6 Signal Processing

6.1 Introduction

Signal processing refers to techniques for manipulating a signal to minimize the effects of noise, to correct all kinds of unwanted distortions or to separate various components of interest. Most signal processing algorithms include the design and realization of filters. A *filter* can be described as a system that transforms signals. *System theory* provides the mathematical background for filter design and realization. A filter as a system has an input and an output, where the *output signal* $y(t)$ is modified with respect to the *input signal* $x(t)$ (Fig. 6.1). The *signal transformation* is often called convolution or, if filters are applied, filtering.

This chapter is on the design and realization of *digital filters* with the help of a computer. However, many natural processes resemble *analog filters* that act over a range of spatial dimensions. A single rainfall event is not recorded in lake sediments because short and low-amplitude events are smeared over a longer time span. Bioturbation also introduces serious distortions for instance to deep-sea sediment records. Aside from such natural filters, the field collection and sampling of geological data alters and smoothes the data with respect to its original form. For example, a finite size sediment sample integrates over a certain period of time and therefore smoothes the natural signal. Similarly, the measurement of mag-

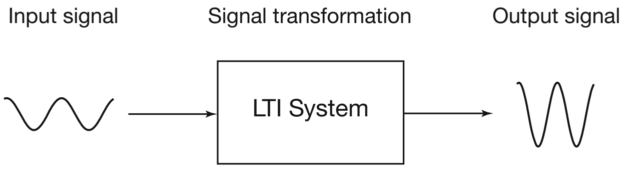

Fig. 6.1 Schematic of a linear time-invariant (LTI) system. The input signal is transformed into an output signal.

netic susceptibility with the help of a loop sensor introduces significant smoothing since the loop integrates over a certain section of the sediment core.

The characteristics of these natural filters are often difficult to determine. Numerical filters, however, are designed with well-defined characteristics. In addition, artificial filters are time invariant in most cases, while natural filters, such as lake mixing or bioturbation, may change with time. An easy way to describe or predict the effect of a filter is to explore the filter output of a simple input signal, such as a sine wave, a square wave, a sawtooth, ramp or step function. Although there is an endless variety of such signals, most systems or filters are described by their impulse response, i.e., the output of a unit impulse.

The chapter starts with a more technical section on generating periodic signals, trends and noise, similar to Chapter 5.2. Chapter 6.3 is on linear time-invariant systems, which provide the mathematical background for filters. The following Chapters 6.4 to 6.9 are on the design, the realization and the application of linear time-invariant filters. Chapter 6.10 then suggests the application of adaptive filters originally developed in telecommunication. Adaptive filters automatically extract noisefree signals from duplicate measurements on the same object. Such filters can be used in a large number of applications, such as noise removal from duplicate paleoceanographic time series or to improve the signal-to-noise ratio of parallel color-intensity transects across varved lake sediments (see Chapter 5, Fig. 5.1). Moreover, such filters are also widley-used in geophysics for noise canceling.

6.2 Generating Signals

MATLAB provides numerous tools to generate basic signals that can be used to illustrate the effects of filters. In Chapter 5, we have generated a signal by adding together three sine waves with different amplitudes and periods. In the following example, the time vector is transposed for the purpose of generating column vectors.

```
t = (1:100)';
x = 2*sin(2*pi*t/50) + sin(2*pi*t/15) + 0.5*sin(2*pi*t/5);

plot(t,x), axis([0 100 -4 4])
```

Frequency-selective filters are very common in earth sciences. They are used for removing certain frequency bands from the data. As an example, we could design a filter that has the capability to suppress the portion of the signal with a periodicity of τ=15, whereas the other two cycles are unaffected. Such simple periodic signals can also be used to predict signal distortions of natural filters.

A *step function* is another basic input signal that can be used for exploring filter characteristics. It describes the transition from a value of one towards zero at a certain time.

```
t = (1:100)';
x = [ones(50,1);zeros(50,1)];

plot(t,x), axis([0 100 -2 2])
```

This signal can be used to study the effects of a filter on a sudden transition. An abrupt climate change could be regarded as an example. Most natural filters tend to smooth such a transition and smear it over a longer time period.

The *unit impulse* is the third important signal that we will use in the following examples. This signal equals zero for all times except for a single data point which equals one.

```
t = (1:100)';
x = [zeros(49,1);1;zeros(50,1)];

plot(t,x), axis([0 100 -4 4])
```

The unit impulse is the most popular synthetic signal for studying the performance of a filter. The output of the filter, the impulse response, describes the characteristics of a filter very well. Moreover, the output of a linear time-invariant filter can be described by the superposition of impulse responses that have been scaled by the amplitude of the input signal.

6.3 Linear Time-Invariant Systems

Filters can be described as systems with an input and output. Therefore, we first describe the characteristics of a more general system before we apply this theory to filters. Important characteristics of a system are

- *Continuity* – A system with continuous inputs and outputs is continuous. Most of the natural systems are continuous. However, after sampling natural signals we obtain discrete data series and model these natural systems as discrete systems, which have discrete inputs and outputs.

- *Linearity* – For linear systems, the output of the linear combination of several input signals

  $$x(t) = k_1 x_1(t) + k_2 x_2(t)$$

  is the same linear combination of the outputs:

  $$y(t) = k_1 y_1(t) + k_2 y_2(t)$$

  The important consequence of linearity is scaling and additivity (*superposition*). Input and output can be multiplied by a constant before or after transformation. Superposition allows to extract additive components of the input and transform these separately. Fortunately, many natural systems show a linear behavior. Complex linear signals such as additive harmonic components can be separated and transformed independently. Milankovitch cycles provide an example of linear superposition in paleoclimate records, although there is an ongoing debate about the validity of this assumption. Numerous nonlinear systems exist in nature that do not obey the properties of scaling and additivity. An example of such a linear system is

  ```
  x = (1:100)';
  y = 2*x;

  plot(x,y)
  ```

  An example of a nonlinear system is

  ```
  x = (-100:100)';
  y = x.^2;

  plot(x,y)
  ```

- *Time invariance* – The system output $y(t)$ does not change with a delay of the input $x(t+i)$. The system characteristics are constant with time. Unfortunately, natural systems often change their characteristics with time. For instance, benthic mixing or bioturbation depends on various environmental parameters such as nutrient supply. Therefore, the system's performance varies with time significantly. In such case, the actual input of the system is hard to determine from the output, e.g., to extract the actual climate signal from a bioturbated sedimentary record.

- *Invertibility* – An invertible system is a system where the original input signal can be reproduced from the system's output. This is an important property if unwanted signal distortions have to be corrected. Here, the known system is inverted and applied to the output to reconstruct the undisturbed input. As an example, a core logger measuring the magnetic susceptibility with a loop sensor integrates over a certain core interval with highest sensitivity at the location of the loop and decreasing sensitivity down- and up-core. The above system is also invertible, i.e., we can compute the input signal from the output signal by inverting the system. The inverse system of the above linear system is

```
x = (1:100)';
y = 0.5*x;

plot(t,y)
```

The nonlinear system

```
x = (-100:100)';
y = x.^2;

plot(x,y)
```

is not invertible. Since this system yields equal responses for different inputs, such as $y=+4$ for inputs $x=-2$ and $x=+2$, the input cannot be reconstructed from the output. A similar situation can also occur in linear systems, such as

```
x = (1:100)';
y = 0;

plot(x,y)
```

The output is zero for all inputs. Therefore, the output does not contain any information about the input.

