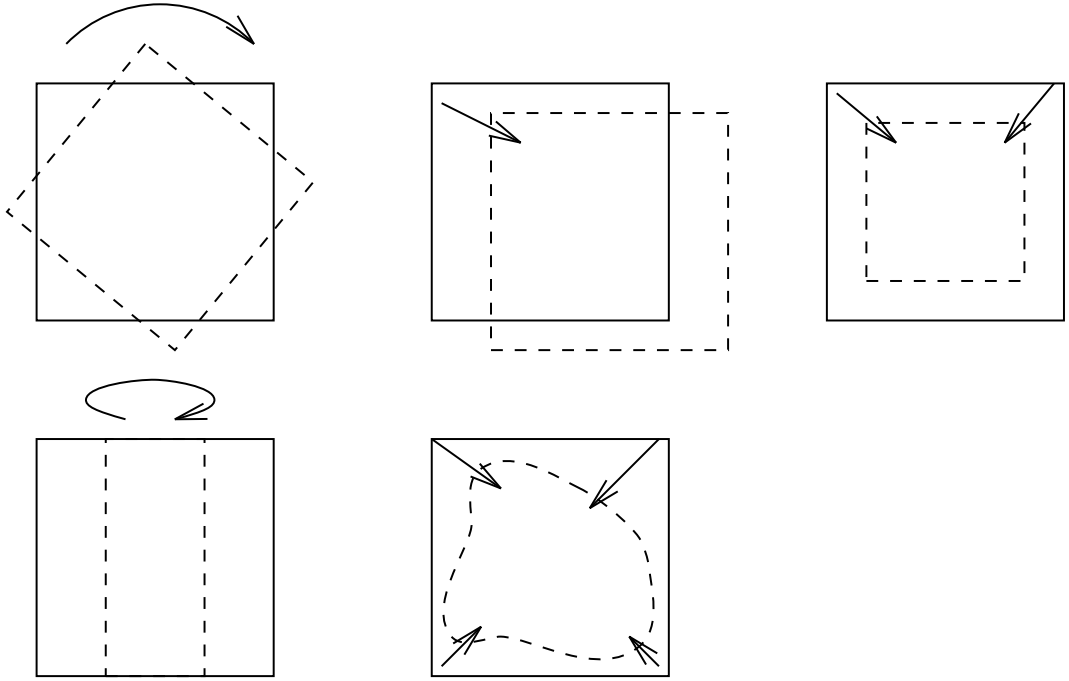


Figure 4.5: *Basic motion configurations.*

Note that twisting and scaling both resemble deformation in that the object appears to be non-rigid. This makes them more difficult to handle. To simplfy much motion analysis a set of *motion assumptions* are often made.

1. Motion has a maximum velocity
2. Small (negligable) accelerations occur over the interval dt
3. Common motion – all points in the same object move in the same way
4. Mutual correpondence – objects remain rigid

One popular method of motion field estimation is *optic flow*

Optic flow

- Assumes we have access to frames with a small dt
- Can give false information e.g. spinning sphere or changing illumination
- Assumes, therefore, that the illumination is *constant* so the observed brighness of objects is constant over t
- Nearby points in the motion field have similar motion – i.e. the velocity field is *smooth*

Let $f(x, y, t)$ be the grey-level at pixel x, y at time t. Consider the expansion of partial derivatives

$$df = f_x dx + f_y dy + f_t dt$$

if f remains a constant function (brightness) then $df \approx 0$ whence

$$-f_t = f_x v_x + f_y v_y = \nabla f \cdot \mathbf{v}$$

The most commonly used method to solve for the velocities is the *Lucas-Kanade* method which assumes places a small neighbourhood around the pixel of interest, in which it is assumed that the *motion vector* is stationary. If we define the set $\{p_1, p_2, ..., p_n\}$ as the pixels in the neighbourhood then

$$\begin{aligned} f_x(p_1)v_x + f_y(p_1)v_y &= -f_t(p_1) \\ f_x(p_2)v_x + f_y(p_2)v_y &= -f_t(p_2) \\ &\vdots \quad \vdots \\ f_x(p_n)v_x + f_y(p_n)v_y &= -f_t(p_n) \end{aligned}$$

which can be written in the form

$$\mathbf{Av} = \mathbf{b}$$

with solution given as

$$\mathbf{v} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$$

using the pseudo inverse of **A**.

Alternate approaches, such as the *Horne-Schunck* method assume that the motion field is *smooth*. If we add the *smoothness* constraint from the motion assuptions, then we may seek to minimize an 'energy' function given by

$$E(x, y, t) = (f_x v_x + f_y v_y + f_t)^2 + \lambda^2(|\nabla v_x|^2 + |\nabla v_y|^2)$$

where λ is the smoothness multiplier of the system. We may reduce this to solving a set of simultaneous differential equations

$$f_x^2 v_x + f_x f_y v_y + f_x f_t - \lambda^2 \Delta v_x = 0$$

$$f_y f_x v_x + f_y^2 v_y + f_y f_t - \lambda^2 \Delta v_y = 0$$

in which Δ is the *Laplacian operator*,

$$\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}.$$

The choice of λ changes the importance we give to the criterion of velocity smoothness in the optical flow estimation. If we allow λ to be small we are liable to find optic flow in textured areas as the motion assuptions inevitably fail. What we wish, therefore, to estimate the smoothest velocity field consistent with the motion. The 'optimal' value of λ may be found by trial and error or by a more

principled approach.

OF in motion analysis

We will assume that perceived motion in the image may arise from four mechanisms

1. Translation perpendicular to the line of sight
2. Scaling – translation along the line of sight
3. Rotation in plane perpendicular to line of sight
4. Rotation in line of sight plane

Note that we assume a rigid body (no deformation).

Optic flow can analyse these four motions

1. A set of parallel velocity vectors
2. A set of vectors with a common focal point
3. A set of concentric vectors
4. Sets of vectors anti-parallel to one another

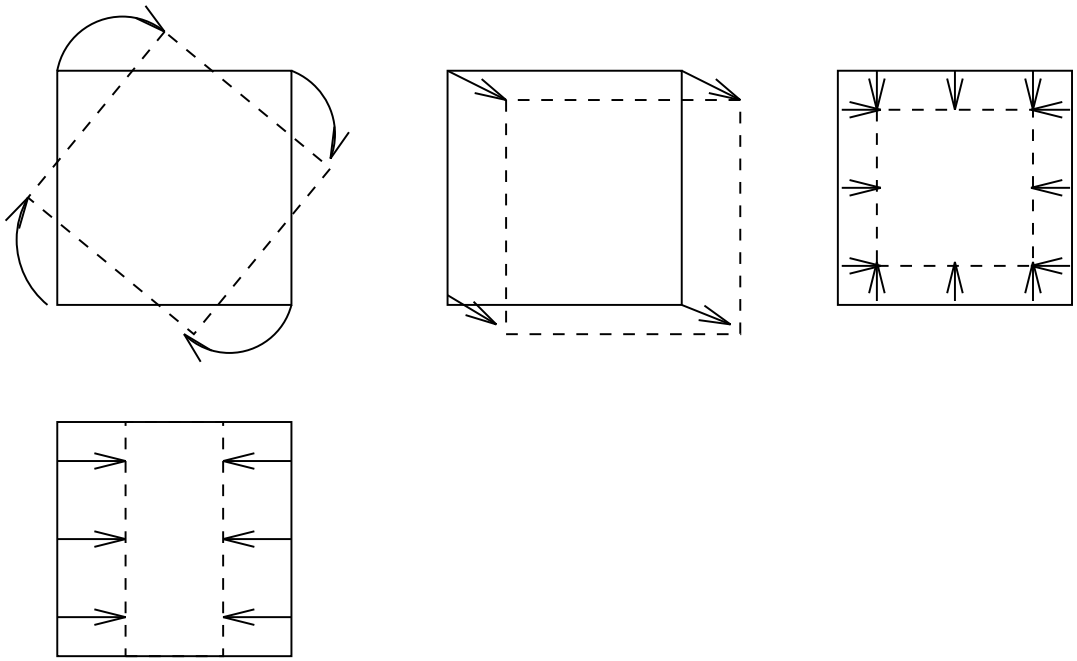


Figure 4.6: *Four basic motions detectable by optic flow.*

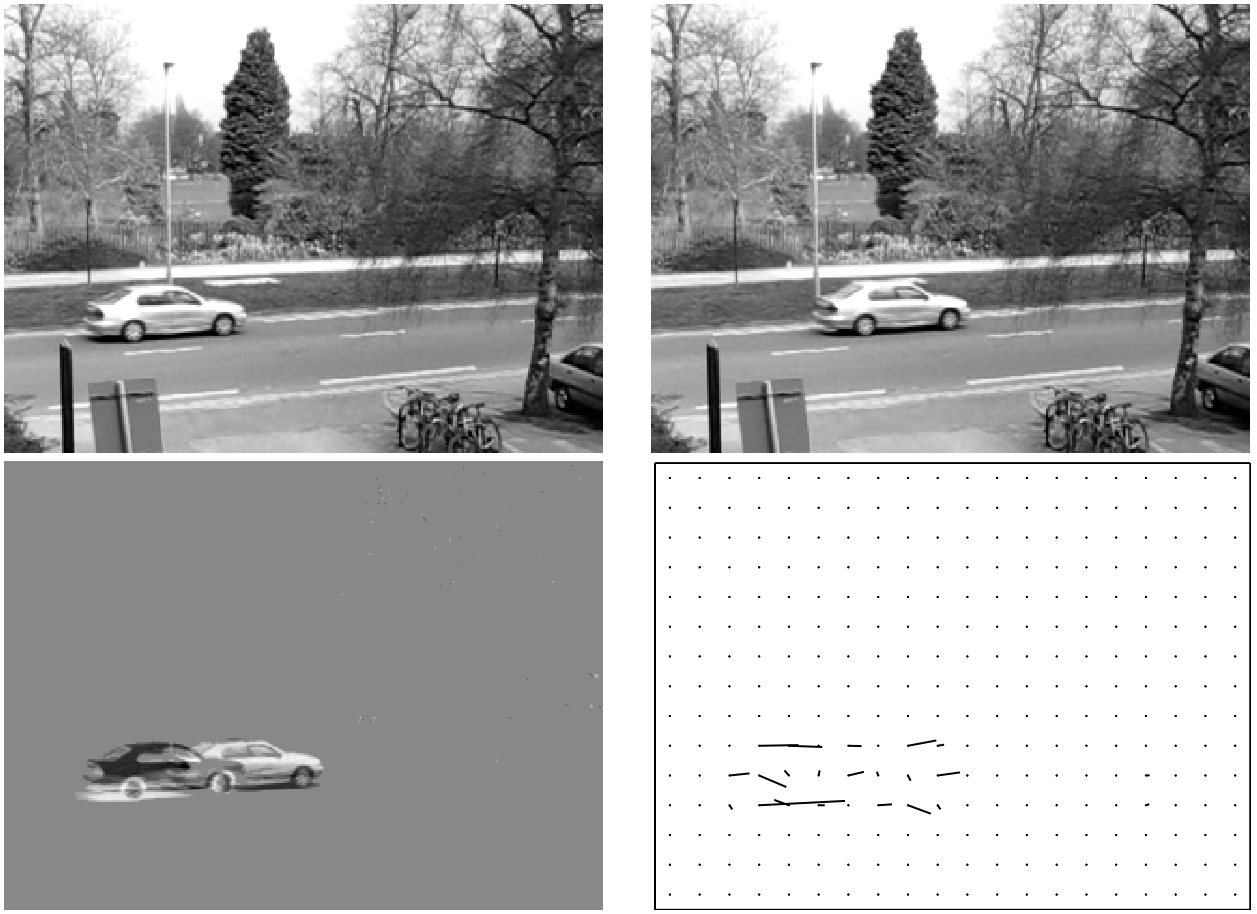


Figure 4.7: *Example of optical flow applied to car motionm using the Lucas-Kanade method.*

Lecture 5 - Stochastic Models

5.1 Introduction

We now discuss autocorrelation and autoregressive processes; that is, the correlation between successive values of a time series and the linear relations between them. We also show how these models can be used for spectral estimation.

5.2 Autocorrelation

Given a time series x_t we can produce a *lagged* version of the time series x_{t-T} which *lags* the original by T samples. We can then calculate the covariance between the two signals

$$\sigma_{xx}(T) = \frac{1}{N-1}\sum_{t=1}^{N}(x_{t-T} - \mu_x)(x_t - \mu_x) \tag{5.1}$$

where μ_x is the signal mean and there are N samples. We can then plot $\sigma_{xx}(T)$ as a function of T. This is known as the *autocovariance* function. The *autocorrelation* function is a normalised version of the autocovariance

$$r_{xx}(T) = \frac{\sigma_{xx}(T)}{\sigma_{xx}(0)} \tag{5.2}$$

Note that $\sigma_{xx}(0) = \sigma_x^2$. We also have $r_{xx}(0) = 1$. Also, because $\sigma_{xy} = \sigma_{yx}$ we have $r_{xx}(T) = r_{xx}(-T)$; the autocorrelation (and autocovariance) are *symmetric* functions or *even functions*. Figure 5.1 shows a signal and a lagged version of it and Figure 5.2 shows the autocorrelation function.

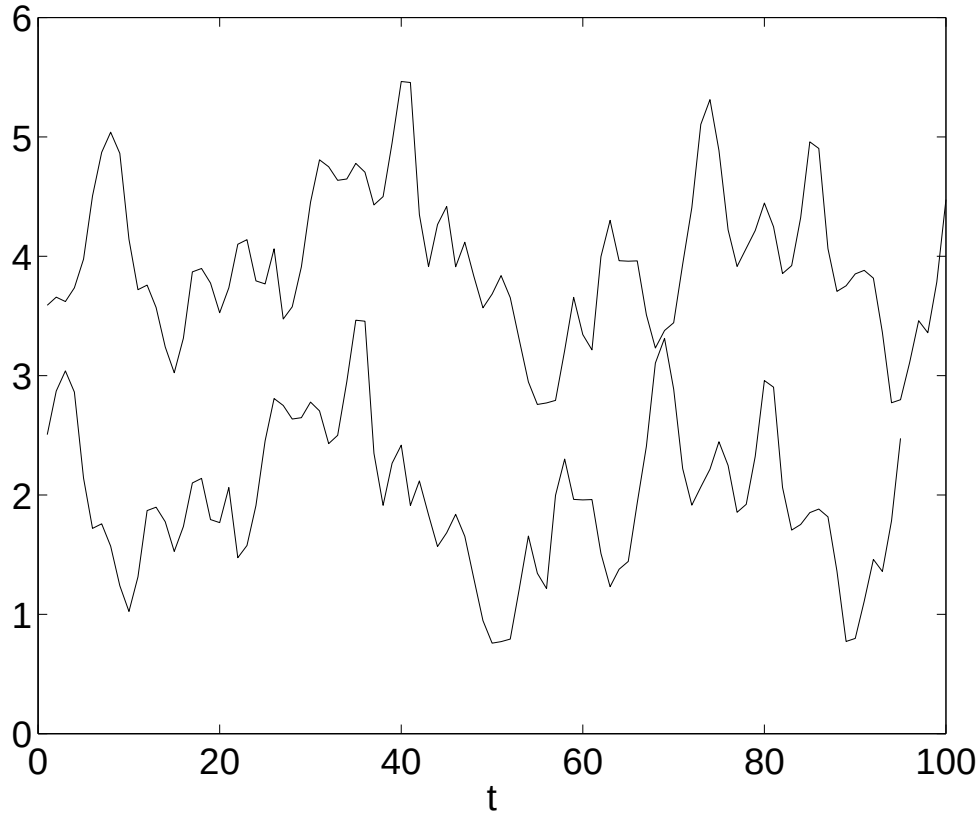
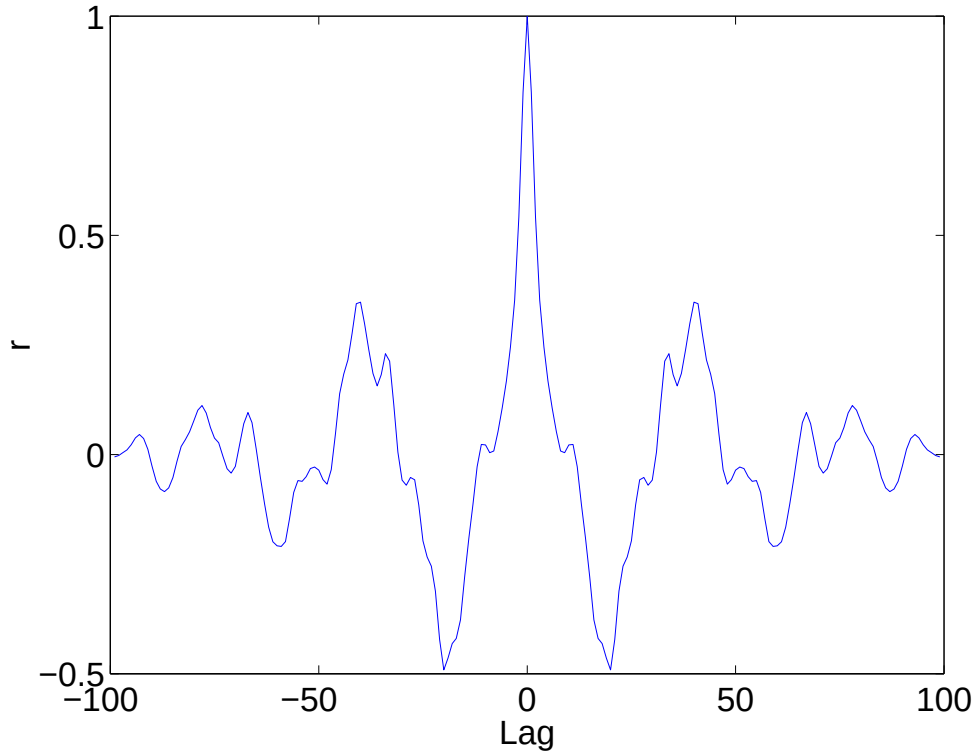


Figure 5.1: *Signal x_t (top) and x_{t+5} (bottom). The bottom trace* **leads** *the top trace by 5 samples. Or we may say it* **lags** *the top by -5 samples.*



(a)

Figure 5.2: *Autocorrelation function for x_t. Notice the negative correlation at lag 20 and positive correlation at lag 40. Can you see from Figure 5.1 why these should occur?*

5.3 Autoregressive models

An autoregressive (AR) model *predicts* the value of a time series from previous values. A pth order AR model is defined as

$$x_t = \sum_{i=1}^{p} x_{t-i} a_i + e_t \tag{5.3}$$

where a_i are the *AR coefficients* and e_t is the prediction error. These errors are assumed to be Gaussian with zero-mean and variance σ_e^2. It is also possible to include an extra parameter a_0 to soak up the mean value of the time series. Alternatively, we can first subtract the mean from the data and then apply the zero-mean AR model described above. We would also subtract any trend from the data (such as a linear or exponential increase) as the AR model assumes stationarity.

The above expression shows the relation for a single time step. To show the relation for all time steps we can use matrix notation.

