Volume 2 of the second edition of the fully revised and updated Digital Signal and Image Processing using MATLAB® is essentially a collection of examples and exercises which also presents applications of digital signal- or image processing, and techniques which were not touched upon in the previous volume. It will be of particular benefit to readers who already possess a good knowledge of MATLAB®, a command of the fundamental elements of digital signal processing and who are familiar with both the fundamentals of continuous-spectrum spectral analysis and who have a certain mathematical knowledge concerning Hilbert spaces.

More than 200 programs and functions are provided in the MATLAB® language, with useful comments and guidance, to enable numerical experiments to be carried out, thus allowing readers to develop a deeper understanding of both the theoretical and practical aspects of this subject.

Gérard Blanchet is Professor at Ecole Nationale Supérieure des Télécommunications, Paris, France. In addition to his research, teaching and consulting activities, he is the author of several books on automatic control systems, digital signal processing and computer architecture. He also develops tools and methodologies to improve knowledge acquisition in various fields.

Maurice Charbit is Professor at Ecole Nationale Supérieure des Télécommunications, Paris, France, where he teaches several courses in signal processing and digital communications. His research interests include statistics, speech and image processing.
Digital Signal and Image Processing using MATLAB®
Contents

Foreword ix

Notations and Abbreviations xi

Chapter 1 Recap on Digital Signal Processing 1
1.1 The sampling theorem ............................................ 2
1.2 Spectral contents .................................................. 7
  1.2.1 Discrete-time Fourier transform (DTFT) .................. 7
  1.2.2 Discrete Fourier transform (DFT) ......................... 8
1.3 Case of random signals .......................................... 10
1.4 Example of the Dual Tone Multi-Frequency (DTMF) ............ 11

Chapter 2 Additional Information About Filtering 15
2.1 Filter implementation ............................................ 15
  2.1.1 Examples of filter structures .............................. 16
  2.1.2 Distributing the calculation load in an FIR filter ........ 20
  2.1.3 FIR block filtering ......................................... 21
  2.1.4 FFT filtering ................................................ 23
2.2 Filter banks ....................................................... 29
  2.2.1 Decimation and expansion ................................... 30
  2.2.2 Filter banks ................................................ 34
2.3 Ripple control .................................................... 42
  2.3.1 Principle ..................................................... 42
  2.3.2 Programming ................................................ 44

Chapter 3 Image Processing 51
3.1 A little geometry ................................................. 51
  3.1.1 3D object .................................................... 51
  3.1.2 Calibration of cameras ...................................... 53
3.2 Pyramidal decompositions ....................................... 62
  3.2.1 Pyramidal decomposition given by Burt and Adelson ........ 64
3.2.2 Pyramidal decomposition using a Haar transformation ....... 65
3.2.3 Stepwise decomposition (lifting scheme) ................. 66

Chapter 4  Numerical Calculus and Simulation 71
4.1 Simulation of continuous-time systems ....................... 71
4.1.1 Simulation by approximation .......................... 71
4.1.2 Exact model simulation ................................ 72
4.2 Solving of ordinary differential equations (ODEs) .......... 76
4.2.1 Conversion from continuous to discrete time ............ 76
4.2.2 Linear case, continuous-time solution .................. 78
4.2.3 Remarks on the Runge–Kutta methods .................. 81
4.3 Systems of equations and zero-seeking ...................... 88
4.3.1 Zeros of a function using the Newton method ........... 88
4.3.2 Roots of a polynomial with the Newton–Raphson method 89
4.3.3 Systems of nonlinear equations ......................... 90
4.4 Interpolation .................................................. 91
4.4.1 Thiele’s interpolation .................................. 92
4.4.2 Another decomposition in continuous fractions .......... 95
4.4.3 Natural cubic splines .................................. 96
4.5 Solving of linear systems ................................... 100
4.5.1 Jacobi method ......................................... 100
4.5.2 Relaxation method .................................... 101
4.5.3 Cholesky factorization .................................. 102