- *Causality* – The system response only depends on present and past inputs $x(0)$, $x(-1)$, …, whereas future inputs $x(+1)$, $x(+2)$, … have no effect on the output $y(0)$. All realtime systems, such telecommunication systems, must be causal since they cannot have future inputs available to them. All systems and filters in MATLAB are indexed as causal. In earth sciences, however, numerous non-causal filters are used. Filtering images and signals extracted from sediment cores are examples where the future inputs are available at the time of filtering. Output signals have to be delayed after filtering to compensate the differences between causal and non-causal indexing.

- *Stability* – A system is stable if the output of a finite input is also finite. Stability is critical in filter design, where filters often have the disadvantage of provoking diverging outputs. In such cases, the filter design has to be revised and improved.

Linear time-invariant (*LTI*) *systems* as a special type of filters are very popular. Such systems have all the advantages that have been described above. They are easy to design and to use in many applications. The following chapters 6.4 to 6.9 describe the design, realization and application of LTI-type filters to extract certain frequency components of signals. These filters are mainly used to reduce the noise level in signals. Unfortunately, many natural systems do not behave as LTI systems. The signal-to-noise ratio often varies with time. Chapter 6.10 describes the application of adaptive filters that automatically adjust their characteristics in a time-variable environment.

6.4 Convolution and Filtering

The mathematical description of a system transformation is the convolution. Filtering is one application of the convolution process. A running mean of length five provides an example of such a simple filter. The output of an arbitrary input signal is

$$y(t) = \frac{1}{5}\sum_{k=-2}^{2} x(t-k)$$

The output $y(t)$ is simply the average of the five input values $x(t-2)$, $x(t-1)$, $x(t)$, $x(t+1)$ and $x(t+2)$. In other words, all the five consecutive input values are multiplied by a factor of 1/5 and summed to form $y(t)$. In this example, all input values are multiplied by the same factor, i.e., they are equally weighted. The five factors used in the above operation are also called filter weights b_k. The filter can be represented by the vector

```
b = [0.2 0.2 0.2 0.2 0.2]
```

consisting of the identical filter weights. Since this filter is symmetric, it does not shift the signal on the time axis. The only function of this filter is to smooth the signal. Therefore, running means of a given length are often used to smooth signals, mainly for cosmetic reasons. A modern spreadsheet software usually contains running means as a function for smoothing data series. The impact of the smoothing filter increases with increasing filter length.

The weights that a filter of arbitrary length may take can vary. As an example, let us assume an asymmetric filter of five weights.

```
  b = [0.05 0.08 0.14 0.26 0.47]
```

The sum of all of the filter weights is one. Therefore, it does not introduce energy to the signal. However, since it is highly asymmetric, it shifts the signal along the time axis, i.e., it introduces a phase shift.

The general mathematical representation of the filtering process is the *convolution*:

$$y(t) = \sum_{k=-N_1}^{N_2} b_k \cdot x(t-k)$$

where b_k is the vector of *filter weights*, N_1+N_2 is the *order of the filter*, which is the length of the filter reduced by one. Filters with five weights have an order of four, as in our example. In contrast to this format, MATLAB uses the engineering standard of indexing filters, i.e., filters are always defined as causal. Therefore, the convolution used by MATLAB is

$$y(t) = \sum_{k=0}^{N} b_k \cdot x(t-k)$$

where N is the order of the filter. A number of frequency-domain tools provided by MATLAB cannot simply be applied to non-causal filters that have been designed for applications in earth sciences. Hence, it is common to carry out phase corrections to simulate non-causality. For example, frequency-selective filters as introduced in Chapter 6.9 can be applied using the function `filtfilt`, which provides zero-phase forward and reverse filtering.

The functions `conv` and `filter` that provide digital filtering with MATLAB are best illustrated in terms of a simple running mean. The n elements of the vector $x(t_1), x(t_2), x(t_3), \ldots, x(t_n)$ are replaced by the arithmetic means of subsets of the input vector. For instance, a running mean over three elements computes the mean of inputs $x(t_{n-1}), x(t_n), x(t_{n+1})$ to obtain the output $y(t_n)$. We can easily illustrate this by generating a random signal

```
clear

t = (1:100)';
randn('seed',0);
x1 = randn(100,1);
```

designing a filter that averages three data points of the input signal

```
b1 = [1 1 1]/3;
```

and convolving the input vector with the filter

```
y1 = conv(b1,x1);
```

The elements of `b1` are the weights of the filter. In our example, all filter weights are the same and they equal 1/3. Note that the `conv` function yields a vector that has the length $n+m-1$, where m is the length of the filter.

```
m1 = length(b1);
```

We should explore the contents of our workspace to check for the length of the input and output of `conv`. Typing

```
whos
```

yields

```
Name         Size            Bytes  Class     Attributes
b1           1x3                24  double
m1           1x1                 8  double
t          100x1               800  double
x1         100x1               800  double
y1         102x1               816  double
```

Here, we see that the actual input series `x1` has a length of 100 data points, whereas the output `y1` has two more elements. Generally, convolution introduces $(m-1)/2$ data points at both ends of the data series. To compare input and output signal, we cut the output signal at both ends.

```
y1 = y1(2:101,1);
```

A more general way to correct the phase shifts of `conv` is

```
y1 = y1(1+(m1-1)/2:end-(m1-1)/2,1);
```

which of course works only for an odd number of filter weights. Then, we can plot both input and output signals for comparison. We also use `legend` to display a legend for the plot.

```
plot(t,x1,'b-',t,y1,'r-')
legend('x1(t)','y1(t)')
```

This plot illustrates the effect of the running mean on the original input series. The output `y1` is significantly smoother than the input signal `x1`. If we increase the length of the filter, we obtain an even smoother signal.

```
b2 = [1 1 1 1 1]/5;
m2 = length(b2);

y2 = conv(b2,x1);
y2 = y2(1+(m2-1)/2:end-(m2-1)/2,1);

plot(t,x1,'b-',t,y1,'r-',t,y2,'g-')
legend('x1(t)','y1(t)','y2(t)')
```

The next chapter introduces a more general description of filters.