We can write the AR model in matrix form by making use of the *embedding matrix*, $\mathbf{M}$, and by writing the signal and AR coefficients as vectors. We now

illustrate this for $p = 4$. This gives

$$\mathbf{M} = \begin{bmatrix} x_4 & x_3 & x_2 & x_1 \\ x_5 & x_4 & x_3 & x_2 \\ .. & .. & .. & .. \\ x_{N-1} & x_{N-2} & x_{N-3} & x_{N-4} \end{bmatrix} \tag{5.4}$$

We can also write the AR coefficients as a vector $\mathbf{a} = [a_1, a_2, a_3, a_4]^T$, the errors as a vector $\mathbf{e} = [e_5, e_6, ..., e_N]^T$ and the signal itself as a vector $\mathbf{X} = [x_5, x_6, ..., x_N]^T$. This gives

$$\begin{bmatrix} x_5 \\ x_6 \\ .. \\ x_N \end{bmatrix} = \begin{bmatrix} x_4 & x_3 & x_2 & x_1 \\ x_5 & x_4 & x_3 & x_2 \\ .. & .. & .. & .. \\ x_{N-1} & x_{N-2} & x_{N-3} & x_{N-4} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} + \begin{bmatrix} e_5 \\ e_6 \\ .. \\ e_N \end{bmatrix} \tag{5.5}$$

which can be compactly written as

$$\mathbf{X} = \mathbf{M}\mathbf{a} + \mathbf{e} \tag{5.6}$$

The AR model is therefore a special case of the multivariate regression model (compare the above equation to that given in the second lecture). The AR coefficients can therefore be computed from the equation

$$\hat{\mathbf{a}} = (\mathbf{M}^T\mathbf{M})^{-1}\mathbf{M}^T\mathbf{X} \tag{5.7}$$

The AR predictions can then be computed as the vector

$$\hat{\mathbf{X}} = \mathbf{M}\hat{\mathbf{a}} \tag{5.8}$$

and the error vector is then $\mathbf{e} = \mathbf{X} - \hat{\mathbf{X}}$. The variance of the noise is then calculated as the variance of the error vector.

To illustrate this process we analyse our data set using an AR(4) model. The AR coefficients were estimated to be

$$\hat{\mathbf{a}} = [1.46, -1.08, 0.60, -0.186]^T \tag{5.9}$$

and the AR predictions are shown in Figure 5.3. The noise variance was estimated to be $\sigma_e^2 = 0.079$ which corresponds to a standard deviation of 0.28. The variance of the original time series was 0.3882 giving a signal to noise ratio of $(0.3882 - 0.079)/0.079 = 3.93$.

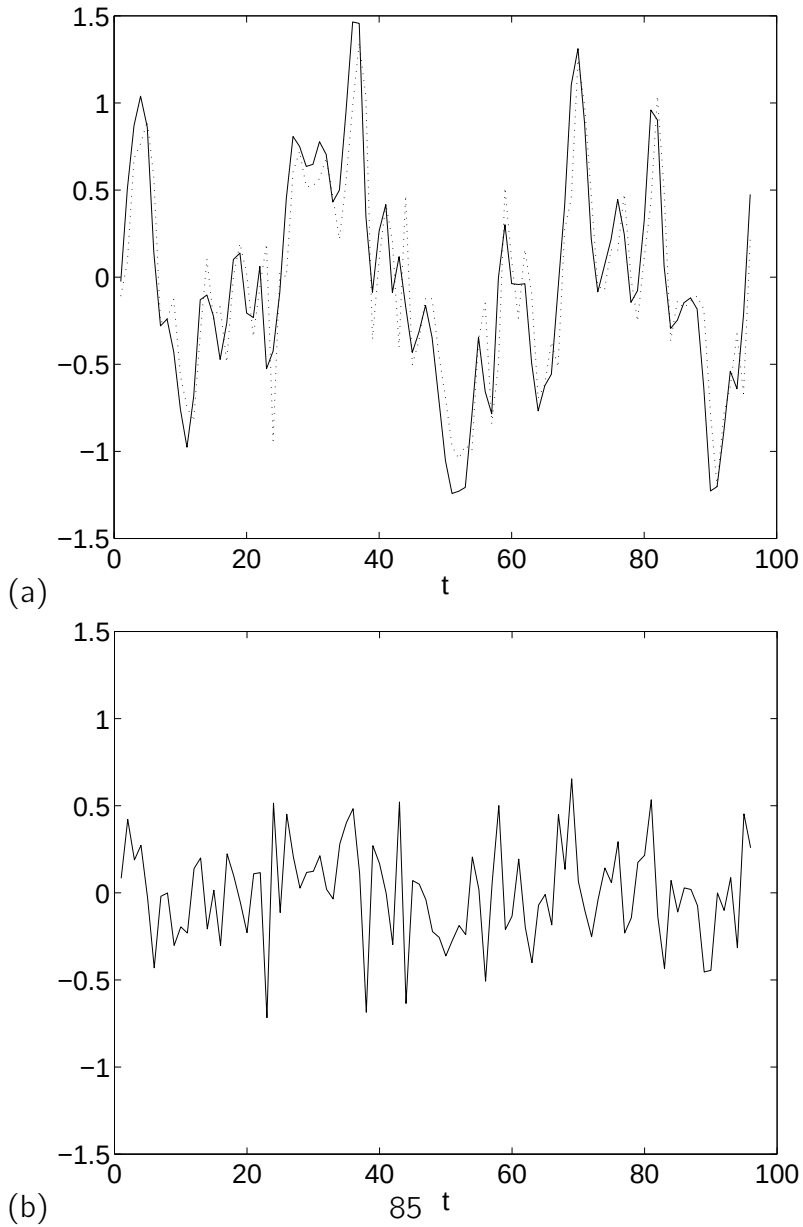


Figure 5.3: *(a) Original signal (solid line),* $\mathbf{X}$*, and predictions (dotted line),* $\hat{\mathbf{X}}$*, from an AR(4) model and (b) the prediction errors,* $\mathbf{e}$*. Notice that the variance of the errors is much less than that of the original signal.*

5.3.1 Random walks

If $p = 1$ and $a_1 = 1$ then the AR model reduces to a random walk model, an example of which is shown in Figure 5.4.

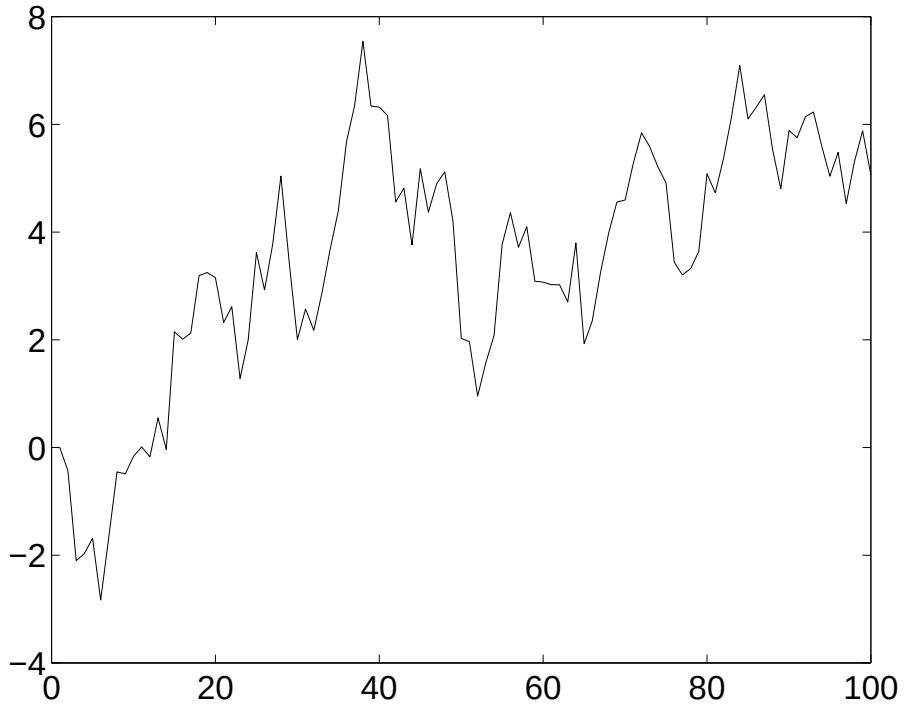


Figure 5.4: *A random walk.*

5.3.2 Relation to autocorrelation

The autoregressive model can be written as

$$x_t = a_1 x_{t-1} + a_2 x_{t-2} + ... + a_p x_{t-p} + e_t \tag{5.10}$$

If we multiply both sides by x_{t-k} we get

$$x_t x_{t-k} = a_1 x_{t-1} x_{t-k} + a_2 x_{t-2} x_{t-k} + ... + a_p x_{t-p} x_{t-k} + e_t x_{t-k} \tag{5.11}$$

If we now sum over t and divide by $N-1$ and assume that the signal is zero mean (if it isn't we can easily make it so, just by subtracting the mean value from every sample) the above equation can be re-written in terms of covariances at different lags

$$\sigma_{xx}(k) = a_1 \sigma_{xx}(k-1) + a_2 \sigma_{xx}(k-2) + ... + a_p \sigma_{xx}(k-p) + \sigma_{e,x} \tag{5.12}$$

where the last term $\sigma_{e,x}$ is the covariance between the noise and the signal. But as the noise is assumed to be independent from the signal $\sigma_{e,x} = 0$. If we now divide every term by the signal variance we get a relation between the correlations at different lags

$$r_{xx}(k) = a_1 r_{xx}(k-1) + a_2 r_{xx}(k-2) + ... + a_p r_{xx}(k-p) \tag{5.13}$$

This holds for all lags. For an AR(p) model we can write this relation out for the first p lags. For $p = 4$

$$\begin{bmatrix} r_{xx}(1) \\ r_{xx}(2) \\ r_{xx}(3) \\ r_{xx}(4) \end{bmatrix} = \begin{bmatrix} r_{xx}(0) & r_{xx}(-1) & r_{xx}(-2) & r_{xx}(-3) \\ r_{xx}(1) & r_{xx}(0) & r_{xx}(-1) & r_{xx}(-2) \\ r_{xx}(2) & r_{xx}(1) & r_{xx}(0) & r_{xx}(-1) \\ r_{xx}(3) & r_{xx}(2) & r_{xx}(1) & r_{xx}(0) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} \tag{5.14}$$

which can be compactly written as

$$\mathbf{r} = \mathbf{R}\mathbf{a} \tag{5.15}$$

where $\mathbf{r}$ is the autocorrelation vector and $\mathbf{R}$ is the autocorrelation matrix. The above equations are known, after their discoverers, as the Yule-Walker relations. They provide another way to estimate AR coefficients

$$\mathbf{a} = \mathbf{R}^{-1}\mathbf{r} \tag{5.16}$$

This leads to a more efficient algorithm than the general method for multivariate linear regression (equation 5.7) because we can exploit the structure in the autocorrelation matrix. By noting that $r_{xx}(k) = r_{xx}(-k)$ we can rewrite the

correlation matrix as

$$\mathbf{R} = \begin{bmatrix} 1 & r_{xx}(1) & r_{xx}(2) & r_{xx}(3) \\ r_{xx}(1) & 1 & r_{xx}(1) & r_{xx}(2) \\ r_{xx}(2) & r_{xx}(1) & 1 & r_{xx}(1) \\ r_{xx}(3) & r_{xx}(2) & r_{xx}(1) & 1 \end{bmatrix} \tag{5.17}$$

Because this matrix is both symmetric and a Toeplitz matrix (the terms along any diagonal are the same) we can use a recursive estimation technique known as the Levinson-Durbin algorithm.

5.3.3 Relation to partial autocorrelation

The partial correlation coefficients in an AR model are known as *reflection coefficients*. At lag m, the partial correlation between x_{t-m} and x_t, is written as k_m; the mth reflection coefficient. It can be calculated as the relative reduction in prediction error

$$k_m = \frac{E_{m-1} - E_m}{E_{m-1}} \tag{5.18}$$

where E_m is the prediction error from an $AR(m)$ model. The reflection coefficients are to the AR coefficients what the correlation is to the slope in a univariate

AR model; if the mth reflection coefficient is significantly non-zero then so is the mth AR coefficient and vice-versa. This enables an intelligent selection of the order of the model.

The Levinson-Durbin algorithm computes reflection coefficients as part of a recursive algorithm for computing the AR coefficients. It finds k_1 and from it calculates the AR coefficient for an $AR(1)$ model, a_1. It then computes k_2 and from it calculates the AR coefficients for an AR(2) model (a_2 is computed afresh and a_1 is re-estimated from a_1 for the $AR(1)$ model - as it will be different). The algorithm continues by calculating k_m and the coefficients for $AR(m)$ from $AR(m-1)$.

5.4 Moving Average Models

A Moving Average (MA) model of order q is defined as

$$x_t = \sum_{i=0}^{q} b_i e_{t-i} \tag{5.19}$$

where e_t is Gaussian random noise with zero mean and variance σ_e^2. They are a type of FIR filter. These can be combined with AR models to get Autoregressive

Moving Average (ARMA) models

$$x_t = \sum_{i=1}^{p} a_i x_{t-i} + \sum_{i=0}^{q} b_i e_{t-i} \tag{5.20}$$

which can be described as an ARMA(p,q) model. They are a type of IIR filter.

Usually, however, FIR and IIR filters have a set of fixed coefficients which have been chosen to give the filter particular frequency characteristics. In MA or ARMA modelling the coefficients are tuned to a particular time series so as to capture the spectral characteristics of the underlying process.

5.5 Spectral Estimation

Autoregressive models can also be used for spectral estimation. An $AR(p)$ model predicts the next value in a time series as a linear combination of the p previous values

$$x_t = -\sum_{k=1}^{p} a_k x_{t-k} + e_t \tag{5.21}$$

where a_k are the AR coefficients and e_t is IID Gaussian noise with zero mean and variance σ^2.

The above equation can be solved by using the z-transform. This allows the equation to be written as

$$a_p z^{t-p} + a_{p-1} z^{t-(p-1)} + \ldots + z^t = e_t \tag{5.22}$$

It can then be rewritten in terms of the pulse transfer function

$$H(z) = \frac{e_t}{1 + \sum_{k=1}^{p} a_k z^{-k}} \tag{5.23}$$

Given that any complex number can be written in exponential form

$$z = \exp(i 2\pi u T_s) \tag{5.24}$$

where u is frequency and T_s is the sampling period we can see that the frequency domain characteristics of an AR model are given by

$$P(f) = \frac{\sigma_e^2 T_s}{|1 + \sum_{k=1}^{p} a_k \exp(-ik 2\pi u T_s)|^2} \tag{5.25}$$

An AR(p) model can provide spectral estimates with $p/2$ peaks; therefore if you know how many peaks you're looking for in the spectrum you can define the AR model order. Alternatively, AR model order estimation methods should automatically provide the appropriate level of smoothing of the estimated spectrum.

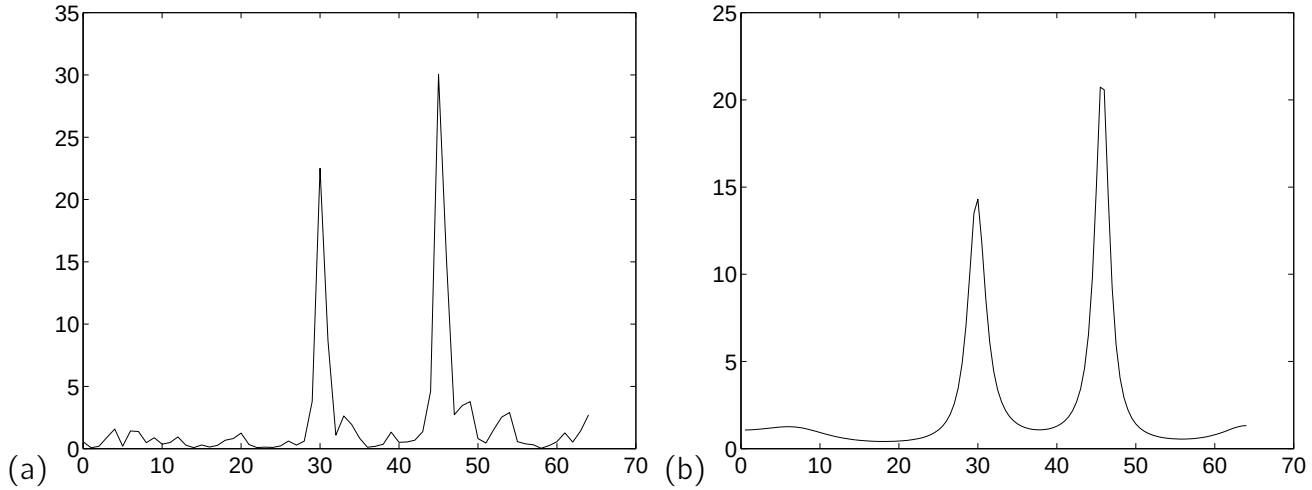


Figure 5.5: *Power spectral estimates of two sinwaves in additive noise using (a) Discrete Fourier transform method and (b) autoregressive spectral estimation.*

AR spectral estimation has two distinct advantages over methods based on the Fourier transform (i) power can be estimated over a continuous range of frequencies (not just at fixed intervals) and (ii) the power estimates have less variance.

Lecture 6 - Multi-variate systems

6.1 Introduction

We now consider the situation where we have a number of time series and wish to explore the relations between them. We first look at the relation between cross-correlation and multivariate autoregressive models and then at the cross-spectral density and coherence.

6.2 Cross-correlation

Given *two* time series x_t and y_t we can delay x_t by T samples and then calculate the *cross-covariance* between the pair of signals. That is

$$\sigma_{xy}(T) = \frac{1}{N-1}\sum_{t=1}^{N}(x_{t-T} - \mu_x)(y_t - \mu_y) \tag{6.26}$$

where μ_x and μ_y are the means of each time series and there are N samples in each. The function $\sigma_{xy}(T)$ is the *cross-covariance* function. The *cross-correlation* is a normalised version

$$r_{xy}(T) = \frac{\sigma_{xy}(T)}{\sqrt{\sigma_{xx}(0)\sigma_{yy}(0)}} \tag{6.27}$$

where we note that $\sigma_{xx}(0) = \sigma_x^2$ and $\sigma_{yy}(0) = \sigma_y^2$ are the variances of each signal. Note that

$$r_{xy}(0) = \frac{\sigma_{xy}}{\sigma_x \sigma_y} \tag{6.28}$$

which is the correlation between the two variables. Therefore unlike the autocorrelation, r_{xy} is not equal to 1 even at zero lag/lead. Figure 6.1 shows two time series and their cross-correlation.