Chapter 5  Speech Processing 105
5.1 A speech signal model ...................................... 105
5.1.1 Overview ............................................. 105
5.1.2 A typology of vocal sounds ............................ 106
5.1.3 The AR model of speech production .................... 107
5.1.4 Compressing a speech signal .......................... 113
5.2 Dynamic Time Warping .................................... 116
5.2.1 The DTW algorithm .................................. 117
5.2.2 Examples of pathfinding rules ......................... 118
5.2.3 Cepstral coefficients .................................. 119
5.3 Modifying the duration of an audio signal .................. 120
5.3.1 PSOLA ............................................. 121
5.3.2 Phase vocoder ....................................... 123
5.4 Eliminating the impulse noise .............................. 124
5.4.1 The signal model ..................................... 125
5.4.2 Click detection ....................................... 126
5.4.3 Restoration .......................................... 128
Chapter 6  Selected Topics  
6.1  Tracking the cardiac rhythm of the fetus  
   6.1.1  Objectives  
   6.1.2  Separating the EKG signals  
   6.1.3  Estimating cardiac rhythms  
6.2  Extracting the contour of a coin  
6.3  Constrained optimization and Lagrange multipliers  
   6.3.1  Equality-constrained optimization  
   6.3.2  Quadratic problem with linear inequality constraints  
   6.3.3  Portfolio optimization  
6.4  Principal Component Analysis (PCA)  
   6.4.1  Determining the principal components  
   6.4.2  2-Dimension PCA  
   6.4.3  Linear Discriminant Analysis  
6.5  GPS positioning  
6.6  The Viterbi algorithm  
   6.6.1  Convolutional non-recursive encoder  
   6.6.2  Decoding and hard decision  

Chapter 7  Hints and Solutions  
H1  Reminders on digital signal-processing  
H2  Additional information on filtering  
H3  Image Processing  
H4  Numerical calculus  
H5  Speech processing  
H6  Selected topics  

Chapter 8  Appendix  
A1  A few properties of a matrix  
A2  A few relations for matrices  

Bibliography  
Index
Foreword

This book represents the continuation to Digital Signal and Image Processing: Fundamentals. It is assumed that the reader possesses a good knowledge of the programming language MATLAB® and a command of the fundamental elements of digital signal processing: the usual transforms (the Discrete Time Fourier Transform (DTFT), the Discrete Fourier Transform and the z-Transform), the properties of deterministic and random signals, and digital filtering. Readers will also need to be familiar with the fundamentals of continuous-spectrum spectral analysis and have a certain amount of mathematical knowledge concerning vector spaces.

In order to prevent the reading becoming a penance, we will offer a few reminders of the basics wherever necessary. This book is essentially a collection of examples, exercises and case studies. It also presents applications of digital signal- or image processing, and techniques which were not touched upon in the previous volume.

Recap on digital signal processing

This section is devoted to the definitions and properties of the fundamental transforms used in digital signal processing: Fourier transform, discrete time Fourier transform and discrete Fourier transform. It concludes with a classic example which enables us to put some known results into practice.

Filter implementation

This section deals with the structures of filters, the introduction of parallelism into the filtering operations (block filtering and filter banks) and, by way of an example, the Parks–McClellan method for FIR filter synthesis (finite impulse response).

Image processing

The section given over to images offers a few geometrical concepts relating to the representation of 3D objects in a 2D space. Therein, we deal with problems
of calibration of cameras. In addition, image compression is also discussed, with
the use of examples (pyramidal decompositions, lifting scheme).

Digital calculus and simulation

This section deals with the algorithms used in most domains in digital process-
ing, and therefore far beyond mere signal processing. It only touches on the
domain using a few examples of methods applied to problems of simulation,
resolution of differential equations, zero-seeking, interpolation and iterative
methods for solving linear systems.

Speech processing

After a brief introduction to speech production, we will discuss the represen-
tation of a speech signal by an autoregressive model, and its application to
compression. Next we will give the descriptions of the techniques widely used
in this field (Dynamic Time Warping and PSOLA) and, finally, an example of
application with “decrackling” for audio recordings.

Selected topics

This last chapter presents case studies that go a little further in depth than
the examples described in the previous sections. “Tracking the cardiac rhythm
of the fetus” and “Extracting the contour of a coin” are classic examples of
the application of the least squares method. Principal component analysis and
linear discriminant analysis are basic methods for the classification of objects
(in a very broad sense).