6.5 Comparing Functions for Filtering Data Series

A very simple example of a *nonrecursive filter* was described in the previous section. The filter output $y(t)$ depends only on the filter input $x(t)$ and the filter weights b_k. Prior to introducing a more general description for linear time-invariant filters, we replace the function `conv` by `filter` that can be used also for *recursive filters*. In this case, the output $y(t_n)$ depends on the filter input $x(t)$, but also on previous elements of the output $y(t_{n-1})$, $y(t_{n-2})$, $y(t_{n-3})$ and so on (Chapter 6.6). First, we use `filter` for nonrecursive filters.

```
clear

t = (1:100)';
randn('seed',0);
x3 = randn(100,1);
```

We design a filter that averages five data points of the input signal.

```
b3 = [1 1 1 1 1]/5;
m3 = length(b3);
```

The input vector can be convolved with the function `conv`. The output is again corrected for the length of the data vector.

```
y3 = conv(b3,x3);
y3 = y3(1+(m3-1)/2:end-(m3-1)/2,1);
```

Although the function `filter` yields an output vector with the same length as the input vector, we have to correct the output as well. Here, the function `filter` assumes that the filter is causal. The filter weights are indexed n, $n-1$, $n-2$ and so on. Therefore, no future elements of the input vector, such as $x(n+1)$, $x(n+2)$ etc. are needed to compute the output $y(n)$. This is of great importance in electrical engineering, the classic field of application of MATLAB, where filters are often applied in real time. In earth sciences, however, in most applications the entire signal is available at the time of processing the data. Filtering the data series is done by

```
y4 = filter(b3,1,x3);
```

and afterwards the phase correction is carried out using

```
y4 = y4(1+(m3-1)/2:end-(m3-1)/2,1);
y4(end+1:end+m3-1,1) = zeros(m3-1,1);
```

which works only for an odd number of filter weights. This command simply shifts the output by `(m-1)/3` towards the lower end of the t-axis, then fills the end of the data series by zeros. Comparing the ends of both outputs illustrates the effect of this correction, where

```
y3(1:5,1)
y4(1:5,1)
```

yields

```
ans =
    0.3734
    0.4437
    0.3044
    0.4106
    0.2971

ans =
    0.3734
    0.4437
    0.3044
    0.4106
    0.2971
```

This was the lower end of the output. We see that both vectors `y3` and `y4` contain the same elements. Now we explorer the upper end of the data vector, where

```
y3(end-5:end,1)
y4(end-5:end,1)
```

causes the output

```
ans =
    0.2268
    0.1592
    0.3292
    0.2110
    0.3683
    0.2414

ans =
    0.2268
    0.1592
         0
         0
         0
         0
```

The vectors are identical up to element `y(end-m3+1)`, then the second vector `y4` contains zeros instead of true data values. Plotting the results with

```
subplot(2,1,1), plot(t,x3,'b-',t,y3,'g-')
subplot(2,1,2), plot(t,x3,'b-',t,y4,'g-')
```

or in one single plot,

```
plot(t,x3,'b-',t,y3,'g-',t,y4,'r-')
```

shows that the results of `conv` and `filter` are identical except for the upper end of the data vector. These observations are important for our next steps in signal processing, particularly if we are interested in leads and lags between various components of signals.

6.6 Recursive and Nonrecursive Filters

Now we expand the nonrecursive filters by a recursive component, i.e., the output $y(t_n)$ depends on the filter input $x(t)$, but also on previous output values $y(t_{n-1}), y(t_{n-2}), y(t_{n-3})$ and so on. This filter requires the nonrecursive filter weights b_i, but also the recursive filters weights a_i (Fig. 6.2). This filter can be described by the *difference equation*:

$$y(t) = \sum_{k=-N_1}^{N_2} b_k \cdot x(t-k) - \sum_{k=1}^{M} a_k \cdot y(t-k)$$

Whereas this is a non-causal version of the difference equation, MATLAB uses the causal indexing again,

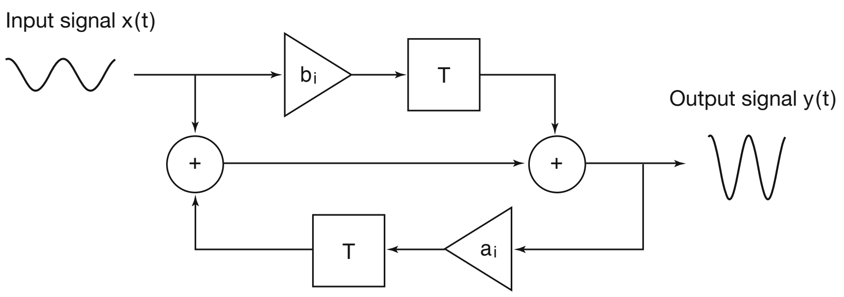

Fig. 6.2 Schematic of a linear time-invariant filter with an input $x(t)$ and an output $y(t)$. The filter is characterized by its weights a_i and b_i, and the delay elements T. Nonrecursive filters only have nonrecursive weights b_i, whereas the recursive filter also requires the recursive filters weights a_i.

$$y(t) = \sum_{k=0}^{N} b_k \cdot x(t-k) - \sum_{k=1}^{M} a_k \cdot y(t-k)$$

with the known problems in the design of zero-phase filters. The larger of the two quantities M and N_1+N_2 or N is the order of the filter.

We use the same synthetic input signal as in the previous example to illustrate the performance of a recursive filter.

```
clear
t = (1:100)';
randn('seed',0);
x5 = randn(100,1);
```

We filter this input using a recursive filter with a set of weights `a5` and `b5`,

```
b5 = [0.0048    0.0193    0.0289    0.0193    0.0048];
a5 = [1.0000   -2.3695    2.3140   -1.0547    0.1874];

m5 = length(b5);

y5 = filter(b5,a5,x5);
```

and correct the output for the phase

```
y5 = y5(1+(m5-1)/2:end-(m5-1)/2,1);
y5(end+1:end+m5-1,1) = zeros(m5-1,1);
```

Now we plot the results.

```
plot(t,x5,'b-',t,y5,'r-')
```

Obviously, this filter changes the signal dramatically. The output contains only low-frequency components, whereas all higher frequencies are eliminated. The comparison of the periodograms of the input and the output reveals that all frequencies above $f = 0.1$ corresponding to a period of $\tau = 10$ are suppressed.

```
[Pxx,F] = periodogram(x5,[],128,1);
[Pyy,F] = periodogram(y5,[],128,1);

plot(F,abs(Pxx),F,abs(Pyy))
```

Hence, we have now designed a frequency-selective filter, i.e., a filter that eliminates certain frequencies whereas other periodicities are relatively unaffected. The next chapter introduces tools to characterize a filter in the time and frequency domain that help to predict the effect of a frequency-selective filter on arbitrary signals.