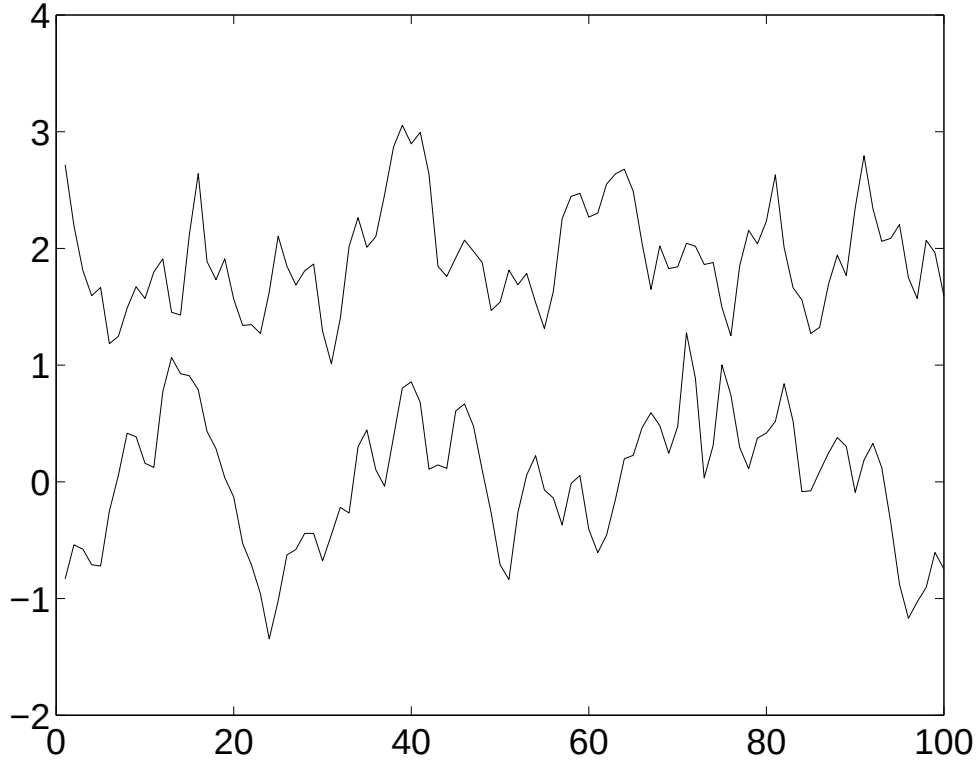


Figure 6.1: *Signals x_t (top) and y_t (bottom).*

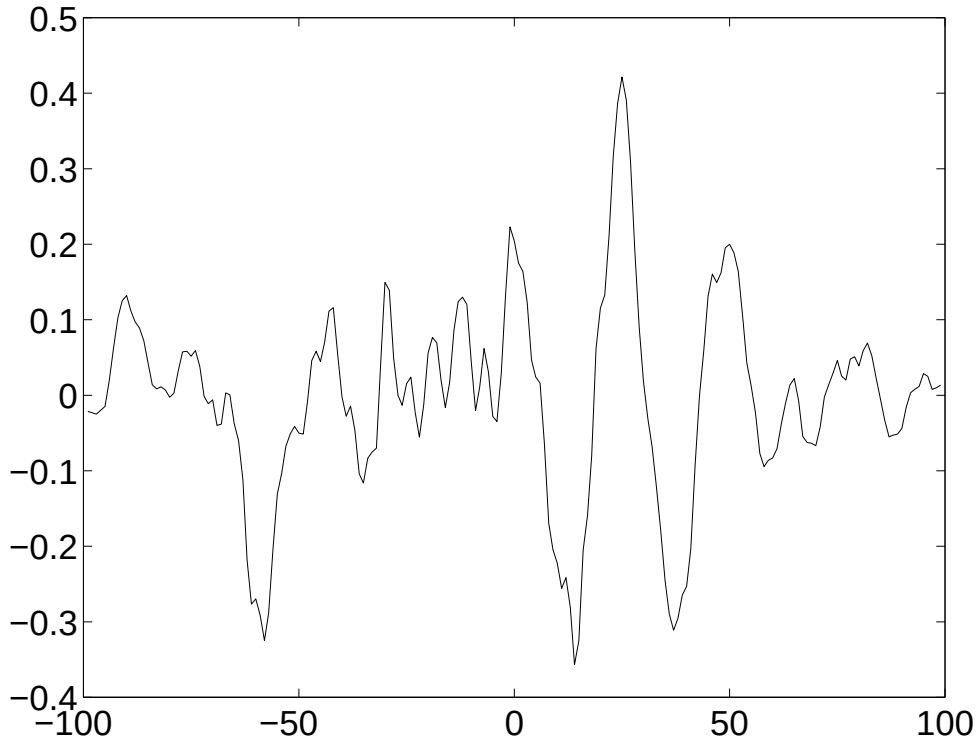


Figure 6.2: *Cross-correlation function $r_{xy}(T)$ for the data in Figure 6.1. A lag of T denotes the top series, x, lagging the bottom series, y. Notice the big positive correlation at a lag of 25. Can you see from Figure 6.1 why this should occur ?*

6.2.1 Cross-correlation is asymmetric

First, we re-cap as to why the auto-correlation is a *symmetric* function. The autocovariance, for a zero mean signal, is given by

$$\sigma_{xx}(T) = \frac{1}{N-1}\sum_{t=1}^{N} x_{t-T}x_t \tag{6.29}$$

This can be written in the shorthand notation

$$\sigma_{xx}(T) = \langle x_{t-T}x_t \rangle \tag{6.30}$$

where the angled brackets denote the average value or *expectation*. Now, for negative lags

$$\sigma_{xx}(-T) = \langle x_{t+T}x_t \rangle \tag{6.31}$$

Subtracting T from the time index (this will make no difference to the expectation) gives

$$\sigma_{xx}(-T) = \langle x_t x_{t-T} \rangle \tag{6.32}$$

which is identical to $\sigma_{xx}(T)$, as the ordering of variables makes no difference to the expected value. Hence, the autocorrelation is a symmetric function.

The cross-correlation is a normalised cross-covariance which, assuming zero mean signals, is given by

$$\sigma_{xy}(T) = \langle x_{t-T}y_t \rangle \tag{6.33}$$

and for negative lags

$$\sigma_{xy}(-T) = \langle x_{t+T} y_t \rangle \tag{6.34}$$

Subtracting T from the time index now gives

$$\sigma_{xy}(-T) = \langle x_t y_{t-T} \rangle \tag{6.35}$$

which is different to $\sigma_{xy}(T)$. To see this more clearly we can subtract T once more from the time index to give

$$\sigma_{xy}(-T) = \langle x_{t-T} y_{t-2T} \rangle \tag{6.36}$$

Hence, the cross-covariance, and therefore the cross-correlation, is an *asymmetric* function.

To summarise: moving signal A right (forward in time) and multiplying with signal B is not the same as moving signal A left and multiplying with signal B; unless signal A equals signal B.

6.2.2 Windowing

When calculating cross-correlations there are fewer data points at larger lags than at shorter lags. The resulting estimates are commensurately less accurate. To take account of this the estimates at long lags can be smoothed using various window operators.

6.2.3 Time-Delay Estimation

If we suspect that one signal is a, possibly noisy, time-delayed version of another signal then the peak in the cross-correlation will identify the delay. For example, figure 6.1 suggests that the top signal lags the bottom by a delay of 25 samples. Given that the sample rate is 125Hz this corresponds to a delay of 0.2 seconds.

6.3 Multivariate Autoregressive models

A multivariate autoregressive (MAR) model is a linear predictor used for modelling multiple time series. An $MAR(p)$ model predicts the next vector value in a d-dimensional time series, $\mathbf{x}_t$ (a *row* vector) as a linear combination of the p previous vector values of the time series

$$\mathbf{x}(t) = \sum_{k=1}^{p} \mathbf{x}(t-k)\mathbf{a}(k) + \mathbf{e}_t \tag{6.37}$$

where each $\mathbf{a}_k$ is a $d - by - d$ matrix of AR coefficients and $\mathbf{e}_t$ is an IID Gaussian noise vector with zero mean and covariance $\mathbf{C}$. There are a total of $n_p = p \times d \times d$ AR coefficients and the noise covariance matrix has $d \times d$ elements. If we write

the lagged vectors as a single augmented row vector

$$\tilde{\mathbf{x}}(t) = [\mathbf{x}(t-1), \mathbf{x}(t-2), ..., \mathbf{x}(t-p)] \tag{6.38}$$

and the AR coefficients as a single augmented matrix

$$\mathbf{A} = [\mathbf{a}(1), \mathbf{a}(2), ..., \mathbf{a}(p)]^T \tag{6.39}$$

then we can write the MAR model as

$$\mathbf{x}(t) = \tilde{\mathbf{x}}(t)\mathbf{A} + \mathbf{e}(t) \tag{6.40}$$

The above equation shows the model at a single time point t.

The equation for the model over all time steps can be written in terms of the embedding matrix, $\tilde{\mathbf{M}}$, whose tth row is $\tilde{\mathbf{x}}(t)$, the error matrix $\mathbf{E}$ having rows $\mathbf{e}(t+p+1)$ and the target matrix $\mathbf{X}$ having rows $\mathbf{x}(t+p+1)$. This gives

$$\mathbf{X} = \tilde{\mathbf{M}}\mathbf{A} + \mathbf{E} \tag{6.41}$$

which is now in the standard form of a multivariate linear regression problem. The AR coefficients can therefore be calculated from

$$\hat{\mathbf{A}} = \left(\tilde{\mathbf{M}}^T\tilde{\mathbf{M}}\right)^{-1}\tilde{\mathbf{M}}^T\mathbf{X} \tag{6.42}$$

and the AR predictions are then given by

$$\hat{\mathbf{x}}(t) = \tilde{\mathbf{x}}(t)\hat{\mathbf{A}} \tag{6.43}$$

The predicion errors are

$$\mathbf{e}(t) = \mathbf{x}(t) - \hat{\mathbf{x}}(t) \tag{6.44}$$

and the noise covariance matrix is estimated as

$$\mathbf{C} = \frac{1}{N - n_p}\mathbf{e}^T(t)\mathbf{e}(t) \tag{6.45}$$

The denominator $N - n_p$ arises because n_p degrees of freedom have been used up to calculate the AR coefficients (and we want the estimates of covariance to be unbiased).

6.3.1 Example

Given two time series and a MAR(3) model, for example, the MAR predictions are

$$\hat{\mathbf{x}}(t) = \tilde{\mathbf{x}}(t)\mathbf{A}$$

$$\hat{\mathbf{x}}(t) = [\mathbf{x}(t-1), \mathbf{x}(t-2), \mathbf{x}(t-3)] \begin{bmatrix} \mathbf{a}(1) \\ \mathbf{a}(2) \\ \mathbf{a}(3) \end{bmatrix}$$

$$\left[\ \hat{x}_1(t)\hat{x}_2(t)\ \right] \tag{6.46}$$

$$= \left[\ x_1(t-1), x_2(t-1), x_1(t-2), x_2(t-2), x_1(t-3), x_2(t-3)\ \right] \begin{bmatrix} \hat{a}_{11}(1) & \hat{a}_{12}(1) \\ \hat{a}_{21}(1) & \hat{a}_{22}(1) \\ \hat{a}_{11}(2) & \hat{a}_{12}(2) \\ \hat{a}_{21}(2) & \hat{a}_{22}(2) \\ \hat{a}_{11}(3) & \hat{a}_{12}(3) \\ \hat{a}_{21}(3) & \hat{a}_{22}(3) \end{bmatrix}$$

Applying an MAR(3) model to our data set gave the following estimates for the AR coefficients, $\mathbf{a}_p$, and noise covariance $\mathbf{C}$, which were estimated from equations 6.42 and 6.45

$$\mathbf{a}_1 = \begin{bmatrix} -1.2813 & -0.2394 \\ -0.0018 & -1.0816 \end{bmatrix}$$

$$\mathbf{a}_2 = \begin{bmatrix} 0.7453 & 0.2822 \\ -0.0974 & 0.6044 \end{bmatrix}$$

$$\mathbf{a}_3 = \begin{bmatrix} -0.3259 & -0.0576 \\ -0.0764 & -0.2699 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 0.0714 & 0.0054 \\ 0.0054 & 0.0798 \end{bmatrix}$$

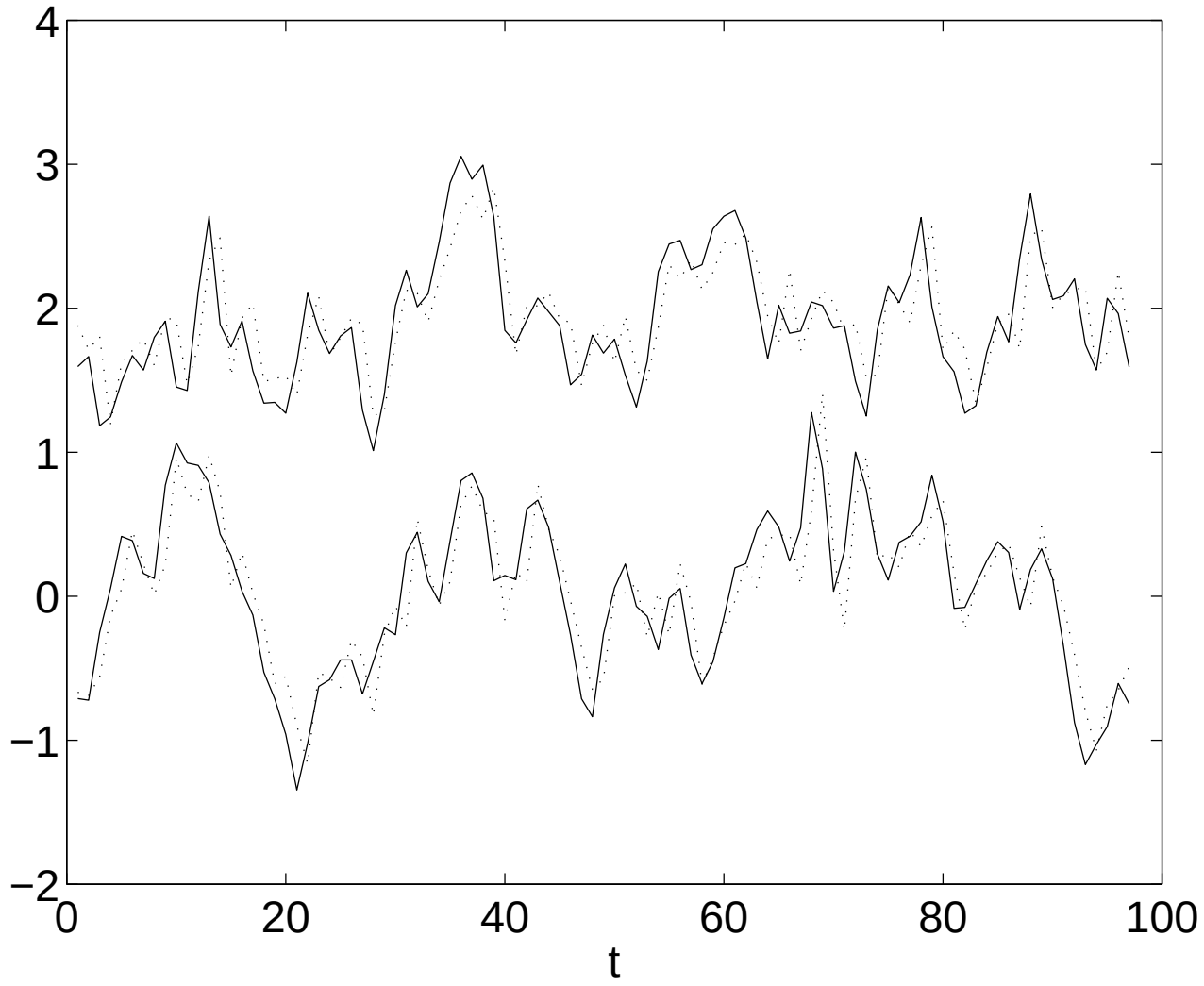


Figure 6.3: *Signals $x_1(t)$ (top) and $x_2(t)$ (bottom) and predictions from MAR(3) model.*

6.4 Cross Spectral Density

Just as the Power Spectral Density (PSD) is the Fourier transform of the auto-covariance function we may define the Cross Spectral Density (CSD) as the Fourier transform of the cross-covariance function

$$P_{12}(\omega) = \sum_{n=-\infty}^{\infty} \sigma_{x_1 x_2}(n) \exp(-i\omega n) \tag{6.47}$$

Note that if $x_1 = x_2$, the CSD reduces to the PSD. Now, the cross-covariance of a signal is given by

$$\sigma_{x_1 x_2}(n) = \sum_{l=-\infty}^{\infty} x_1(l) x_2(l-n) \tag{6.48}$$

Substituting this into the earlier expression gives

$$P_{12}(\omega) = \sum_{n=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} x_1(l) x_2(l-n) \exp(-i\omega n) \tag{6.49}$$

By noting that

$$\exp(-i\omega n) = \exp(-i\omega l) \exp(i\omega k) \tag{6.50}$$

where $k = l - n$ we can see that the CSD splits into the product of two integrals

$$P_{12}(\omega) = X_1(\omega)X_2(-\omega) \tag{6.51}$$

where

$$X_1(\omega) = \sum_{l=-\infty}^{\infty} x_1(l)\exp(-i\omega l) \tag{6.52}$$

$$X_2(-w) = \sum_{k=-\infty}^{\infty} x_2(k)\exp(+i\omega k)$$

For real signals $X_2^*(\omega) = X_2(-\omega)$ where * denotes the complex conjugate. Hence, the cross spectral density is given by

$$P_{12}(\omega) = X_1(\omega)X_2^*(\omega) \tag{6.53}$$

This means that the CSD can be evaluated in one of two ways (i) by first estimating the cross-covariance and Fourier transforming or (ii) by taking the Fourier transforms of each signal and multiplying (after taking the conjugate of one of them). A number of algorithms exist which enhance the spectral estimation ability of each method.

The CSD is complex

The CSD is complex because the cross-covariance is asymmetric (the PSD is real because the auto-covariance is symmetric; in this special case the Fourier transorm reduces to a cosine transform).

6.4.1 More than two time series

The frequency domain characteristics of a multivariate time-series may be summarised by the power spectral density *matrix*. For d time series

$$\mathbf{P}(\omega) = \begin{pmatrix} P_{11}(\omega) & P_{12}(\omega) & \cdots & P_{1d}(\omega) \\ P_{21}(\omega) & P_{22}(\omega) & \cdots & P_{2d}(\omega) \\ \cdots & \cdots & \cdots & \cdots \\ P_{d1}(\omega) & P_{d2}(\omega) & \cdots & P_{dd}(\omega) \end{pmatrix} \tag{6.54}$$

where the diagonal elements contain the spectra of individual channels and the off-diagonal elements contain the cross-spectra. The matrix is called a *Hermitian matrix* because the elements are complex numbers.

6.4.2 Coherence and Phase

The *complex coherence function* is given by

$$r_{ij}(\omega) = \frac{P_{ij}(\omega)}{\sqrt{P_{ii}(\omega)}\sqrt{P_{jj}(\omega)}} \tag{6.55}$$

The coherence, or *magnitude squared coherence* (MSC), between two channels is given by

$$\text{MSC}_{ij}(\omega) = \mid r_{ij}(\omega) \mid^2 \tag{6.56}$$

The phase spectrum, between two channels is given by

$$\theta_{ij}(\omega) = \tan^{-1}\left[\frac{Im(r_{ij}(\omega))}{Re(r_{ij}(\omega))}\right] \tag{6.57}$$

The MSC measures the linear correlation between two time series at each frequency and is directly analagous to the squared correlation coefficient in linear regression. As such the MSC is intimately related to *linear filtering*, where one signal is viewed as a filtered version of the other. This can be interpreted as a linear regression at each frequency. The optimal regression coefficient, or linear filter, is given by

$$H(\omega) = \frac{P_{xy}(\omega)}{P_{xx}(\omega)} \tag{6.58}$$

This is analagous to the expression for the regression coefficient $a = \sigma_{xy}/\sigma_{xx}$. The MSC is related to the optimal filter as follows

$$r^2_{xy}(\omega) = |H(\omega)|^2 \frac{P_{xx}(\omega)}{P_{yy}(\omega)} \tag{6.59}$$

which is analagous to the equivalent expression in linear regression $r^2 = a^2(\sigma_{xx}/\sigma_{yy})$.

At a given frequency, if the phase of one signal is *fixed* relative to the other, then the signals can have a high coherence at that frequency. This holds even if one signal is entirely out of phase with the other (note that this is different from adding up signals which are out of phase; the signals cancel out. We are talking about the coherence *between* the signals).

At a given frequency, if the phase of one signal changes relative to the other then the signals will not be coherent at that frequency. The time over which the phase relationship is constant is known as the *coherence time*.

6.4.3 Welch's method for estimating coherence

Algorithms based on Welch's method (such as the cohere function in the matlab system identification toolbox) are widely used. The signal is split up into a number of segments, N, each of length T and the segments may be overlapping. The

complex coherence estimate is then given as

$$\hat{r}_{ij}(\omega) = \frac{\sum_{n=1}^{N} X_i^n(\omega)(X_j^n(\omega))^*}{\sqrt{\sum_{n=1}^{N} X_i^n(\omega)^2}\sqrt{\sum_{n=1}^{N} X_j^n(\omega)^2}} \tag{6.60}$$

where n sums over the data segments. This equation is exactly the same form as for estimating correlation coefficients. Note that if we have only $N = 1$ data segment then the estimate of coherence will be 1 regardless of what the true value is (this would be like regression with a single data point). Therefore, we need a number of segments.