The section devoted to optimization under constraints could have been part
of the section on numerical methods. The method of Lagrange multipliers is
encountered in a multitude of applications. In terms of applications, we present
the case of optimization of a stock portfolio.

We conclude with the example of the Viterbi algorithm for the hard de-
coding of convolutional codes. This algorithm is, in fact, a particular case for
searching for the shortest possible path in a lattice.
Notations and Abbreviations

∅ empty set

\[ \sum_{k,n} = \sum_k \sum_n \]

\[ \text{rect}_T(t) = \begin{cases} 1 & \text{when } |t| < T/2 \\ 0 & \text{otherwise} \end{cases} \]

\[ \text{sinc}(x) = \frac{\sin(\pi x)}{\pi x} \]

\[ 1(x \in A) = \begin{cases} 1 & \text{when } x \in A \\ 0 & \text{otherwise} \end{cases} \quad \text{(indicator function of } A) \]

\[ (a, b] = \{ x : a < x \leq b \} \]

\[ \delta(t) \]

\[ \begin{cases} \text{Dirac distribution when } t \in \mathbb{R} \\ \text{Kronecker symbol when } t \in \mathbb{Z} \end{cases} \]

\[ \text{Re}(z) \quad \text{real part of } z \]

\[ \text{Im}(z) \quad \text{imaginary part of } z \]

\[ \lfloor x \rfloor \quad \text{integer part of } x \]

i or j = \sqrt{-1}

\[ x(t) \Leftrightarrow X(f) \quad \text{Fourier transform} \]

\[ (x \ast y)(t) \quad \text{continuous time convolution} \]

\[ = \int_{\mathbb{R}} x(u)y(t-u)du \]

\[ (x \circledast y)(t) \quad \text{discrete time convolution} \]

\[ = \sum_{u \in \mathbb{Z}} x(u)y(t-u) = \sum_{u \in \mathbb{Z}} x(t-u)y(u) \]

\[ y^{(n)}(t) = \frac{d^n y(t)}{dt^n}, \text{nth order derivative} \]
### Mathematical Symbols and Notations

- $\mathbf{x}$ or $\underline{x}$: vector $\mathbf{x}$
- $\mathbf{I}_N$: $(N \times N)$-dimension identity matrix
- $A^*$: complex conjugate of $A$
- $A^T$: transpose of $A$
- $A^H$: transpose-conjugate of $A$
- $A^{-1}$: inverse matrix of $A$
- $A^\#$: pseudo-inverse matrix of $A$

### Probability and Expectation

- $\mathbb{P}\{X \in A\}$: probability that $X \in A$
- $\mathbb{E}\{X\}$: expectation value of $X$
- $X_c = X - \mathbb{E}\{X\}$: zero-mean random variable
- $\text{var}\{X\} = \mathbb{E}\{|X_c|^2\}$: variance of $X$
- $\mathbb{E}\{X|Y\}$: conditional expectation of $X$ given $Y$