6.7 Impulse Response

The impulse response is a very convenient way of describing the filter characteristics (Fig. 6.3). A useful property of the impulse response h in LTI systems involves the convolution of the input signal $x(t)$ with h to obtain the output signal $y(t)$.

$$y(t) = \sum_{k=-N_1}^{N_2} h_k \cdot x(t-k)$$

It can be shown that the impulse response h is identical to the filter weights in the case of nonrecursive filters, but is different for recursive filters. Alternatively, the convolution is often written in a short form:

$$y(t) = h(t) * x(t)$$

In many examples, the convolution in the time domain is replaced by a simple multiplication of the Fourier transforms $H(f)$ and $X(f)$ in the frequency domain.

$$Y(f) = H(f) \cdot X(f)$$

The output signal $y(t)$ in the time domain is then obtained by a reverse

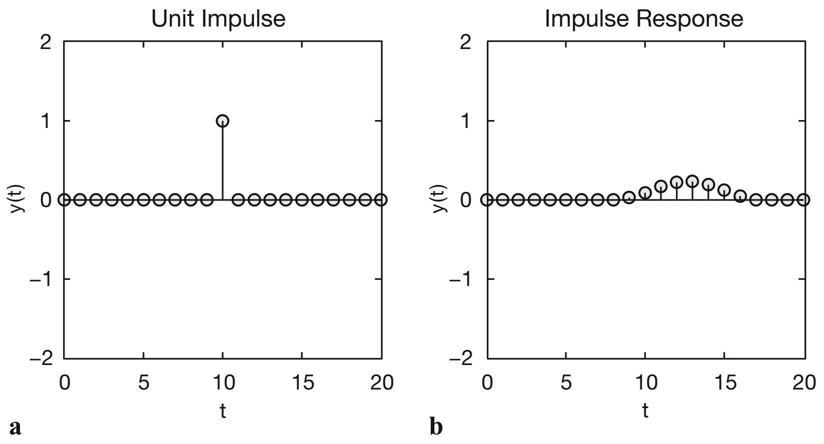

Fig. 6.3 Transformation of **a** a unit impulse to compute **b** the impulse response of a system. The impulse response is often used to describe and predict the performance of a filter.

Fourier transformation of $Y(f)$. The signals are often convolved in the frequency domain for simplicity of the multiplication as compared to a convolution in the time domain. However, the Fourier transformation itself introduces a number of artifacts and distortions and therefore, convolution in the frequency domain is not without problems. In the following examples we apply the convolution only in the time domain.

First, we generate an unit impulse:

```
clear
t = (0:20)';
x6 = [zeros(10,1);1;zeros(10,1)];

stem(t,x6), axis([0 20 -4 4])
```

The function `stem` plots the data sequence `x6` as stems from the x-axis terminated with circles for the data value. This might be a better way to plot digital data than using the continuous lines generated by `plot`. We now feed this to the filter and explore the output. The impulse response is identical to the weights of nonrecursive filters.

```
b6 = [1 1 1 1 1]/5;
m6 = length(b6);

y6 = filter(b6,1,x6);
```

We correct this for the phase shift of the function `filter` again, although this might not be important in this example.

```
y6 = y6(1+(m6-1)/2:end-(m6-1)/2,1);
y6(end+1:end+m6-1,1) = zeros(m6-1,1);
```

We obtain an output vector `y6` of the same length and phase as the input vector `x6`. We plot the results for comparison.

```
stem(t,x6)
hold on
stem(t,y6,'filled','r')
axis([0 20 -2 2])
```

In contrast to `plot`, the function `stem` accepts only one data series. Therefore, the second series `y6` is overlaid on the same plot using the function `hold`. The effect of the filter is clearly seen on the plot. It averages the unit impulse over a length of five elements. Furthermore, the values of the output equal the filter weights of `a6`, in our example 0.2 for all elements of `a6` and `y6`.

For a recursive filter, the output `y6` does not agree with the filter weights. Again, an impulse is generated first.

```
clear
t = (0:20)';
x7 = [zeros(10,1);1;zeros(10,1)];
```

Subsequently, an arbitrary recursive filter with weights of `a7` and `b7` is designed.

```
b7 = [0.0048    0.0193    0.0289    0.0193    0.0048];
a7 = [1.0000   -2.3695    2.3140   -1.0547    0.1874];

m7 = length(b7);

y7 = filter(b7,a7,x7);

y7 = y7(1+(m7-1)/2:end-(m7-1)/2,1);
y7(end+1:end+m7-1,1) = zeros(m7-1,1);
```

The stem plot of the input `x2` and the output `y2` shows an interesting impulse response:

```
stem(t,x7)
hold on
stem(t,y7,'filled','r')
axis([0 20 -2 2])
```

The signal is again smeared over a wider area. It is also shifted towards the right. Therefore, this filter not only affects the amplitude of the signal, but also shifts the signal towards lower or higher values. Phase shifts are usually unwanted characteristics of filters, although in some applications shifts along the time axis might be of particular interest.

6.8 Frequency Response

Next, we investigate the frequency response of a filter, i.e., the effect of a filter on the amplitude and phase of a signal (Fig. 6.4). The frequency response $H(f)$ of a filter is the Fourier transform of the impulse response $h(t)$. The absolute of the complex frequency response $H(f)$ is the magnitude response of the filter $A(f)$.

$$A(f) = \left|H(f)\right|$$

The argument of the complex frequency response $H(f)$ is the phase response of the filter.

$$\Phi(f) = \arg\left(H(f)\right)$$

Since MATLAB filters are all causal it is difficult to explore the phase of signals using the corresponding functions included in the Signal Processing Toolbox. The user's guide for this toolbox simply recommends to delay the filter output in the time domain by a fixed number of samples, as we have done it in the previous examples. As an example, a sine wave with a period of 20 and an amplitude of 2 is used as an input signal.

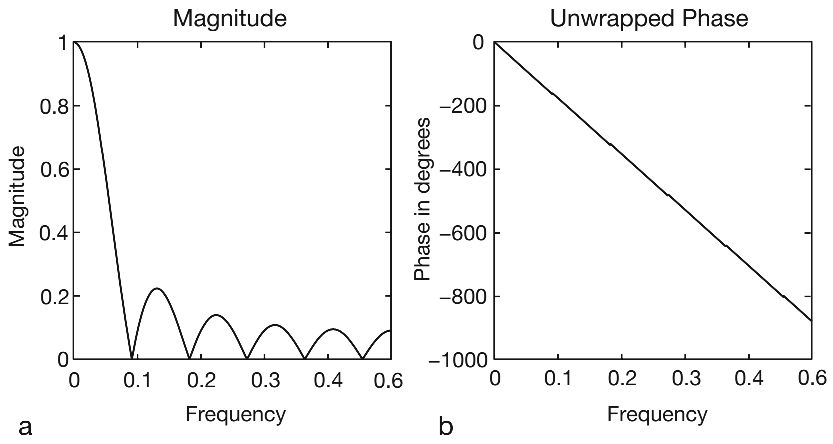

Fig. 6.4 a Magnitude and **b** phase response of a running mean over eleven elements.

```
clear
t = (1:100)';
x8 = 2*sin(2*pi*t/20);
```

A running mean over eleven elements is designed and this filter is applied to the input signal.

```
b8 = ones(1,11)/11;
m8 = length(b8);

y8 = filter(b8,1,x8);
```

The phase is corrected for causal indexing.

```
y8 = y8(1+(m8-1)/2:end-(m8-1)/2,1);
y8(end+1:end+m8-1,1) = zeros(m8-1,1);
```

Both input and output of the filter are plotted.

```
plot(t,x8,t,y8)
```

The filter obviously reduces the amplitude of the sine wave. Whereas the input signal has an amplitude of 2, the output has an amplitude of

```
max(y8)

ans =
    1.1480
```

The filter reduces the amplitude of a sine with a period of 20 by

```
1-max(y8(40:60))/2

ans =
    0.4260
```

i.e., approximately 43%. The elements 40 to 60 are used for computing the maximum value of `y8` to avoid edge effects. On the other hand, the filter does not affect the phase of the sine wave, i.e., both input and output are in phase.