Note that this only applies to Welch-type algorithms which compute the CSD from a product of Fourier transforms. We can trade-off good spectral resolution (requiring large T) with low-variance estimates of coherence (requiring large N and therefore small T). To an extent, by increasing the overlap between segments (and therefore the amount of computation, ie. number of FFTs computed) we can have the best of both worlds.

6.4.4 Multivariate Autoregressive (MAR) models

Just as the PSD can be calculated from AR coefficients so the PSD's and CSD's can be calculated from MAR coefficients. First we compute

$$\mathbf{A}(\omega) = \mathbf{I} + \sum_{k}^{p} \mathbf{a}_k \exp(-ik\omega T) \tag{6.61}$$

where $\mathbf{I}$ is the identity matrix, ω is the frequency of interest and T is the sampling period. $\mathbf{A}(\omega)$ will be complex. This is analagous to the denominator in the equivalent AR expression $(1 + \sum_{k=1}^{p} a_k \exp(-ik\omega t))$. Then we calculate the PSD matrix as follows

$$\mathbf{P}_{MAR}(\omega) = T \left[\mathbf{A}(\omega)\right]^{-1} \mathbf{C} \left[\mathbf{A}(\omega)^{*T}\right]^{-1} \tag{6.62}$$

where $\mathbf{C}$ is the residual covariance matrix. Once the PSD matrix has been calculated, we can calculate the coherences of interest using equation 6.56.

6.5 Example

To illustrate the estimation of coherence we generated two signals. The first, x, being a 10Hz sine wave with additive Gaussian noise of standard deviation

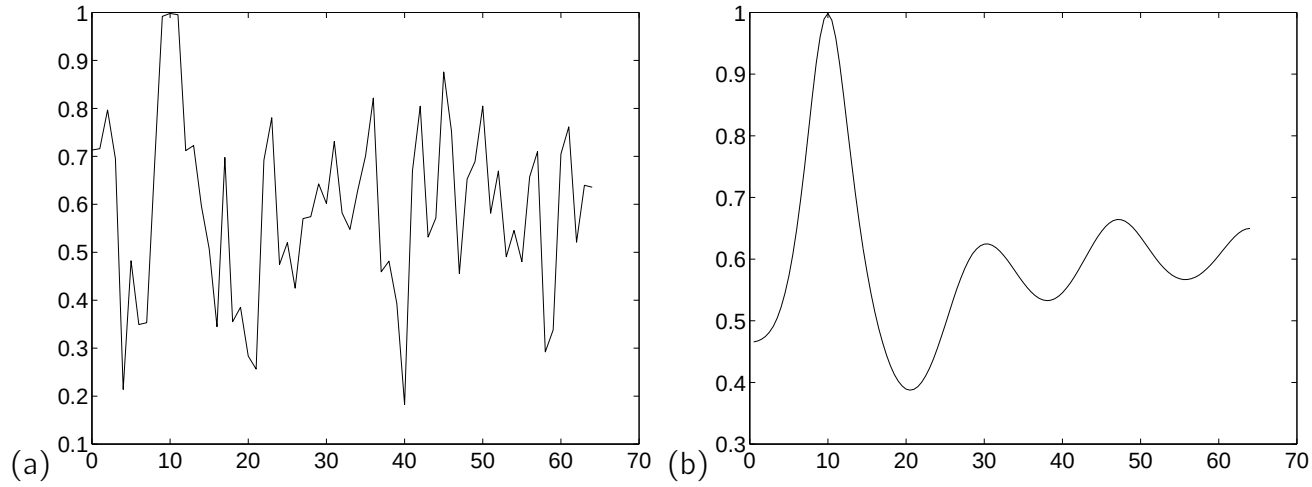


Figure 6.4: *Coherence estimates from (a) Fourier transform method and (b) Multivariate Autoregressive model.*

0.3 and the second y being equal to the first but with more additive noise of the same standard deviation. Five seconds of data were generated at a sample rate of 128Hz. We then calculated the coherence using (a) Welch's modified periodogram method with $N = 128$ samples per segment and a 50% overlap between segments and smoothing via a Hanning window and (b) an MAR(8) model.

Ideally, we should see a coherence near to 1 at 10Hz and zero elsewhere. However, the coherence is highly non-zero at other frequencies. This is because due to the noise component of the signal there is power (and some cross-power) at all frequencies. As coherence is a ratio of cross-power to power it will have a high variance unless the number of data samples is large. You should therefore be careful when interpreting coherence values.

Lecture 7 – Non-linear filters

7.1 Order statistic filters – the median filter

Consider a set of p digital samples from $f[n-p+1] : f[n]$. Order the set from smallest to largest, and let this sample set be $z[1] : z[p]$. Order statistic filters thence perform linear filtering on this ordered set, rather than the original.

What use might this be? Consider a signal, corrupted by spiking noise, as in Fig 7.1.

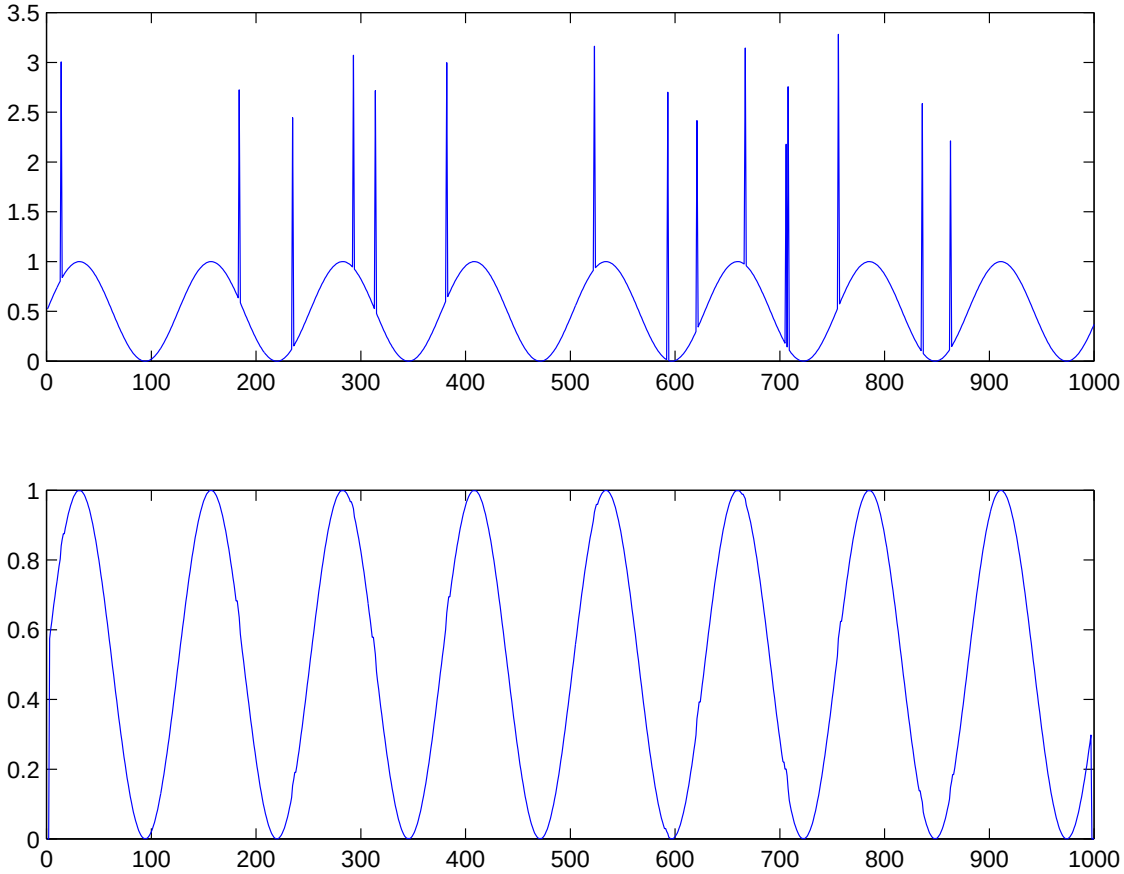


Figure 7.1: *(top) Spiky signal. (bottom) median filtered output,* $p = 5$

We see that the signal (a sine wave) is corrupted by *impulsive noise*. The effect of this noise is that the maximal and minimal values we observe in any small window on the data are likely to be caused by the noise process, not from the signal.

If we apply a filter mask to the ordered set $z[k]$ such that the output of the filter is just the middle value in the set (I assume that p is odd). This value is the *median* of the values and the filter is refered to as the *median filter*. How well it does at removing impulsive noise is shown in the lower plot of the figure.

7.2 Mathematical morphology

In general, such order statistic filters can be generalised to a set of filter operations which respond to the 'shape' of the data. Mathematical Morphological filters allow both 1-d and 2-d (images in grey-scale and binary images) filtering using this idea.

- Algebraic system of operators acting on complex shapes
- Decompose into meaningful parts
- Separate from extraneous parts

- 'Optimal' reconstruction from distorted, noisy forms

Example of non-morphological algebraic system – convolution & its frequency domain representation. We may describe any function as a sum of harmonic components. This enables the 'undoing' of noise corruption etc. by filtering i.e. convolving with some kernel function.

What the algebra of convolution does for linear systems, so the algebra of mathematical morphology does for *shape*.

- Shape is a primary carrier of information in machine vision and in many signal processing applications.

- Identification & decomposition of objects
- Binary morphology
- Grey-scale morphology

- Set theory

- Apply to sets in *any* dimension
  (signals & binary vision $d = 2$, grey-scale $d = 3$)

7.3 Primary morphological operations

- Dilation
- Erosion

7.3.1 Dilation

- Combinations of two sets
- Vector addition of set elements

If A, B are sets in N-space (E^N) with element sets $a = \{a_1, ..., a_N\}$ and $b = \{b_1, ..., b_N\}$ then the dilation of A by B is defined as

$$A \oplus B = \{c \in E^N \mid c = a + b \text{ for all } [a \in A, b \in B]\}$$

This operation may be more simply regarded as a translation operation. If B_a represents a translation such that the origin of B now lies on point a of A, then

$$A \oplus B = \bigcup_{a \in A} B_a$$

The following figure illustrates this

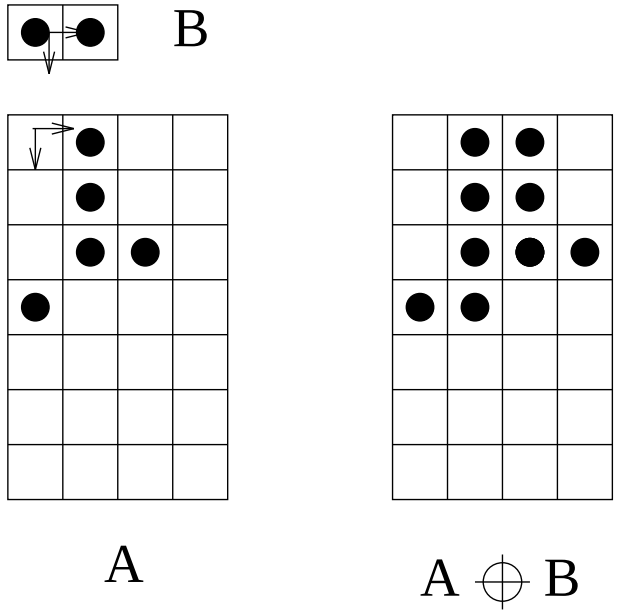


Figure 7.2: *Binary dilation*

- *Commutative*, $A \oplus B = B \oplus A$
- $A \oplus B = \bigcup_{b \in B} A_b$
- A represents image, B the *structuring element*

- $O \in B$ or not? (see figure)
- Dilation is *associative*

$$(A \oplus B) \oplus C = A \oplus (B \oplus C)$$

i.e. a chain rule

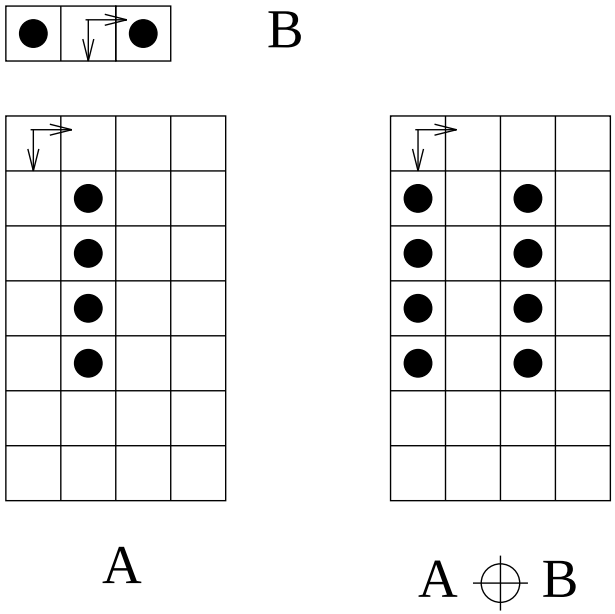


Figure 7.3: *$O \in B$ or not?*

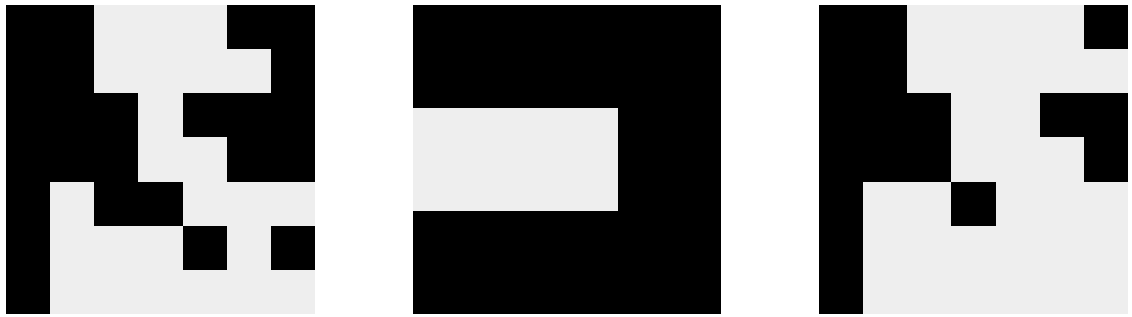


Figure 7.4: *Dilation. Empty (blank) squares are now coloured dark, and binary 1 squares are white. (left) Original data, (middle) Structure element with origin at middle left and (right) resultant dilation.*

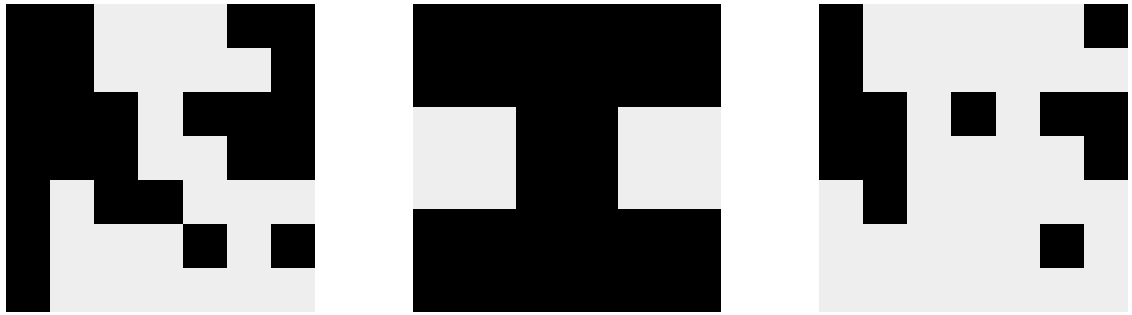


Figure 7.5: *Dilation. (left) data, (middle) structure element with origin at* **centre** *and (right) dilated data.*

What do different structure elements do?

- Dilation by a disk – isotropic expansion
- Dilation by a rod – row or column expansion

Example : 'noise' removal by dilation. Consider

$$J = I \bigcap (I \oplus N_4)$$

If a pixel has *any* 4-neighbours then it will be in $I \oplus N_4$, if not then it won't. So $I \bigcap (I \oplus N_4)$ gives only those points which are 4-connected.

Some other points

- If B_t is a translated structure element, then

$$A \oplus B_t = (A \oplus B)_t$$

- $(B \bigcup C) \oplus A = (B \oplus A) \bigcup (C \oplus A)$
- If $J = A \oplus B$ where origin $\in B$ then $A \subset J$ – this is known as an *extensive operator*
- If $A \subseteq B$ then

$$A \oplus K \subseteq B \oplus K$$

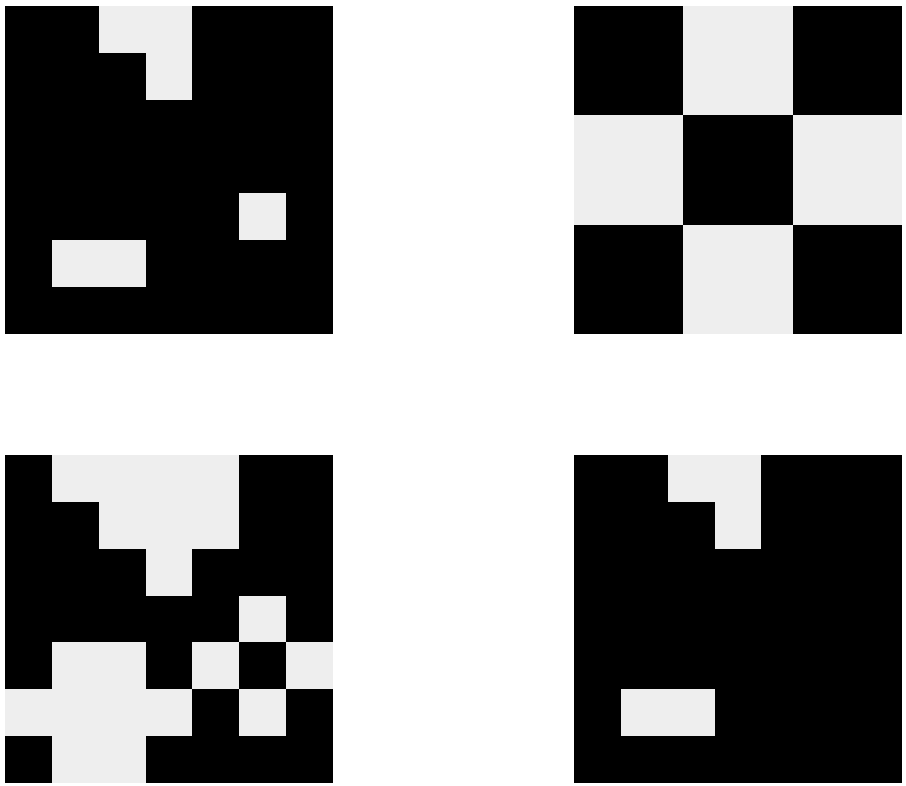


Figure 7.6: *Noise removal by intersection with dilated image. (top left) data, (top right) structure element, origin in centre, (bottom left) dilation and (bottom right) output of intersection.*

7.4 Binary erosion

- Morphological dual to dilation

$$A \ominus B = \{x \in E^N \mid x + b \in A \text{ for all } b \in B\}$$

i.e. this may be regarded as a 'search' for B in A.

Figure 7.7: *Binary erosion. (left) data, (middle) structure element, origin in centre, (right) eroded output.*

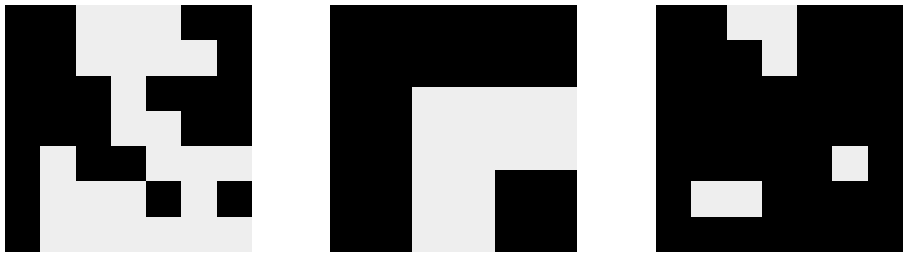


Figure 7.8: *Binary erosion. (left) data, (middle) structure element, origin in centre, (right) eroded output.*

- Dilation regarded as a *union of translates*
- Erosion regarded as an *intersection of negative translates*

$$A \ominus B = \bigcap_{b \in B} A_{-b}$$

- Like dilation, erosion is *increasing* so if $A \subseteq B$ then $A \ominus K \subseteq B \ominus K$
- Eroding with a larger element gives a *smaller* result

Dilation & erosion bear a *foreground/background* relationship. This is formally represented as a *duality* relationship.

For example, de Morgan's laws for binary sets are have dual relationships between *and* and *or* functions.

$$(A \bigcup B)^c = A^c \bigcap B^c$$

For binary images, there are *two* complement operators

- Logical – negation, $A \mapsto A^c$
- Geometric – reflection about the origin, $A \mapsto A^R$

$$(A \ominus B)^c = A^c \oplus B^R$$

and

$$(A \oplus B)^c = A^c \ominus B^R$$

- Erosion is not associative, i.e.

$$(A \ominus B) \ominus C \neq A \ominus (B \ominus C)$$

- Any large erosion can be accomplished using a series of smaller ones
- Duality does *not* imply cancellation
  if $A = B \ominus C$

$$A \oplus C = (B \ominus C) \oplus C \neq B$$

7.5 Uses

There are many uses of erosion / dilation, for example

- Use of erosion to calculate the *genus* of a binary image (the number of connected components minus the number of holes)
- The 'hit-and-miss' transform (HaM). This Selects out pixels which have certain geometric properties (e.g. corner points).
- Thickening and thinning
- Template matching – let the template be set T contained in a window W. Look at all x in I for T and for $W - T$ in I^c

  $$T_x \subseteq I \text{ and } (W - T)_x \subseteq I^c$$

  or, in terms of the HaM transform

  $$I \otimes (T, [W - T])$$

7.6 Opening and Closing

- Perform morphological (shape) filtering. Analogous to bandpass/stop filters in frequency domain
- Can select sub & super shapes of an image
- Opening / closing – like *ideal* BPFs. Once performed, repeated application changes nothing (images are then said to be *open* or *closed* with respect to some structure element)

Opening

The *opening* of image A with stucture element K is defined, in terms of erosion/dilation as :

$$A \circ K = (A \ominus K) \oplus K$$

If we let $B = A \ominus K$ then the above becomes

$$A \circ K = B \oplus K$$

Remembering that the set-element definition of dilation is the *union of translates*, i.e.

$$B \oplus K = \bigcup_{b \in B} K_b$$

hence

$$A \circ K = \bigcup_{y \in A \ominus K} K_y$$

and as the erosion $A \ominus K \subseteq A$ so

$$A \circ K = \bigcup_{K_y \subseteq A} K_y$$

This means that we can picture opening as sweeping K over A *from the inside*, never allowing any part of K to go outside the boundary of A.

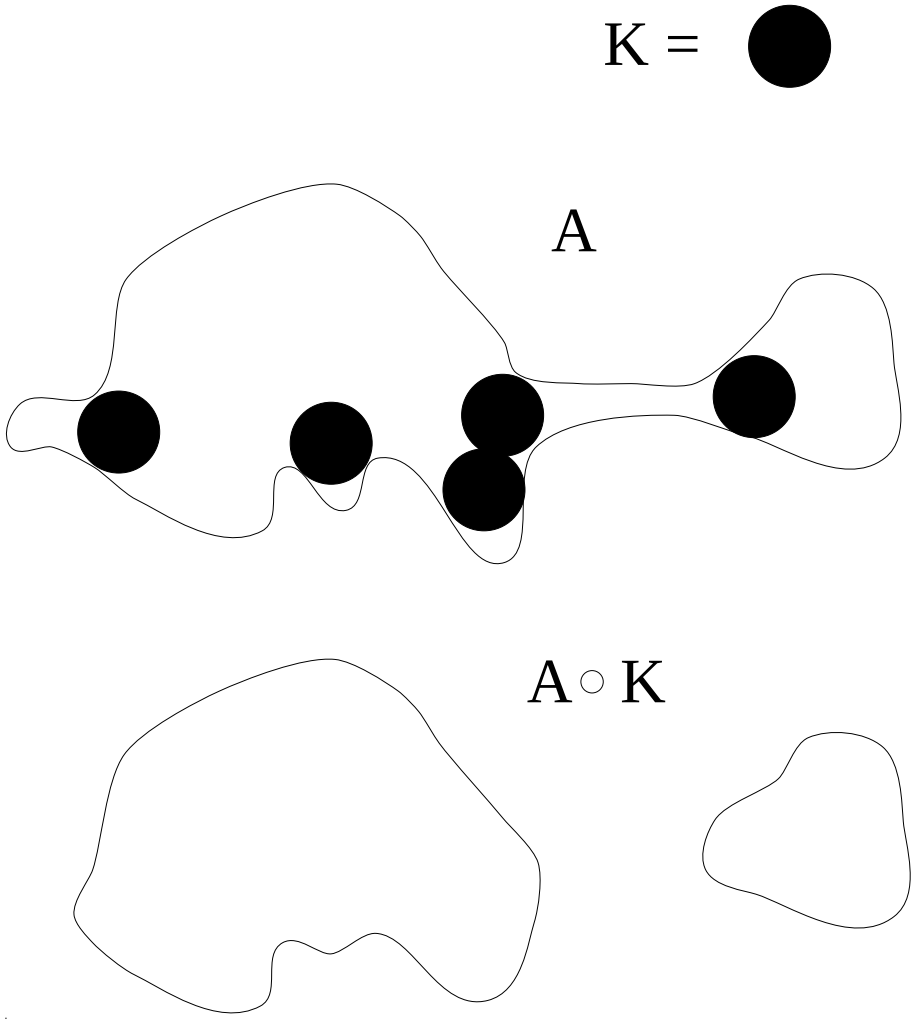


Figure 7.9: *Opening – sweeping of element inside the image*

7.6.1 An example of shape extraction

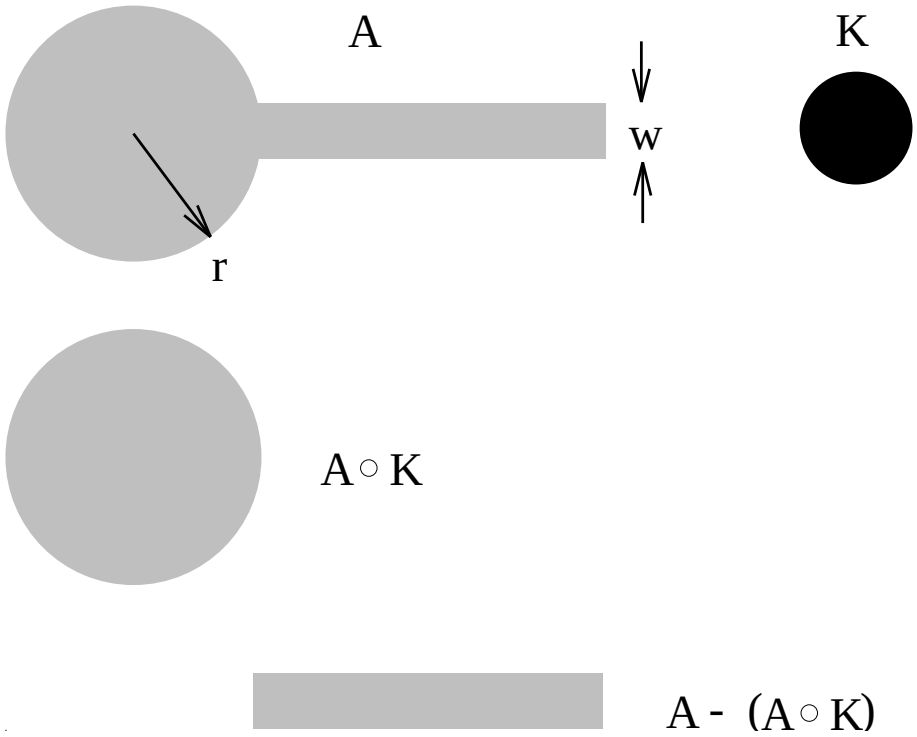


Figure 7.10: *Shape extraction using opening*

In morphological shape extraction, *the order is important*

- Apply the *larger* primitive *first*
- Eliminate the *small* structures *first*

7.7 Closing

Opening sweeps over the *inside* of a binary object. Consider now the case of *complementing* the image $A \mapsto A^c$.

- Open this shape with element K^R (the reflection of K)
- This sweeps over the *inside* of A^c
- Any small gaps between structures in A are small *links* in A^c and are *removed*
- Complement the resultant image

$$(A^c \circ K^R)^c$$

This is defined as the *closing* of A by K. We may think of this as sweeping the *outside* of A using a structure element, K^R.

- a duality exists (this can be proved from the dilation-erosion dualities also) between opening and closing

$$(A \circ K)^c = A^c \bullet K^R$$
$$(A \bullet K)^c = A^c \circ K^R$$

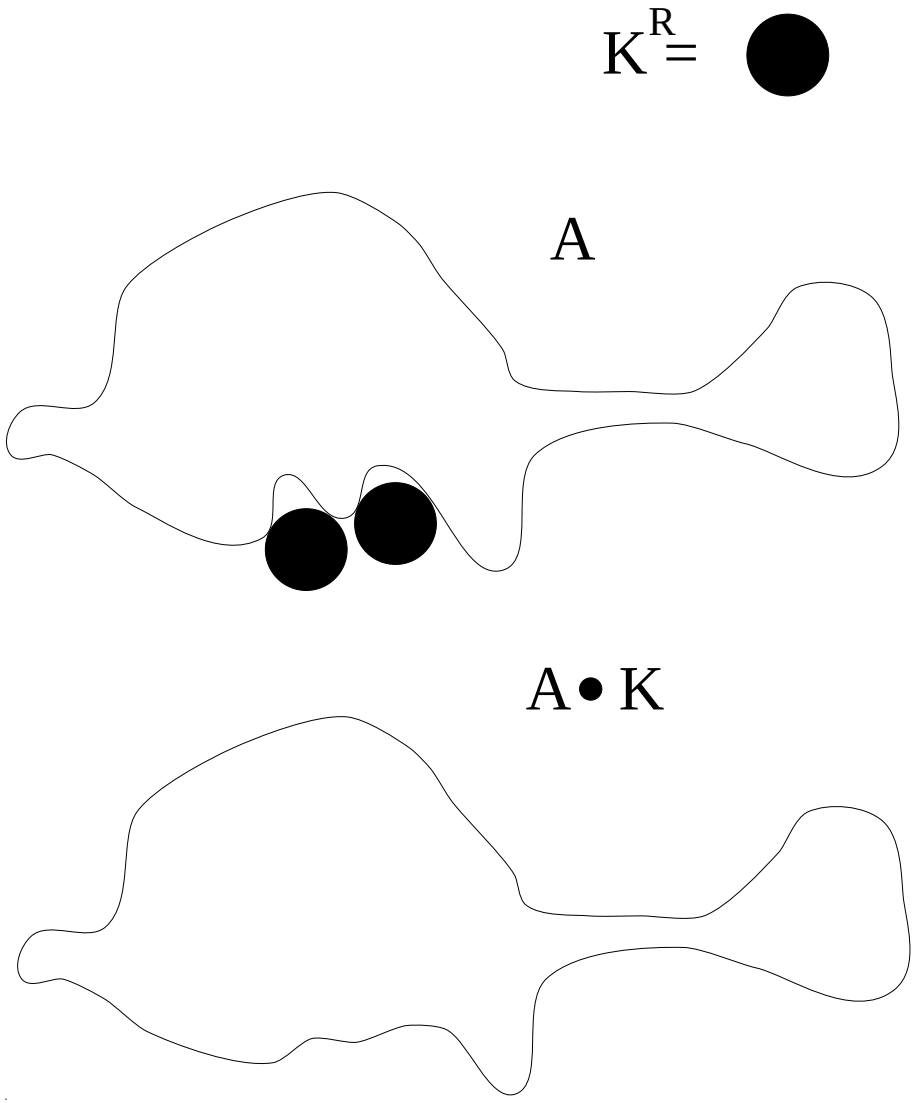


Figure 7.11: *Closing – sweeping of element outside the image*

Some more identities...

$$A \oplus K = (A \oplus K) \circ K$$

$$A \ominus K = (A \ominus K) \bullet K$$

$$(A \circ K) \circ K = A \circ K$$

$$(A \bullet K) \bullet K = A \bullet K$$

There are a variety of uses that morphological processing is put to, for example, to enhance fine details in an image or to remove noise.

7.7.1 The top hat transform

The THT is defined as

$$T = A - (A \circ K)$$

the resultant image, T, hence consists of those pixels in A whose local distribution is *smaller* than that of the structure element K. We may thus think of the THT as a kind of morphological 'high-pass filter'.

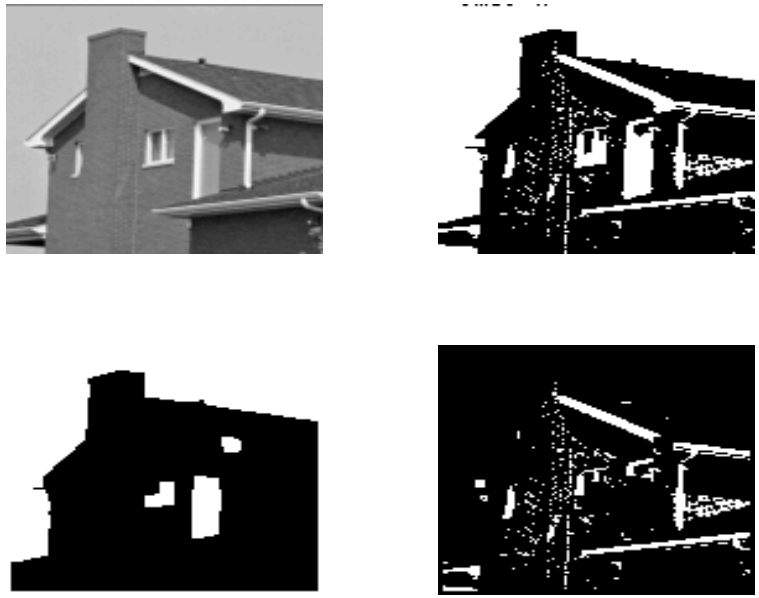


Figure 7.12: *Original, binary image, opened image, THT image. Structure element is 5 pixel square.*

7.7.2 Removing noise

There are many types of noise in grey-scale images, if we concentrate on binary images then noise is 'salt-and-pepper' (a.k.a. drop-out), where pixels are randomly complemented. One strategy for removing such noise is to open then close with a disk element, D, where the radius of D is *small* compared to the local boundary curvature of components in the image, but is *larger* than the local noise (which is pixel-by-pixel). A good choice may be $r = 3$ pixels, say.

- Perform $(I \circ D) \bullet D$
- Opening removes impulsive noise in the background
- Closing removes dropout noise in objects
- Trade-off between noise removal and boundary changing
- At high noise levels, noise may by chance 'floc' into clumps which are not removed by a *small* disk. But a larger disk smooths the object boundaries...

7.8 Grey-scale Morphology

We have concentrated so far on *binary images* – often we want to directly process the *grey-scale* original. G-S morphology lets us operate on shapes defined by their grey-scale levels and spatial attributes (i.e. the image 'space' is now 3-D not 2-D). We can then consider a grey-scale image with only one row - which is just a signal - and then apply the same theory to obtain operators for signal processing.

7.8.1 The *top surface* & the *umbra*

Consider a grey-scale scan line (a row of an image, say) as shown in the figure. The *umbra* (shadow) is defined by the G-S space lying *below* the scan line, f_c, and is denoted as $U[f_c]$, each point on the G-S line f_c generates a *set* of umbra points, $\{u_{i,c}\}$

The *top* of an umbra set is the set of maximum values of $\{u_{i,c}\}$ at for each column co-ordinate, i.e.

$$T_c = \max\{u_{i,c}\}$$

so top and umbra are inverse operators,

$$T\{U[f]\} = f$$

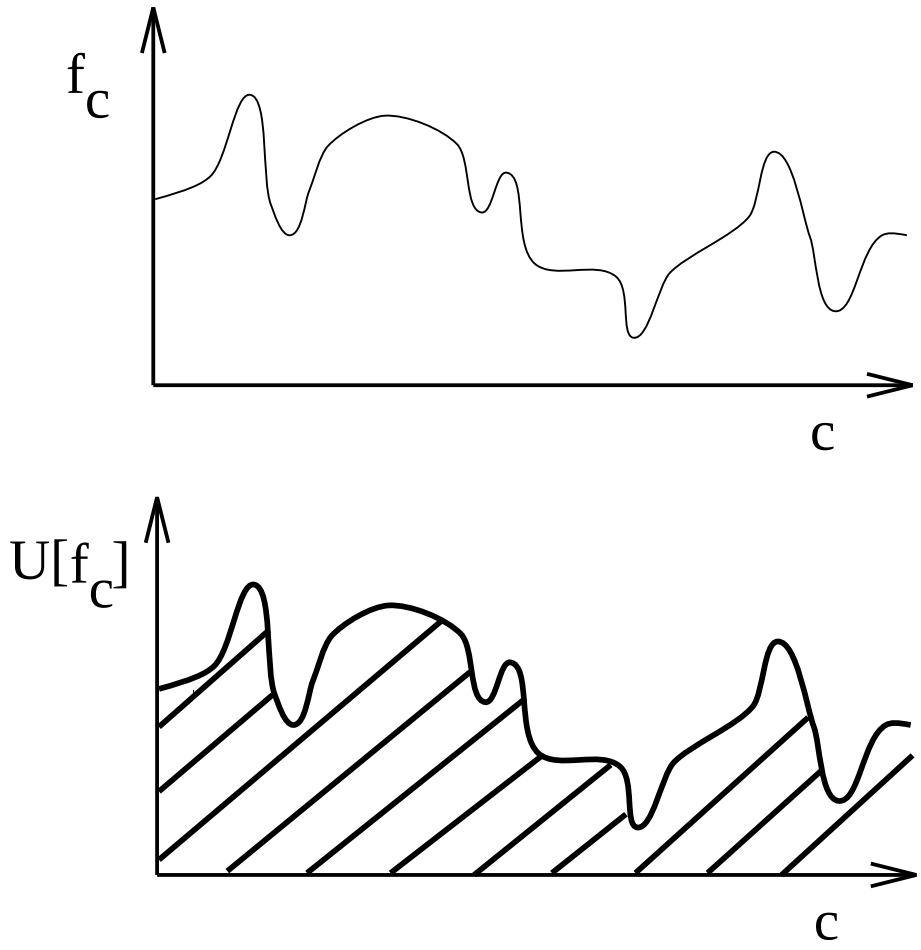


Figure 7.13: *The top and umbra operations.*

7.9 Grey-scale dilation & erosion

G-S dilation is defined as

$$f \oplus k = T\{U[f] \oplus U[k]\}$$

i.e. the 'binary' dilation of the two umbra functions.

In the same way we may define the G-S erosion as

$$f \ominus k = T\{U[f] \ominus U[k]\}$$

We note that, like the binary versions, G-S erosion and dilation are *order independant*.

$$f \ominus k = k \ominus f$$

$$f \oplus k = k \oplus f$$

7.10 Grey-scale opening and closing

By the definitions of opening and closing in terms of erosions and dilations so :

$$f \circ k = (f \ominus k) \oplus k$$

$$f \bullet k = (f \oplus k) \ominus k$$

And a duality relationship holds; remember that for binary operators

$$(A \circ K)^c = A^c \bullet K^R$$

for G-S

$$-(f \circ k) = (-f) \bullet k^R$$

Note that the G-S equivalent of complementing is inversion.