The same filter, however, has a different impact on a different signal. Let us design another sine wave with a similar amplitude, but with a different period of 15.

```
clear
t = (1:100)';
x9 = 2*sin(2*pi*t/15);
```

Applying a similar filter and correcting the output for the phase shift of the function `filter` yields

```
b9 = ones(1,11)/11;
m9 = length(b9);

y9 = filter(b9,1,x9);

y9 = y9(1+(m9-1)/2:end-(m9-1)/2,1);
y9(end+1:end+m9-1,1) = zeros(m9-1,1);
```

The output is again in phase with the input, but the amplitude is dramatically reduced as compared to the input.

```
plot(t,x9,t,y9)

1-max(y9(40:60))/2

ans =
    0.6768
```

The running mean over eleven elements reduces the amplitude of this signal by 67%. More generally, the filter response obviously depends on the frequency of the input. The frequency components of a more complex signal containing multiple periodicities are affected in a different way. The frequency response of a filter

```
clear
b10 = ones(1,11)/11;
```

can be computed using the function `freqz`.

```
[h,w] = freqz(b10,1,512);
```

The function `freqz` returns the complex frequency response `h` of the digital filter `b10`. The frequency axis is normalized to π. We transform the frequency axis to the true frequency values. The true frequency values are `w` times the sampling frequency, which is one in our example, divided by 2*`pi`.

```
f = 1*w/(2*pi);
```

Next, we calculate the magnitude of the frequency response and plot the magnitude over the frequency.

```
magnitude = abs(h);

plot(f,magnitude)
xlabel('Frequency'), ylabel('Magnitude')
title('Magnitude')
```

This plot can be used to predict the magnitude of the filter for any frequency of an input signal. An exact value of the magnitude can also be obtained by simple interpolation of the magnitude,

```
1-interp1(f,magnitude,1/20)

ans =
    0.4260
```

which is the expected ca. 43% reduction of the amplitude of a sine wave with period 20. The sine wave with period 15 experiences an amplitude reduction of

```
1-interp1(f,magnitude,1/15)

ans =
    0.6751
```

i.e., around 68% similar to the value observed at the beginning. The frequency response can be calculated for all kinds of filters. It is a valuable tool to predict the effects of a filter on signals in general. The phase response can also be calculated from the complex frequency response of the filter (Fig. 6.4):

```
phase = 180*angle(h)/pi;

plot(f,phase)
xlabel('Frequency'), ylabel('Phase in degrees')
title('Phase')
```

The phase angle is plotted in degrees. We observe frequent 180° jumps in this plot that are an artifact of the `arctangent` function inside the function `angle`. We can unwrap the phase response to eliminate the 180° jumps using the function `unwrap`.

```
plot(f,unwrap(phase))
xlabel('Frequency'), ylabel('Phase in degrees')
title('Phase')
```

Since the filter has a linear phase response, no shifts of the frequency components of the signal occur relative to each other. Therefore, we would not expect any distortions of the signal in the frequency domain. The phase shift of the filter can be computed using

```
interp1(f,unwrap(phase),1/20) * 20/360

ans =
   -5.0000
```

and

```
interp1(f,unwrap(phase),1/15) * 15/360

ans =
   -5.0000
```

respectively. Since MATLAB uses causal indexing for filters, the phase needs to be corrected, similar to the delayed output of the filter. In our example, we used a filter of the length eleven. We have to correct the phase by (11–1)/2=5. This suggests a zero phase shift of the filter for both frequencies.

This also works for recursive filters. Assume a simple sine wave with period 8 and the previously employed recursive filter.

```
clear
t = (1:100)';
x11 = 2*sin(2*pi*t/8);

b11 = [0.0048    0.0193    0.0289    0.0193    0.0048];
a11 = [1.0000   -2.3695    2.3140   -1.0547    0.1874];

m11 = length(b11);

y11 = filter(b11,a11,x11);
```

Correct the output for the phase shift introduced by causal indexing and plot both input and output signals.

```
y11= y11(1+(m11-1)/2:end-(m11-1)/2,1);
y11(end+1:end+m11-1,1) = zeros(m11-1,1);

plot(t,x11,t,y11)
```

The magnitude is reduced by

```
1-max(y11(40:60))/2

ans =
    0.6465
```

which is also supported by the magnitude response

```
[h,w] = freqz(b11,a11,512);

f = 1*w/(2*pi);

magnitude = abs(h);

plot(f,magnitude)
xlabel('Frequency'), ylabel('Magnitude')
title('Magnitude Response')
```

```
1-interp1(f,magnitude,1/8)

ans =
    0.6462
```

The phase response

```
phase = 180*angle(h)/pi;

f = 1*w/(2*pi);

plot(f,unwrap(phase))
xlabel('Frequency'), ylabel('Phase in degrees')
title('Magnitude Response')

interp1(f,unwrap(phase),1/8) * 8/360

ans =
    -5.0144
```

must again be corrected for causal indexing. The sampling interval was one, the filter length is five. Therefore, we have to add $(5-1)/2=2$ to the phase shift of –5.0144. This suggests a corrected phase shift of –3.0144, which is exactly the delay seen on the plot.

```
plot(t,x11,t,y11), axis([30 40 -2 2])
```

The next chapter gives an introduction to the design of filters with a desired frequency response. These filters can be used to amplify or suppress different components of arbitrary signals.

6.9 Filter Design

Now we aim to design filters with a desired frequency response. Firstly, a synthetic signal with two periods, 50 and 15, is generated. The power-spectrum of the signal shows the expected peaks at the frequencies 0.02 and ca. 0.07.

```
t = 0 : 1000;
x12 = 2*sin(2*pi*t/50) + sin(2*pi*t/15);

plot(t,x12), axis([0 200 -4 4])

[Pxx,f] = periodogram(x12,[],1024,1);

plot(f,abs(Pxx))
xlabel('Frequency')
ylabel('Power')
```

We add some gaussian noise with amplitude one and explore the signal and its periodogram.

```
xn12 = x12 + randn(1,length(t));

plot(t,xn12), axis([0 200 -4 4])

[Pxx,f] = periodogram(xn12,[],1024,1);

plot(f,abs(Pxx))
xlabel('Frequency')
ylabel('Power')
```

The Butterworth filter design technique is a widely-used method to create filters of any order with a lowpass, highpass, bandpass and bandstop configuration (Fig. 6.5). In our example, we like to design a five-order lowpass filter with a cutoff frequency of 0.08. The inputs of the function `butter` are the order of the filter and the cutoff frequency normalized to the Nyquist frequency, which is 0.5 in our example, that is half of the sampling frequency.

```
[b12,a12] = butter(5,0.08/0.5);
```

The frequency characteristics of the filter show a relatively smooth transition from the passband to the stopband, but the advantage of the filter is its low order.

```
[h,w] = freqz(b12,a12,1024);
f = 1*w/(2*pi);

plot(f,abs(h)), grid
xlabel('Frequency')
ylabel('Magnitude')
```

We can again apply the filter to the signal by using the function `filter`. However, frequency selective filters such as lowpass, highpass, bandpass and bandstop are designed to suppress certain frequency bands, whereas phase shifts should be avoided. The function `filtfilt` provides zero-phase forward and reverse digital filtering. After filtering in the forward direction, the filtered sequence is reversed and it runs back through the filter. The magnitude of the signal is not affected by this operation, since it is either 0 or 100% of the initial amplitude, depending on the frequency. In contrast, all phase shifts introduced by the filter are zeroed by the forward and reverse application of the same filter. This function also helps to overcome the problems with causal indexing of filters in MATLAB. It eliminates the phase differences of the causal vs. non-causal versions of the same filter.