But, how do we interpret the G-S opening and closing more easily?

Using the definitions for G-S erosion & dilation :

$$\begin{aligned} f \circ k &= (f \ominus k) \oplus k \\ &= (T\{U[f] \ominus U[k]\}) \oplus k \\ &= T\{U[T\{U[f] \ominus U[k]\}] \oplus U[k]\} \end{aligned}$$

and as $T\{U[f]\} = f$ so

$$f \circ k = T\{(U[f] \ominus U[k]) \oplus U[k]\}$$

The operation on the umbrae in the above is equivalent to a binary opening between $U[f]$ and $U[k]$, hence

$$f \circ k = T\{U[f] \circ U[k]\}$$

Remember that binary opening was thought of as the 'sweeping' of an element round the *inside* boundary of an image structure.

We may thus interpret the G-S opening as the top surface of what remains of $U[f]$ *after sweeping* $U[k]$ *along the* **underside** *of* f*.*

By application of the duality between opening and closing, *we may interpret G-S closing as sweeping the* **reflection of** $U[k]$, $U[k]^R$, *over the* **topside** *of* f*.*

- Opening smooths from the underside
- Closing smooths from the topside
- Concepts of *open*, *closed* to an element hold
- $(f \circ k) \circ k = f \circ k$
- $(f \bullet k) \bullet k = f \bullet k$

- Can perform, for example, a top-hat transform or noise cleaning

Going from a set of scan lines to a 3-D image (G-S) is easy :

- *Opening* sweeps the umbra of the structure element *under* the G-S surface, smoothing from *below*
- *Closing* sweeps the inverted umbra of the element *over* the G-S surface, smoothing from *above*
- Choice of structure element... Ball, paraboloid are commonly used

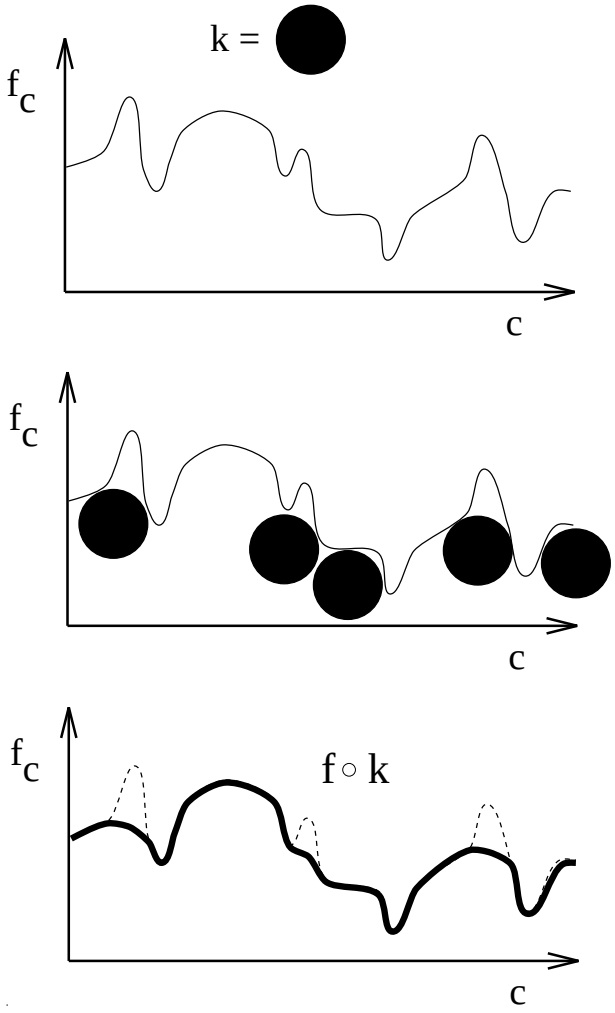


Figure 7.14: *Grey-scale opening*

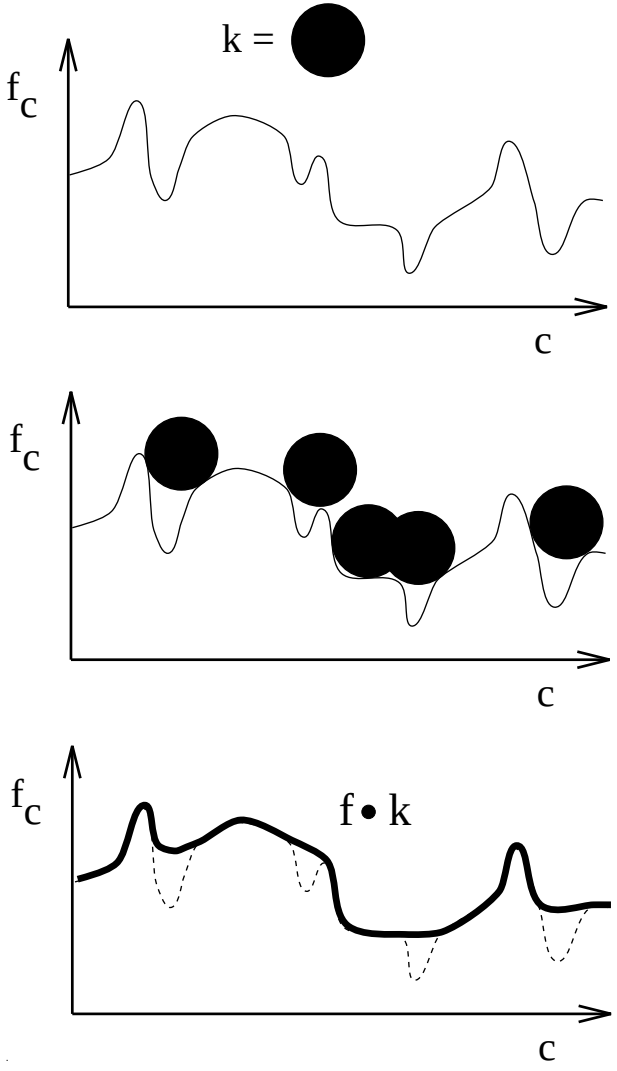
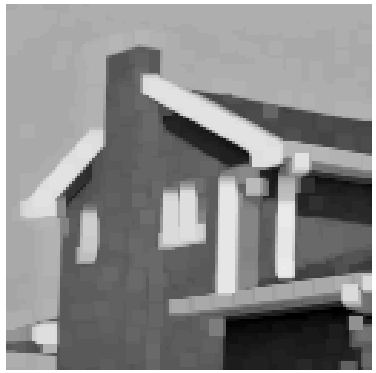


Figure 7.15: *Grey-scale closing*



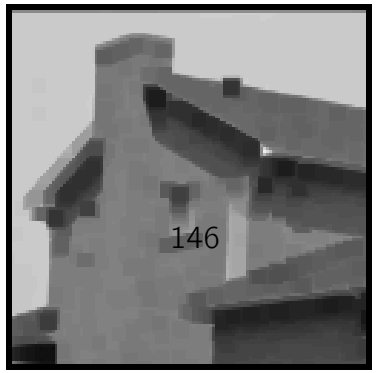


Figure 7.16: *Grey-scale morphological operations. Original, dilation and erosion*

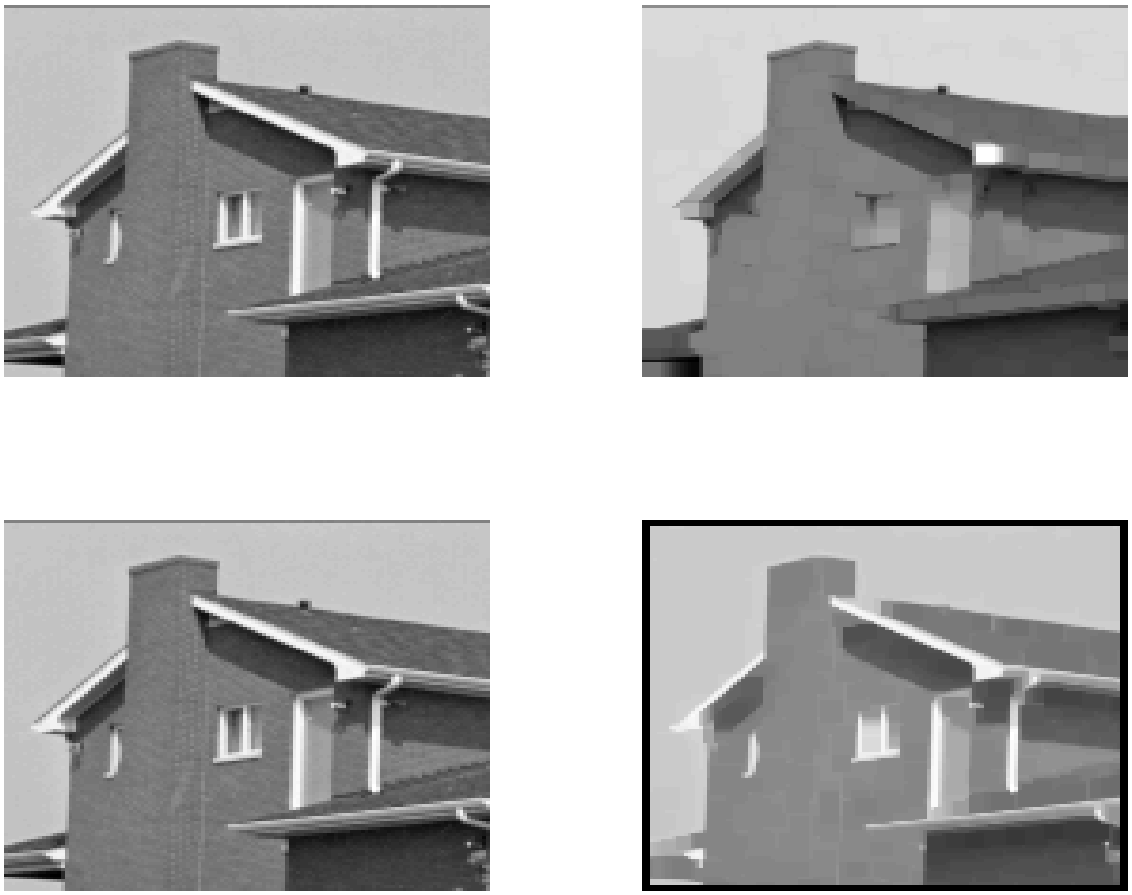


Figure 7.17: *Grey-scale morphological operations. Opening and closing.*

7.10.1 Signals

Applying the previous theory to signals is straightforward given that we can consider an N point signal as just a $1 \times N$ grey-scale image. We may apply all the same operations - the examples below shows hoe the shape of the signal is changed and how easily noise components may be removed.

(a)

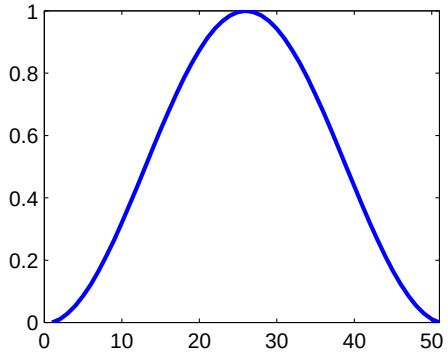


Figure 7.18: *Simple signal structure element*

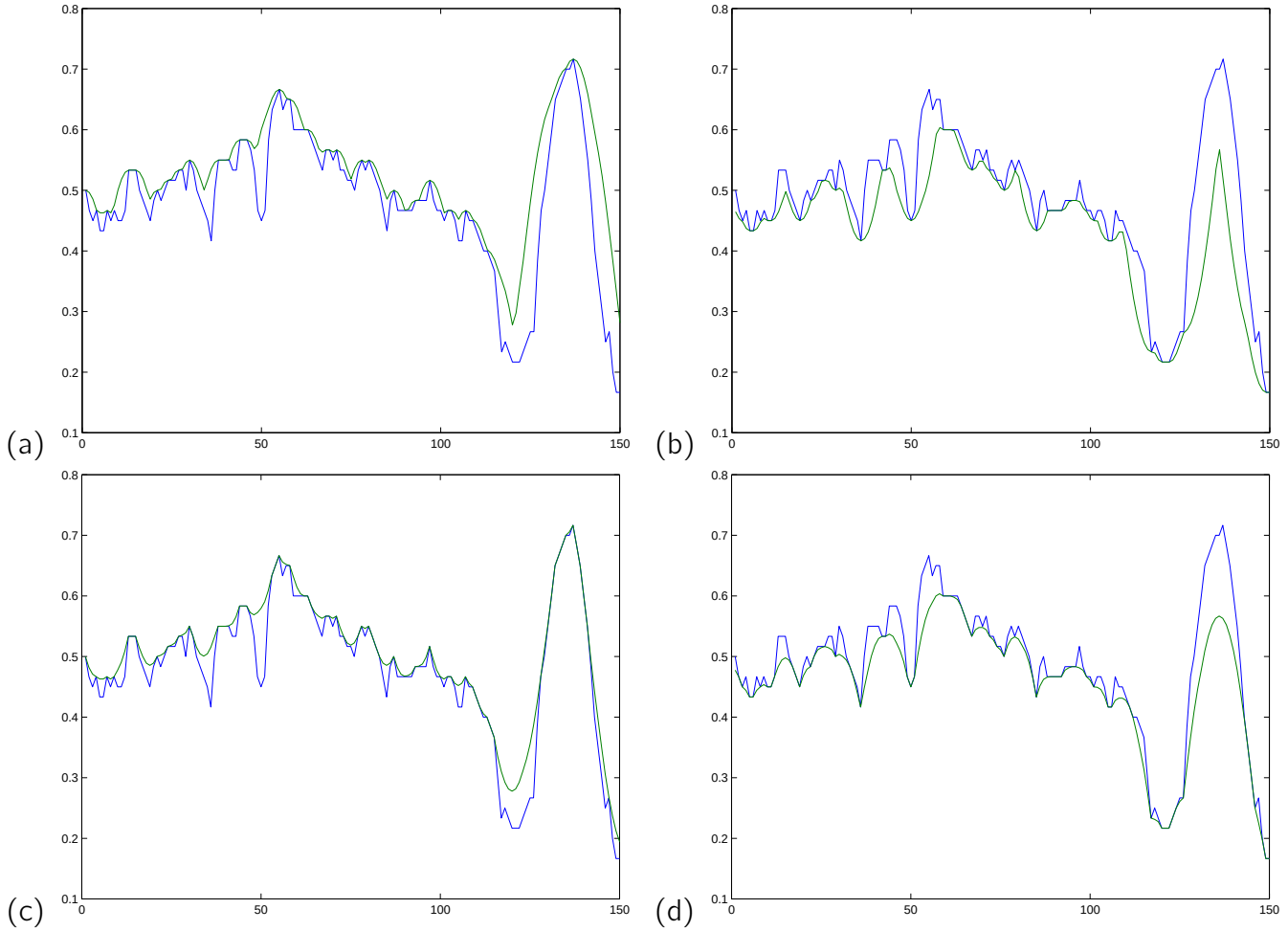


Figure 7.19: *(a) Dilation, (b) Erosion, (c) Closing and (d) Opening of signal.*

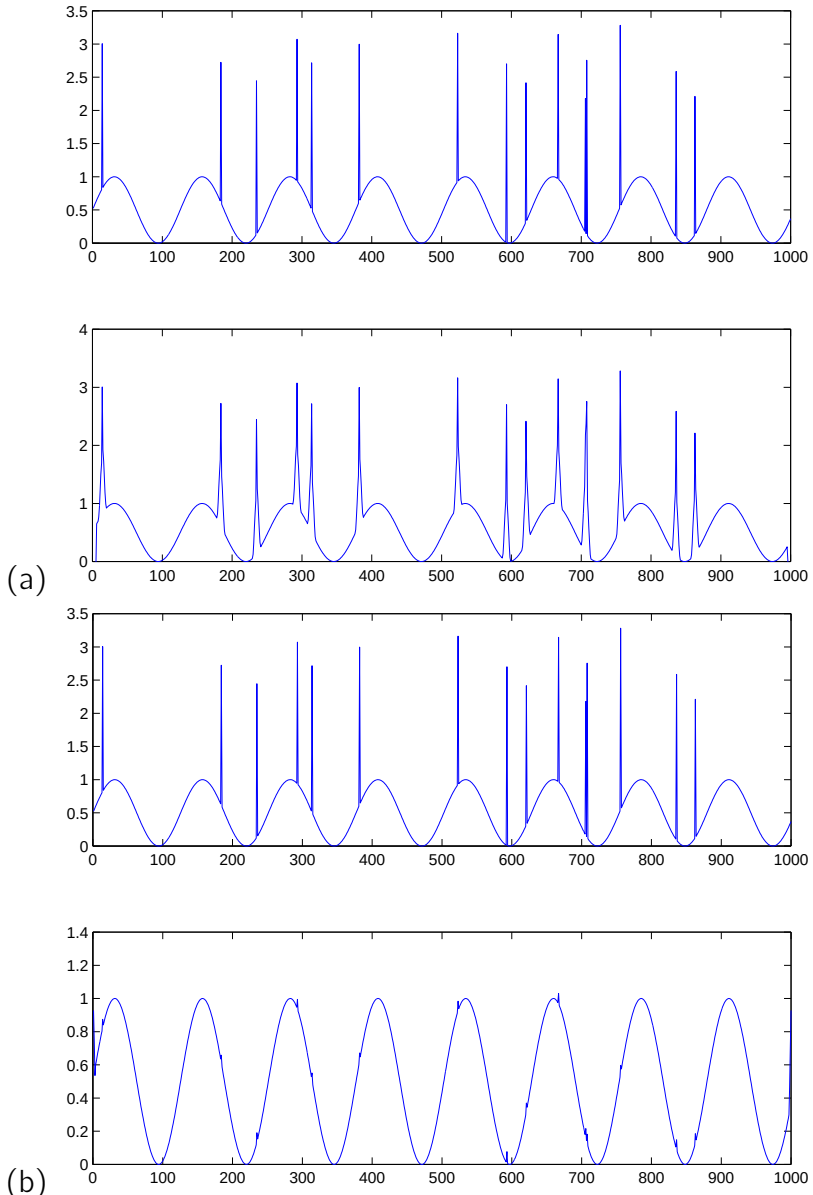


Figure 7.20: *(a) Closing and (b) Opening of noisy sine signal.*

Lecture 8 – Compression & coding

Why do we want to compress signals and images?

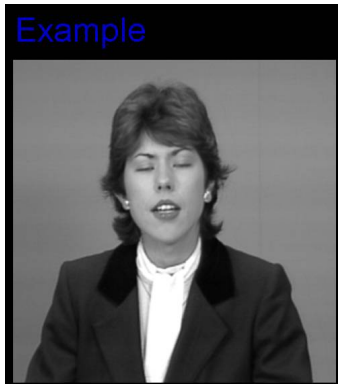
- Storage
- Bandwidth of transmission

When we consider compression, we rely on the fact that there is *redundency* in the data.

Example PAL colour video 768 x 576 pixels at 25 frames per second = 33 Mbytes per second. Hence a 2hour video = 238 Gbytes (cf CD holds 0.7 Gbytes, a DVD holds 7 Gbytes). So we need compression!

Lossless or lossy? Lossless compression has no errors in reconstruction. Examples include gzip, zip and winzip. Typical compression is of order 2:1 to 3:1.

Lossy compression can achieve much better compression ratios, at the loss of some detail. Achieved rates can be 10:1 to 50:1 and still have no preceivable difference! For example...



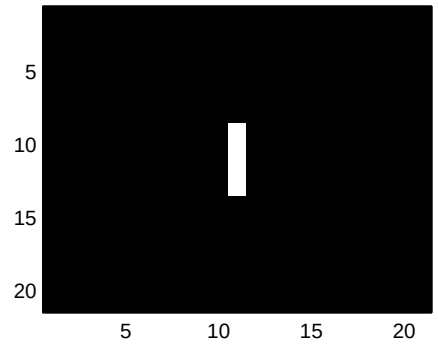
ORIGINAL

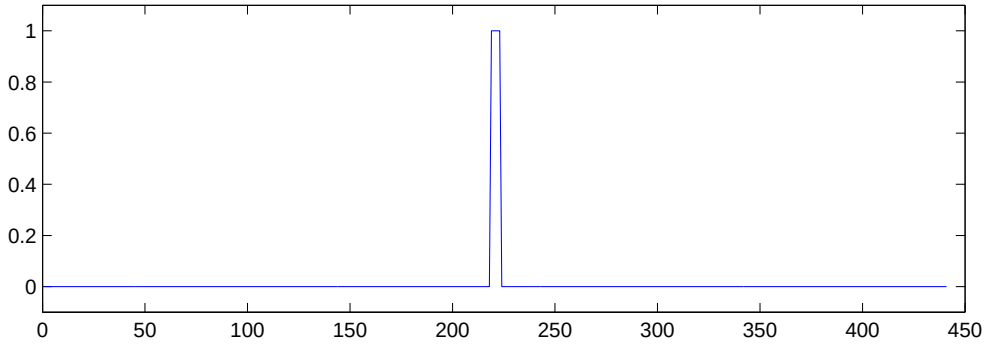
JPEG COMPRESSION 16:1

8.1 Lossless compression

Let's look at an image of a really simple object - a blob in an image...

There is a lot of redundency here. We can use a coding which tells us that at particular pixels, as we scan from top left to bottom right, change their value.



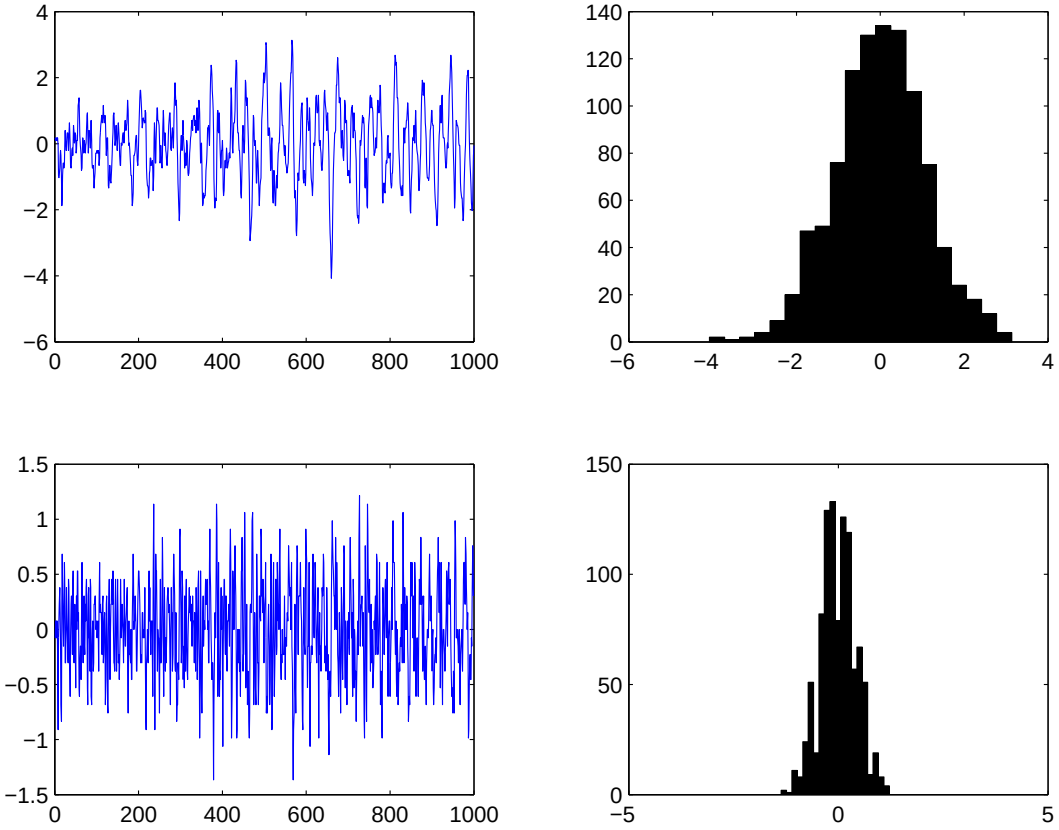


Hence

$$[0], [218, +1], [223, -1], [0]$$

tells us all we need to know about this image.

Consider the signal:



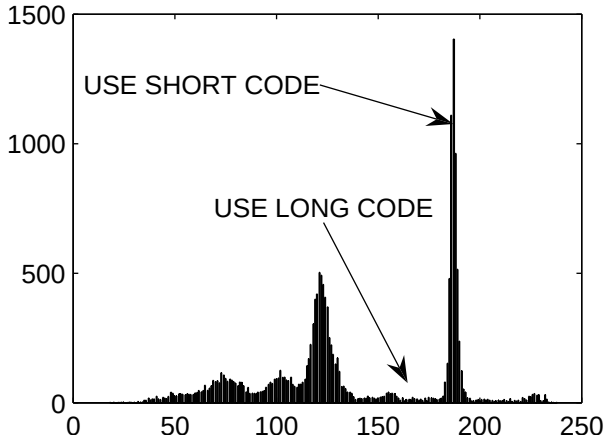
It is clear that if we only describe *changes* to the data then we need less storage

and so achieve compression!

Although very clever lossless compression schemes exist, much better compression ratios can be achieved if we are prepared to lose some information.

8.2 Lossy compression

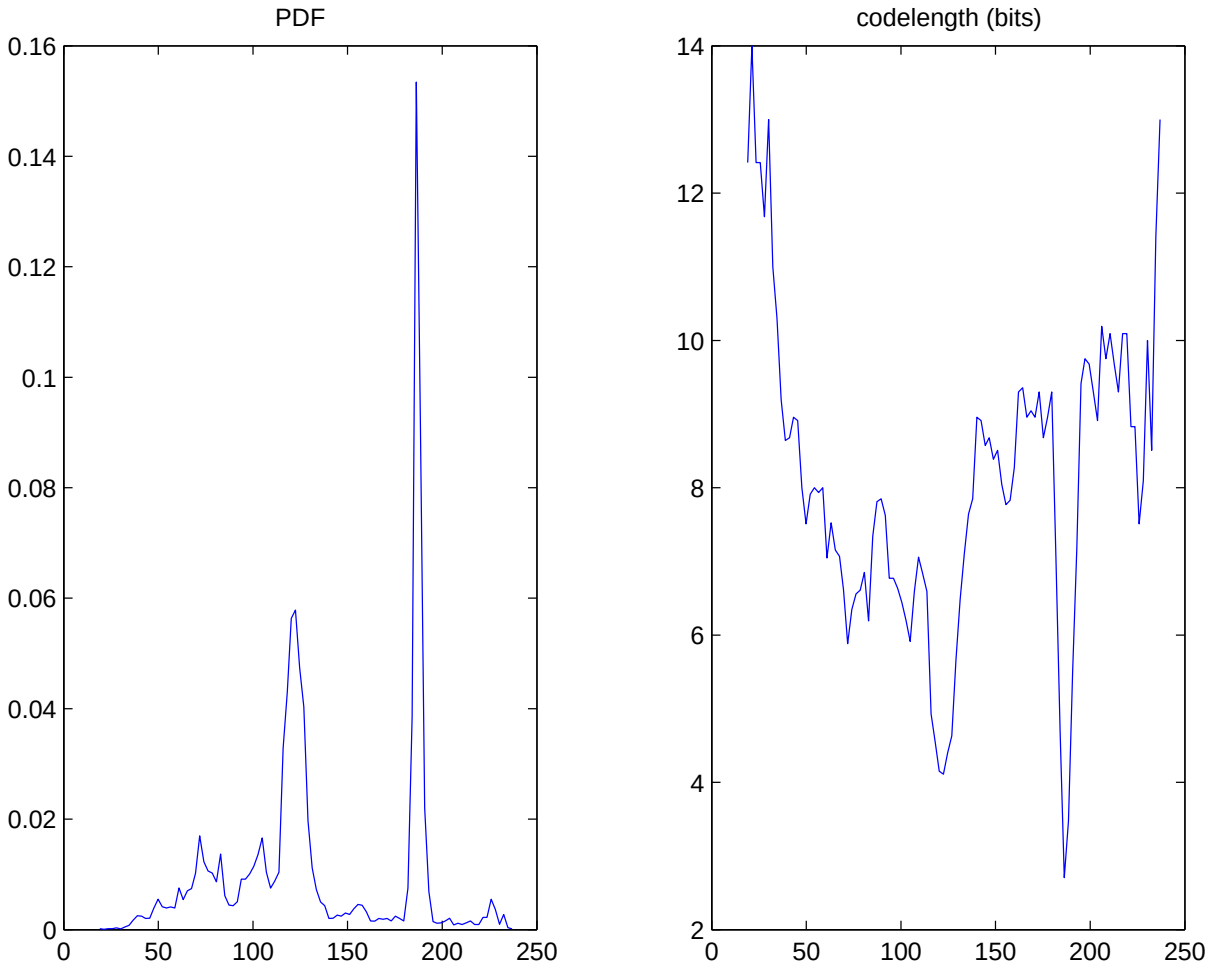
Let's look at the intensity histogram.



We can save by using short codes for the intensity values that are very frequently used, such as around 200, and long codes for the ones that are hardly used, such

as around 150 or close to 0. Optimally it turns out that the code length

$$\text{codelength} \propto -\log \mathrm{p}(\text{intensity})$$



If we took this approach we would end up with lossless compression, but we could go even further and quantise little used values more coarsely. Doing this kind of compression on the *original signal or image* is not the easiest way though. We can represent the data in a *basis* format first.

8.2.1 Representations, DFT, DCT

$$f(x) = \sum_i w_i \phi_i(x)$$

is a general linear basis. We can see that the Discrete Fourier Transform is of this form.

$$F(u,v) = \frac{1}{M}\frac{1}{N}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\exp[-i2\pi(ux/M + vy/N)]$$

It turns out that it is simpler to use a cosine expansion, as it saves on coding. This is the Discrete Cosine Transform (DCT) and it is defined as

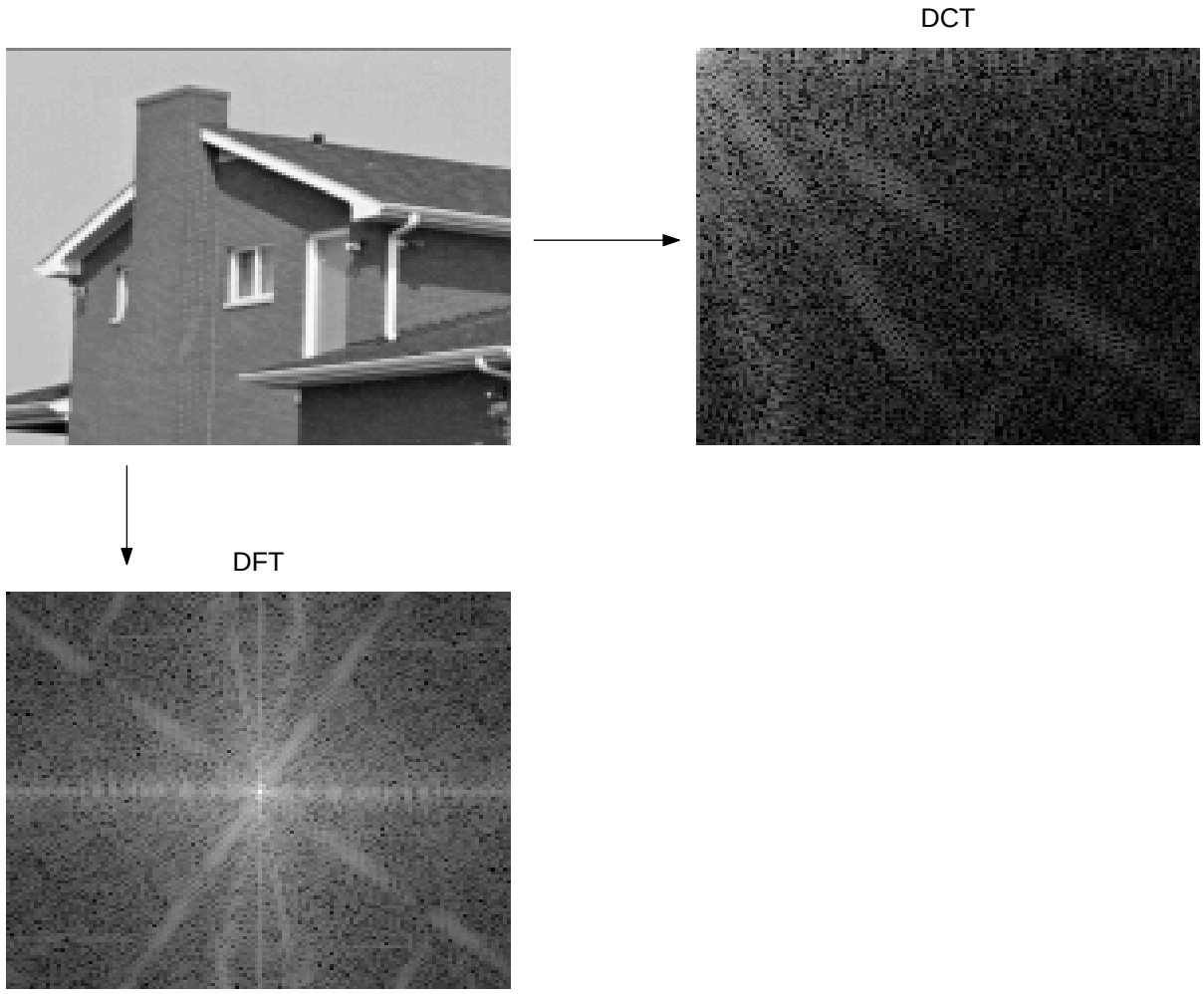
$$F(u,v) = \frac{4c(u)c(v)}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\cos\left(\frac{\pi}{M}u(x+1/2)\right)\cos\left(\frac{\pi}{N}v(y+1/2)\right)$$

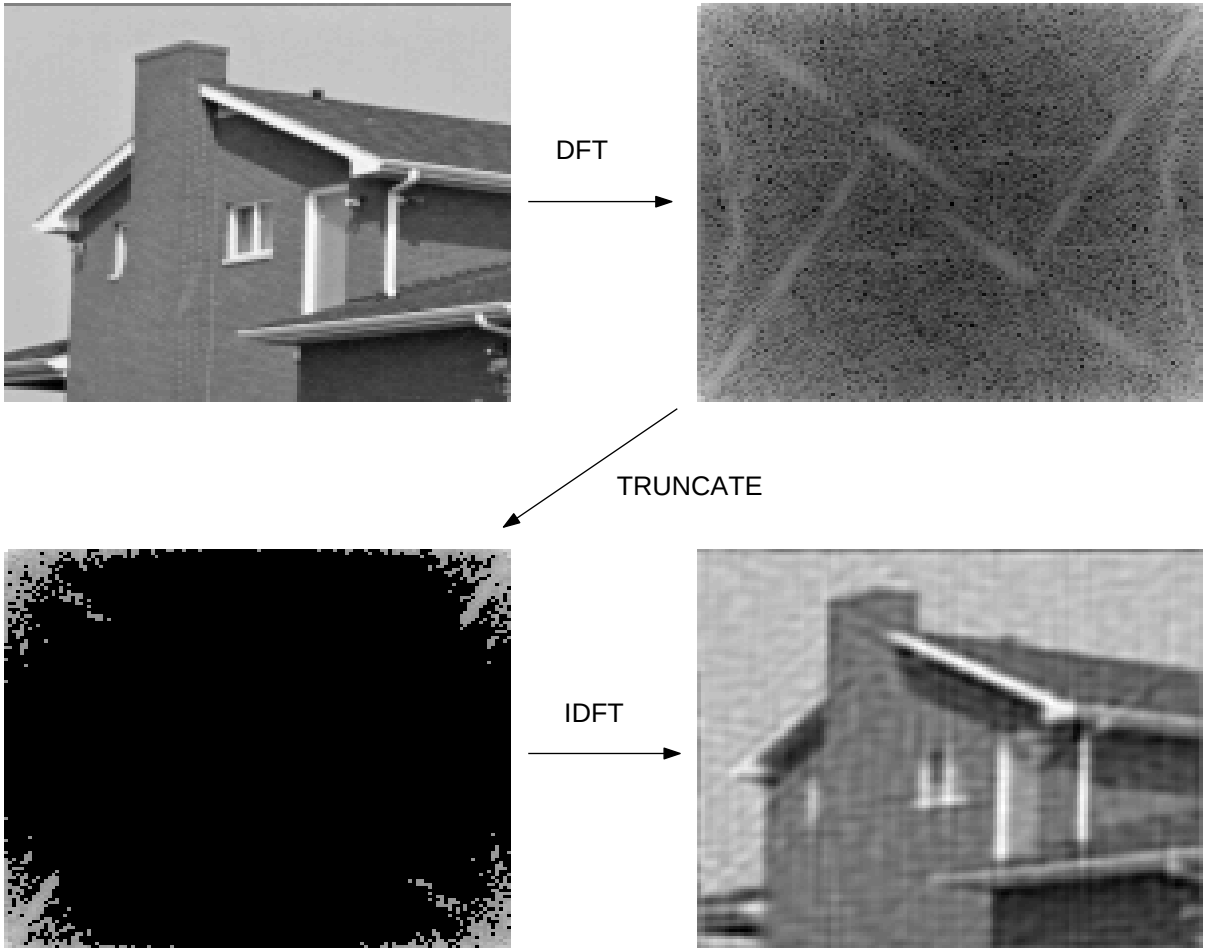
where

$$\begin{aligned} c(a) &= \frac{1}{\sqrt{(2)}} \text{ for } a = 0 \\ &= 1 \text{ for } a = 1, 2, ... \end{aligned}$$