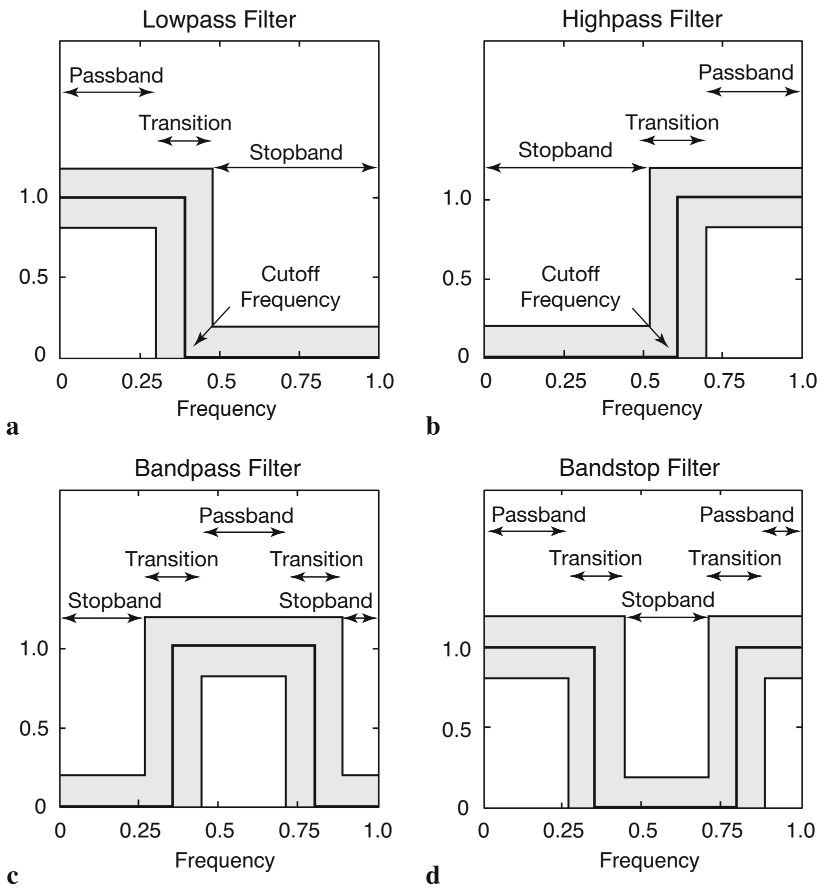

Fig. 6.5 Frequency response of the fundamental types of frequency-selective filters. **a** Lowpass filter to suppress the high-frequency component of a signal. In earth sciences, such filters are often used to suppress high-frequency noise in a low-frequency signal. **b** Highpass filter are employed to remove all low frequencies and trends in natural data. **c-d** Bandpass and bandstop filters extract or suppress a certain frequency band. Whereas the solid line in all graphs depicts the ideal frequency response of a frequency-selective filter, the gray band shows the tolerance for a low-order design of such a filter. In practice, the frequency response lies within the gray band.

Filtering and plotting the results clearly illustrates the effects of the filter.

```
xf12 = filtfilt(b12,a12,xn12);

plot(t,xn12,'b-',t,xf12,'r-')
axis([0 200 -4 4])
```

One might now wish to design a new filter with a more rapid transition from passband to stopband. Such a filter needs a higher order. It needs to have a larger number of filter weights. We now create a 15-order Butterworth filter as an alternative to the above filter.

```
[b13,a13] = butter(15,0.08/0.5);

[h,w] = freqz(b13,a13,1024);

f = 1*w/(2*pi);

plot(f,abs(h)), grid
xlabel('Frequency')
ylabel('Magnitude')
```

The frequency response is clearly improved. The entire passband is relatively flat at a value of 1.0, whereas the stopband is approximately zero everywhere. Next, we modify our input signal by introducing a third period of 5. This signal is then used to illustrate the operation of a Butterworth bandstop filter.

```
x14 = 2*sin(2*pi*t/50) + sin(2*pi*t/15) + 0.5*sin(2*pi*t/5);
plot(t,x14), axis([0 200 -4 4])

[Pxx,f] = periodogram(x14,[],1024,1);

plot(f,abs(Pxx))
```

The new Butterworth filter is a bandstop filter. The stopband of the filter is between the frequencies 0.06 and 0.08. It can therefore be used to suppress the period of 15 corresponding to a frequency of approximately 0.07.

```
xn14 = x14 + randn(1,length(t));

[b14,a14] = butter(5,[0.06 0.08]/0.5,'stop');
xf14 = filtfilt(b14,a14,x14);

[Pxx,f] = periodogram(xf14,[],1024,1);

plot(f,abs(Pxx))

plot(t,xn14,'b-',t,xf14,'r-'), axis([0 200 -4 4])
```

The plots show the effect of this filter. The frequency band between 0.06 and 0.08, and therefore also the frequency of 0.07 was successfully removed from the signal.

6.10 Adaptive Filtering

The fixed filters used in the previous chapters make the basic assumption that the signal degradation is known and it does not change with time. In most applications, however, an *a priori* knowledge of the signal and noise statistical characteristics is usually not available. In addition, both the noise level and the variance of the genuine signal can be highly nonstationary with time, e.g., stable isotope records during the glacial-interglacial transition. Fixed filters thus cannot be used in a nonstationary environment without a knowledge of the signal-to-noise ratio.

In contrast, adaptive filters widely used in the telecommunication industry could help to overcome these problems. An adaptive filter is an inverse modeling process, which iteratively adjusts its own coefficients automatically without requiring any *a priori* knowledge of signal and noise. The operation of an adaptive filter includes, (1) a filtering process, the purpose of which is to produce an output in response to a sequence of data, and (2) an adaptive process providing a mechanism for the adaptive control of the filter weights (Haykin 1991).