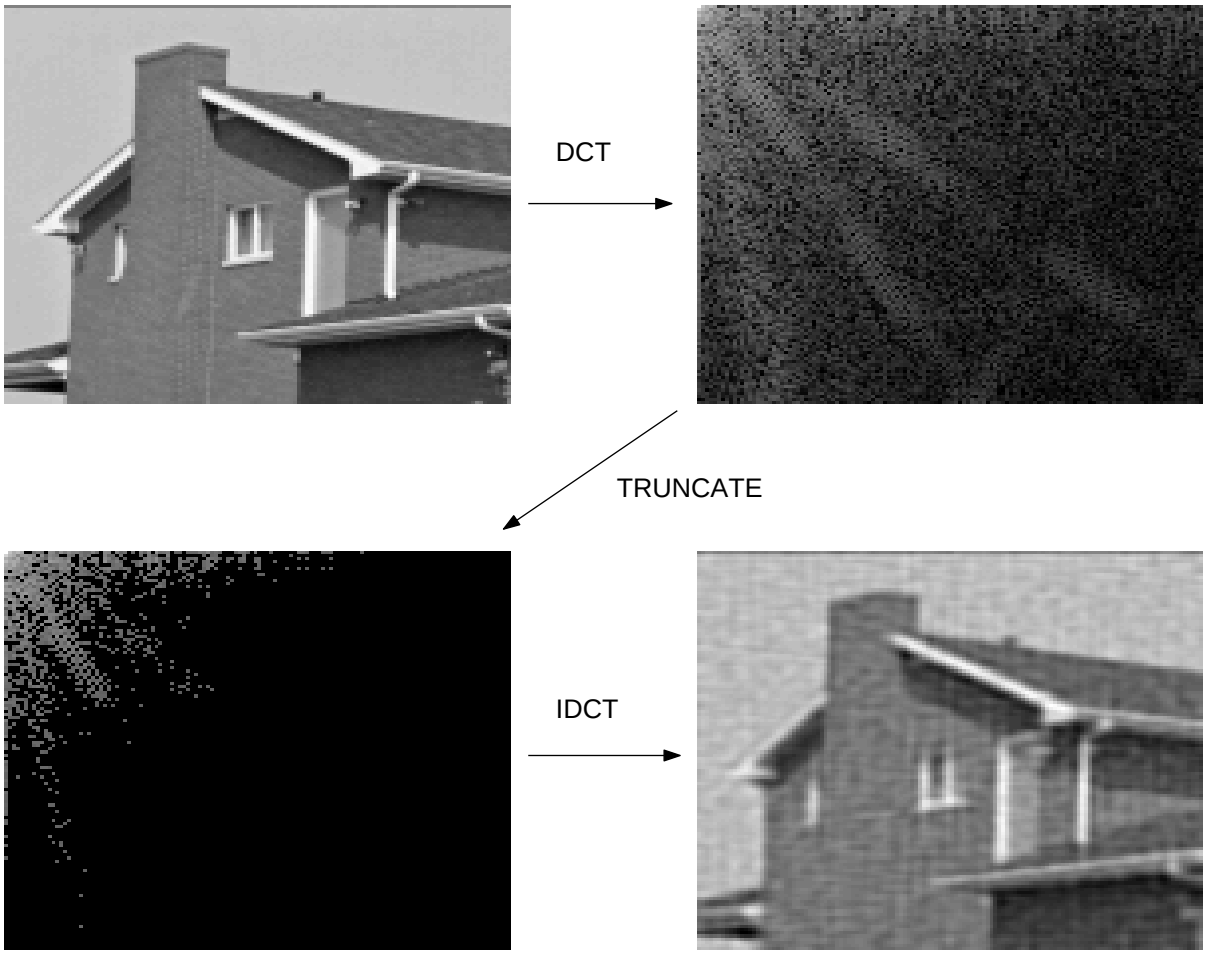
The inverse DCT is defined as

$$f(x, y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} c(u)c(v)F(u, v) \cos\left(\frac{\pi}{M} u(x + 1/2)\right) \cos\left(\frac{\pi}{N} v(y + 1/2)\right)$$


DCT
DFT



14:1 COMPRESSION

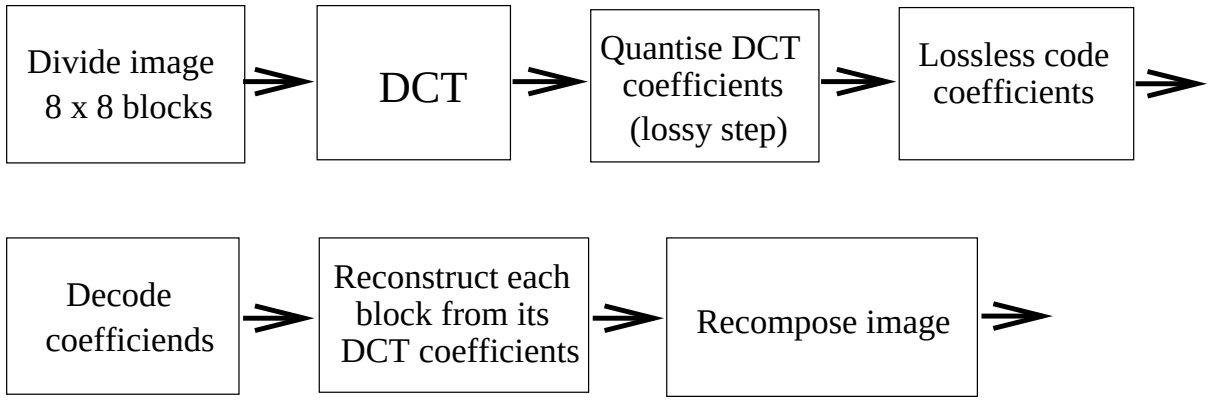

DCT
TRUNCATE
IDCT
20:1 COMPRESSION

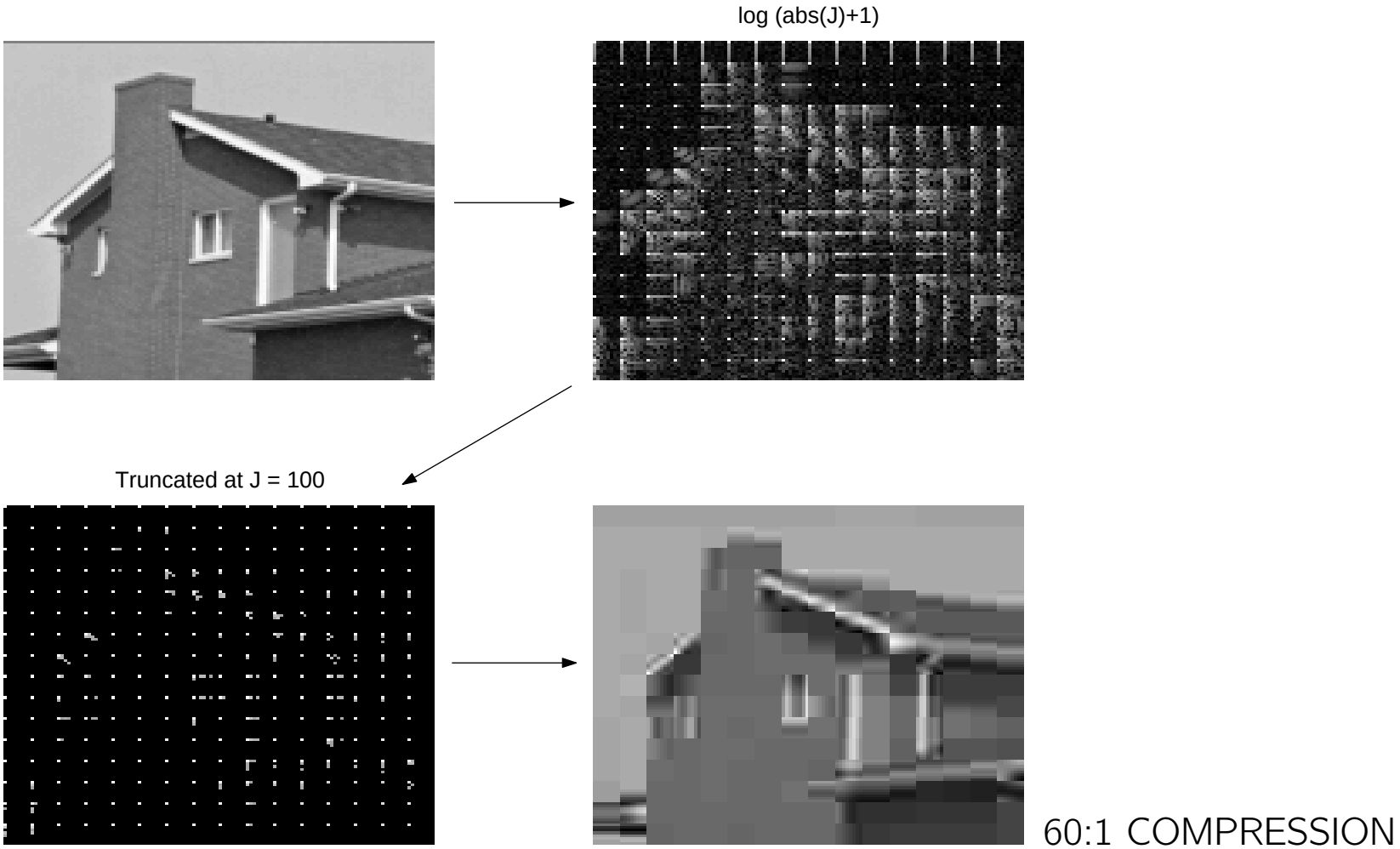
8.2.2 JPEG

The Joint Photographics Experts Group (JPEG) compression scheme uses lossy and lossless compression. The JPEG method

- World standard for compression of general images
- Allows progressive display of images (ie from low quality to high)

Rather than apply a transform to the entire image in one go, it uses the fact that information is local and so divides the image into 8×8 blocks on which the DCT is performed.

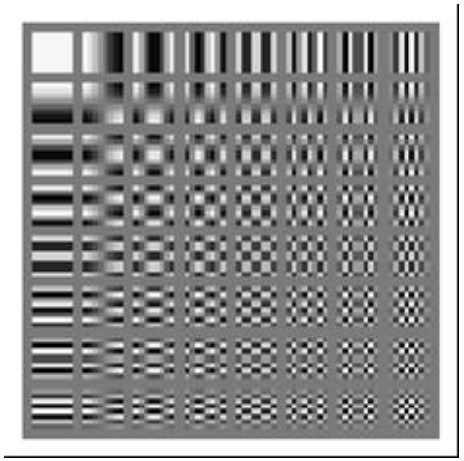




60:1 COMPRESSION

The quantisation consists of two parts:

Weighting matrix The human eye does not have the same sensitivity to all frequencies. Therefore, a coarse quantisation of the DCT coefficients corresponding to high frequencies is less annoying for the human being than the same quantisation applied to low frequencies. Hence, to obtain a minimal perceptual distortion; each coefficients should be individually weighted: This is achieved by the use of a weighting matrix which is sent as side information:

BASIS 

$$Z(u,v) = \begin{matrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{matrix}$$

$$\text{out}(u,v) = \text{round}[F(u,v)/Z(u,v)]$$

- $Z(u, v)$ is chosen to weight coefficients which are visually salient
- we can scale $Z(u, v)$ to alter compression quality

Uniform quantisation factor In addition to this weighting matrix. a uniform quantisation factor can be applied to the entire image. This will quantise into a set of M levels. The more M the greater the number of coefficients that are *not* set to zero and so the lower the compression. The resultant quantised coefficients are then de-normalised by multiplication by $Z(u, v)$.

Differential coding of DC coefficients The DC DCT coefficient of each block can become quite large. Therefore only the difference between the DCT coefficient of the current and the previous block is coded and transmitted.

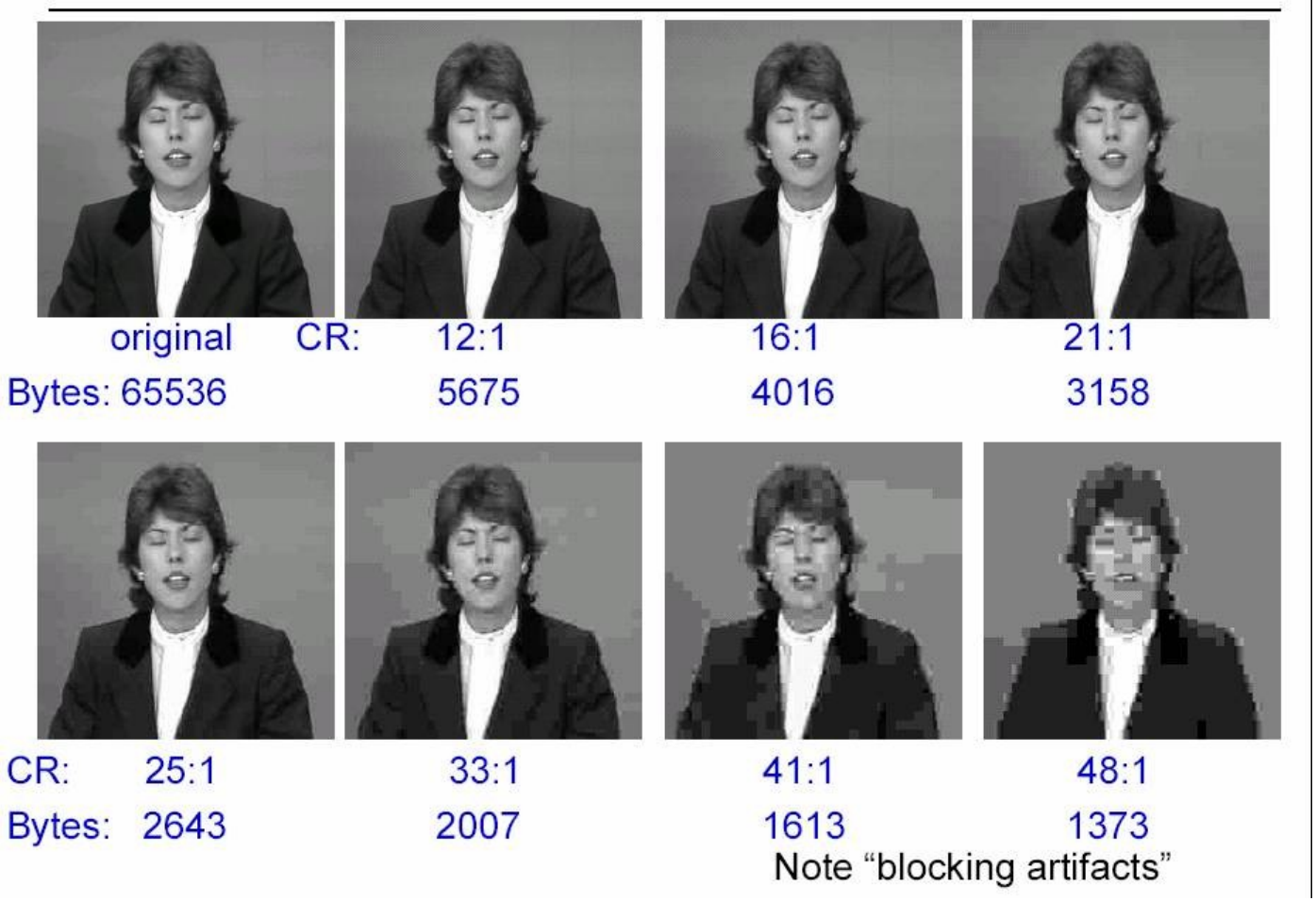


Figure 8.1: JPEG performance.

8.3 Wavelets

The WT is an operation that forms a signal representation that, unlike the Fourier transform for example, is local in both time and frequency domains. The WT relies upon smoothing the time-domain signal at different scales; thus if $\psi_s(x)$ represents a wavelet at scale s, then the WT of a function $f(x) \in L^2(\Re)$ is defined as a convolution given by :

$$Wf(s,x) = f * \psi_s(x) \tag{8.63}$$

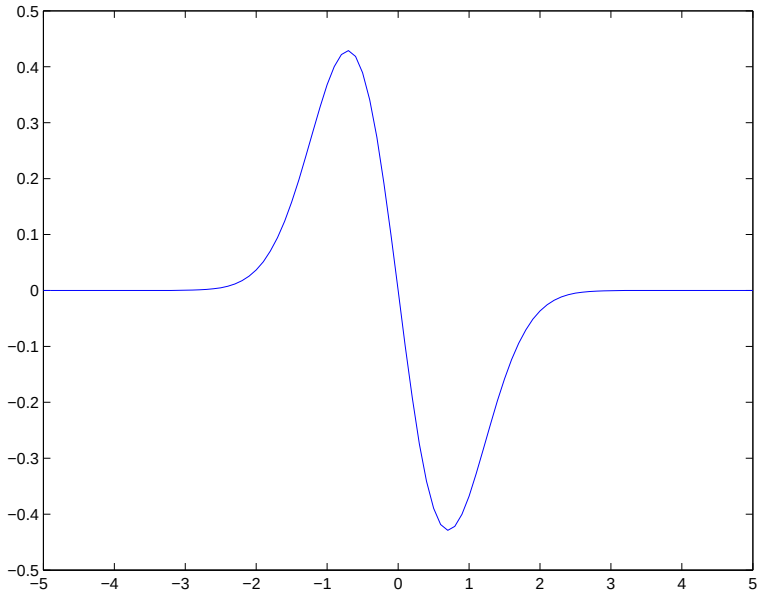
Moreover, the signal $f(x)$ may be reconstructed to a good approximation from a knowledge of the modulus maxima of the WT alone, and hence these maxima offer a compact representation ideal for noise removal. The scaled wavelets are constructed from a 'mother' wavelet, $\psi(x)$, such that :

$$\psi_s(x) = (1/s)\psi(x/s) \tag{8.64}$$

with the function $\psi(x)$ chosen to be a derivative of a smoothing function.

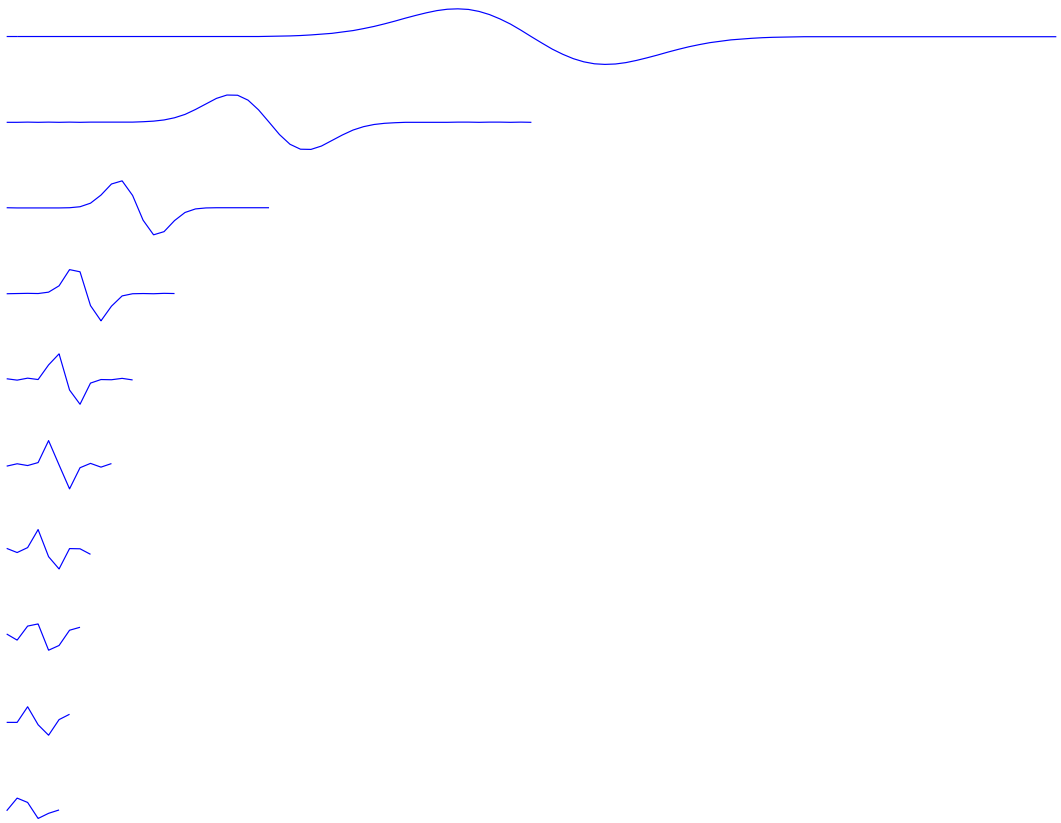
Just as a simple example we choose this smoothing function to be a Gaussian, $G(x)$, and define the 'mother' wavelet as :

$$\psi(x) = \frac{dG(x)}{dx} \tag{8.65}$$



We can allow the scale to vary - it turns out that we can just retain decimatation by a constant factor, normally 2.

$$s_n = 2s_{n-1} \text{ hence } s_n = 2^n s_o$$



We note that as these wavelets are to be represented as a digital sequence, the smallest wavelet requires at least 4 sample points for definition. This implies that the wavelet decomposition has a high-frequency cut-off at $f_{samp}/4$.

Wavelets can be thought of (explicitely) as a filter bank whose frequencies are in a *scale-space*.

8.3.1 JPEG-2000

This new version of JPEG uses a similar approach to the original JPEG but instead of DCT coding, wavelets are used instead. This gives a big improvement in compression quality at the same compression levels.

COMPRESSION

158:1