In most practical applications, the adaptive process is oriented towards minimizing an error signal or cost function e. The estimation error e at an instant i is defined by the difference between some desired response d_i and the actual filter output y_i, that is the filtered version of a signal x_i, as shown by

$$e_i = d_i - y_i$$

where $i=1, 2, \ldots, N$ and N is the length of the input data vector. In the case of a nonrecursive filter characterized by the vector of filter weights W with f elements, the filter output y_i is given by the inner product of the transposed vector W and the input vector X_i.

$$y_i = W^T \cdot X_i$$

The selection of the desired response d that is used in the adaptive process depends the application. Traditionally, d is a combined signal that contains a signal s and random noise n_0. The signal x contains a noise n_1 uncorrelated with the signal s but correlated in some unknown way to the noise n_0. In noise canceling systems, the practical objective is to produce a system output y that is a best fit in the least-squares sense to the signal d.

Different approaches have been developed to solve this multivariate minimum error optimization problem (e.g., Widrow and Hoff 1960, Widrow et al. 1975, Haykin 1991). Selection of one algorithm over another is influenced by various factors: the rate of convergence (number of adaptive steps required for the algorithm to converge close enough to an optimum solution), misadjustment (measure of the amount by which the final value of the mean-squared error deviates from the minimum squared error of an optimal filter, e.g., Wiener 1945, Kalman and Bucy 1961), and tracking (the capability of the filter to work in a nonstationary environment, i.e., to track changing statistical characteristics of the input signal) (Haykin 1991).

The simplicity of the least-mean-squares (LMS) algorithm, originally developed by Widrow and Hoff (1960), has made it the benchmark against which other adaptive filtering algorithms are tested. For applications in earth sciences, we use this filter to extract the noise from two signals *S* and *X*, both containing the same signal *s*, but uncorrelated noise n_1 and n_2 (Hattingh 1988). As an example, consider a simple duplicate set of measurements on the same material, e.g., two parallel stable isotope records from the same foraminifera species. What you will expect are two time-series with *N* elements containing the same desired signal overlain by different uncorrelated noise. The first record is used as the primary input *S*

$$S = (s_1, s_2, \ldots, s_N)$$

and the second record is the reference input *X*.

$$X = (x_1, x_2, \ldots, x_N)$$

As demonstrated by Hattingh (1988), the required noise-free signal can be extracted by filtering the reference input *X* using the primary input *S* as the desired response *d*. The minimum error optimization problem is solved by the norm least-mean-square. The mean-squared error e_i^2 is a second-order function of the weights in the nonrecursive filter. The dependence of e_i^2 on the unknown weights may be seen as a multidimensional paraboloid with a uniquely defined minimum point. The weights corresponding to the minimum point of this error performance surface define the optimum Wiener solution (Wiener 1945). The value computed for the weight vector *W* using the LMS algorithm represents an estimator whose expected value approaches the Wiener solution as the number of iterations approaches infinity (Haykin 1991). Gradient methods are used to reach the minimum point of the error performance surface. For simplification of the optimization problem, Widrow and Hoff (1960) developed an approximation for the required gradient function that can be computed directly from the data. This leads to a simple relation for updating the filter-weight vector *W*.

$$W_{i+1} = W_i + 2 \cdot u \cdot e_i \cdot X_i$$

The new parameter estimate W_{i+1} is based on the previous set of filter weights W_i plus a term, which is the product of a bounded step size u, a function of the input state X_i and a function of the error e_i. In other words, error e_i calculated from the previous step is fed back to the system to update filter coefficients for the next step (Fig. 6.6). The fixed convergence factor u regulates the speed and stability of adaption. A small value ensures a higher accuracy, but more data are needed to teach the filter to reach the optimum solution. In the modified version of the LMS algorithm by Hattingh (1988), this problem is overcome by feeding the data back so that the canceler can have another chance to improve its own coefficients and adapt to the changes in the data.

In the following function `canc`, each of these loops is called an iteration since many of these loops are required to achieve optimal results. This algorithm extracts the noise-free signal from two vectors `x` and `s` containing the correlated signal and uncorrelated noise. As an example, we generate two signals containing the same sine wave, but different gaussian noise.

```
x = 0 : 0.1 : 100;
y = sin(x);
yn1 = y + 0.2*randn(size(y));
yn2 = y + 0.2*randn(size(y));

plot(x,yn1,x,yn2)
```

Save the following code in a text file *canc.m* and include it into the search path. The algorithm `canc` formats both signals, feeds them into the filter loop,

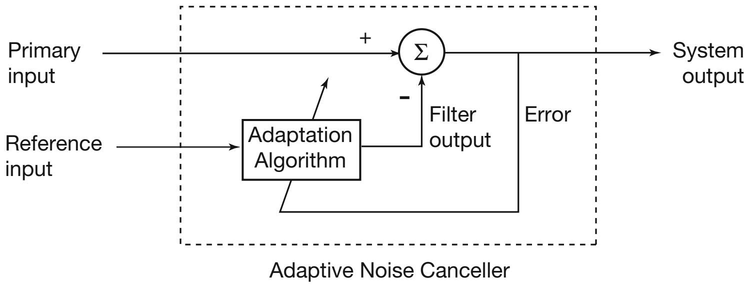

Fig. 6.6 Schematic of an adaptive filter. Each iteration involves a new estimate of the filter weights W_{i+1} based on the previous set of filter weights W_i plus a term which is the product of a bounded step size u, a function of the filter input X_i, and a function of the error e_i. In other words, error e_i calculated from the previous step is fed back to the system to update filter coefficients for the next step (modified from Trauth 1998).

corrects the signals for phase shifts and formats the signals for the output.

```
function [zz,yy,ee] = canc(x,s,u,l,iter)
% CANC Correlated Adaptive Noise Canceling
[n1,n2] = size(s); n = n2; index = 0;        % Formatting
if n1 > n2
    s = s'; x = x'; n = n1; index = 1;
end
w(1:l) = zeros(1,l); e(1:n) = zeros(1,n);    % Initialization
xx(1:l) = zeros(1,l); ss(1:l) = zeros(1,l);
z(1:n) = zeros(1,n); y(1:n) = zeros(1,n);
ors = s; ms(1:n) = mean(s) .* ones(size(1:n));
s = s - ms; x = x - ms; ors = ors - ms;
for it = 1 : iter                            % Iterations
    for I = (l+1) : (n+1)                    % Filter loop
        for k = 1 : l
            xx(k) = x(I-k); ss(k) = s(I-k);
        end
        for J = 1 : l
            y(I-1) = y(I-1) + w(J) .* xx(J);
            z(I-1) = z(I-1) + w(J) .* ss(J);
        end
            e(I-1) = ors(I-1-(fix(l/2)))-y(I-1);
        for J = 1 : l
            w(J) = w(J) + 2.*u.*e(I-1).*xx(J);
        end
    end                                      % End filter loop
    for I = 1 : n                            % Phase correction
        if I <= fix(l/2)
            yy(I) = 0; zz(I) = 0; ee(I) = 0;
        elseif I > n-fix(l/2)
            yy(I) = 0; zz(I) = 0; ee(I) = 0;
        else
            yy(I) = y(I+fix(l/2));
            zz(I) = z(I+fix(l/2));
            ee(I) = abs(e(I+fix(l/2)));
        end
            yy(I) = yy(I) + ms(I);
            zz(I) = zz(I) + ms(I);
    end                                      % End phase correction
    y(1:n) = zeros(size(1:n));
    z(1:n) = zeros(size(1:n));
    mer(it) = mean(ee((fix(l/2)):(n-fix(l/2))).^2);
end                                          % End iterations
if index == 1                                % Reformatting
    zz = zz'; yy = yy'; ee = ee';
end
```

The required inputs are the signals `x` and `s`, the step size `u`, the filter length `l` and the number of iterations `iter`. In our example, the two noisy signals are `yn1` and `yn2`. For instance, we choose a filter with `l=5` filter weights. A value of `u` in the range of $0 < u < 1/\lambda_{max}$ where λ_{max} is the largest eigenvalue of the autocorrelation matrix of the reference input, leads to reasonable results (Haykin 1991) (Fig. 6.7). The value of `u` is computed by

```
k = kron(yn1,yn1');
u = 1/max(eig(k))
```

which yields

```
u =
    0.0019
```

We now run the adaptive filter `canc` for 20 iterations and use the above value of `u`.

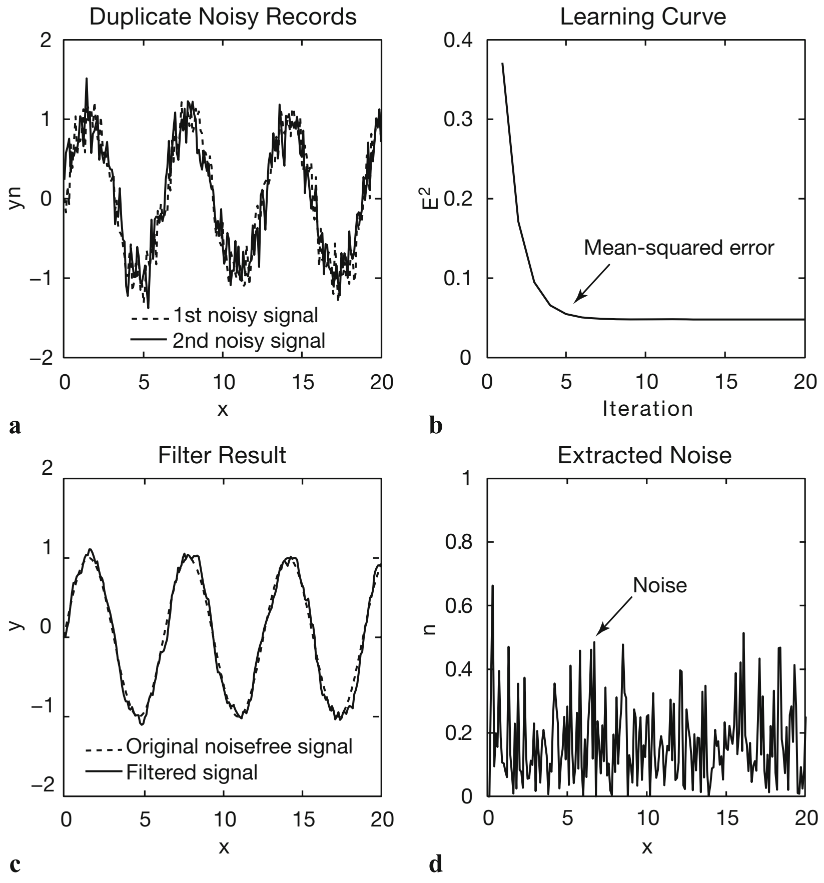

Fig. 6.7 Output of the adaptive filter. **a** The duplicate records corrupted by uncorrelated noise are fed into the adaptive filter with 5 weights with a convergence factor of 0.0019. After 20 iterations, the filter yields the **b** learning curve, **c** the noisefree record and **d** the noise extracted from the duplicate records.

```
[z,e,mer] = canc(yn1,yn2,0.0019,5,20);
```

The evolution of the mean-squared error

```
plot(mer)
```

illustrates the performance of the adaptive filter, although the chosen step size `u=0.0019` obviously leads to a relatively fast convergence. In most examples, a smaller step size decreases the rate of convergence, but increases the quality of the final result. We therefore reduce `u` by one order of magnitude and run the filter again with more iterations.

```
[z,e,mer] = canc(yn1,yn2,0.0001,5,20);
```

The plot of the mean-squared error against the iterations

```
plot(mer)
```

now convergences after around six iterations. We now compare the filter output with the original noise-free signal.

```
plot(x,y,'b',x,z,'r')
```

This plot shows that the noise level of the signal has been reduced dramatically by the filter. Finally, the plot

```
plot(x,e,'r')
```

shows the noise extracted from the signal. In practice, the user should vary the parameters `u` and `l` to obtain the optimum result.

The application of this algorithm has been demonstrated on duplicate oxygen-isotope records from ocean sediments (Trauth 1998). The work by Trauth (1998) illustrates the use of the modified LMS algorithm, but also another type of adaptive filters, the recursive least-squares (RLS) algorithm (Haykin 1991) in various environments.

Recommended Reading

Alexander ST (1986) Adaptive Signal Processing: Theory and Applications. Springer, Berlin Heidelberg New York
Buttkus B (2000) Spectral Analysis and Filter Theory in Applied Geophysics. Springer, Berlin Heidelberg New York
Cowan CFN, Grant PM (1985) Adaptive Filters. Prentice Hall, Englewood Cliffs, New Jersey

Grünigen DH (2004) Digitale Signalverarbeitung, mit einer Einführung in die kontinuierlichen Signale und Systeme, Dritte bearbeitete und erweiterte Auflage. Fachbuchverlag Leipzig, Leipzig

Hattingh M (1988) A new Data Adaptive Filtering Program to Remove Noise from Geophysical Time- or Space Series Data. Computers & Geosciences 14(4):467–480

Haykin S (2003) Adaptive Filter Theory. Prentice Hall, Englewood Cliffs, New Jersey

Kalman R, Bucy R (1961) New Results in Linear Filtering and Prediction Theory. ASME Tans. Ser. D Jour. Basic Eng. 83:95–107

Sibul LH (1987) Adaptive Signal Processing. IEEE Press

The Mathworks (2006) Signal Processing Toolbox User's Guide – For the Use with MATLAB®. The MathWorks, Natick, MA

Trauth MH (1998) Noise Removal from Duplicate Paleoceanographic Time-Series: The Use of adaptive Filtering Techniques. Mathematical Geology 30(5):557–574

Widrow B, Hoff Jr. M (1960) Adaptive Switching Circuits. IRE WESCON Conv. Rev. 4:96–104

Widrow B, Glover JR, McCool JM, Kaunitz J, Williams CS, Hearn RH, Zeidler JR, Dong E, Goodlin RC (1975) Adaptive Noise Cancelling: Principles and Applications. Proc. IEEE 63(12):1692–1716

Wiener N (1949) Extrapolation, Interpolation and Smoothing of Stationary Time Series, with Engineering Applications. MIT Press, Cambridge, Mass (reprint of an article originally issued as a classified National Defense Research Report, February, 1942)