### Terminology

- ADC: Analog to Digital Converter
- ADPCM: Adaptive Differential PCM
- AR: Autoregressive
- ARMA: AR and MA
- BER: Bit Error Rate
- bps: Bits per second
- cdf: Cumulative distribution function
- CF: Clipping Factor
- CZT: Causal z-Transform
- DAC: Digital to Analog Converter
- DCT: Discrete Cosine Transform
- d.e./de: Difference equation
- DFT: Discrete Fourier Transform
- DTFT: Discrete Time Fourier Transform
- DTMF: Dual Tone Multi-Frequency
- dsp: Digital signal processing/processor
- e.s.d./esd: Energy spectral density
- FIR: Finite Impulse Response
- FFT: Fast Fourier Transform
- FT: Continuous Time Fourier Transform
<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDFT</td>
<td>Inverse Discrete Fourier Transform</td>
</tr>
<tr>
<td>i.i.d./iid</td>
<td>Independent and Identically Distributed</td>
</tr>
<tr>
<td>IIR</td>
<td>Infinite Impulse Response</td>
</tr>
<tr>
<td>ISI</td>
<td>InterSymbol Interference</td>
</tr>
<tr>
<td>LDA</td>
<td>Linear discriminant analysis</td>
</tr>
<tr>
<td>lms</td>
<td>Least mean squares</td>
</tr>
<tr>
<td>MA</td>
<td>Moving Average</td>
</tr>
<tr>
<td>MAC</td>
<td>Multiplication ACcumulation</td>
</tr>
<tr>
<td>OTF</td>
<td>Optical Transfer Function</td>
</tr>
<tr>
<td>PAM</td>
<td>Pulse Amplitude Modulation</td>
</tr>
<tr>
<td>PCA</td>
<td>Principal Component Analysis</td>
</tr>
<tr>
<td>p.d.</td>
<td>Probability Distribution</td>
</tr>
<tr>
<td>ppi</td>
<td>Points per Inch</td>
</tr>
<tr>
<td>p.s.d./PSD</td>
<td>Power Spectral Density</td>
</tr>
<tr>
<td>PSF</td>
<td>Point Spread Function</td>
</tr>
<tr>
<td>PSK</td>
<td>Phase Shift Keying</td>
</tr>
<tr>
<td>QAM</td>
<td>Quadrature Amplitude Modulation</td>
</tr>
<tr>
<td>rls</td>
<td>Recursive least squares</td>
</tr>
<tr>
<td>rms</td>
<td>Root mean square</td>
</tr>
<tr>
<td>r.p./rp</td>
<td>Random process</td>
</tr>
<tr>
<td>SNR</td>
<td>Signal to Noise Ratio</td>
</tr>
<tr>
<td>r.v./rv</td>
<td>Random variable</td>
</tr>
<tr>
<td>STFT</td>
<td>Short Term Fourier Transform</td>
</tr>
<tr>
<td>TF</td>
<td>Transfer Function</td>
</tr>
<tr>
<td>WSS</td>
<td>Wide (Weak) Sense Stationary (Second Order) Process</td>
</tr>
<tr>
<td>ZOH</td>
<td>Zero-Order Hold</td>
</tr>
<tr>
<td>ZT</td>
<td>z-Transform</td>
</tr>
</tbody>
</table>
Chapter 1

Recap on Digital Signal Processing

Signal processing consists of handling data in order to extract information considered relevant, or to modify them so as to give them useful properties: extracting, for example, information on a plane’s speed or distance from a RADAR signal, making an old and decayed sound recording clearer, synthesizing a sentence on an answering machine, transmitting information through a communication channel, etc.

The processing is called digital if it deals with a discrete sequence of values \( \{x_1, x_2, \ldots\} \). There are two types of scenario: either the observation is already a sequence of numbers, as is the case for example for economic data, either the observed phenomenon is “continuous-time”, and the signal’s value \( x(t) \) must then be measured at regular intervals.

This second scenario has tremendous practical applications. This is why an entire section of this chapter is devoted to the operation called sampling.

The acquisition chain is described in Figure 1.1.

![Digital signal acquisition](image-url)
The essential part of the acquisition device is usually the analog-to-digital
converter, or ADC, which samples the value of the input voltage at regular
intervals—every $T_s$ seconds—and provides a coded representation at the output.

To be absolutely correct, this coded value is not exactly equal to the value
of $x(nT_s)$. However, in the course of this chapter, we will assume that $x_s(n) = x(nT_s)$. The sequence of these numerical values will be referred to as the digital
signal, or more plainly as the signal.

$T_s$ is called the sampling period and $F_s = 1/T_s$ the sampling frequency. The
gap between the actual value and the coded value is called quantization noise.

Obviously, the sampling frequency must be high enough “in order not to
lose too much information” — a concept we will discuss later on — from the original
signal, and there is a connection between this frequency and the sampled
signal’s “frequential content”. Anybody who conducts experiments knows this
“graph plotting principle”: when the signal’s value changes quickly (presence
of “high frequencies”), “many” points have to be plotted (it would actually be
preferable to use the phrase “high point density”), whereas when the signal’s
value changes slowly (presence of low frequencies), fewer points need to be
plotted.

To sum up, the signal sampling must be done in such a way that the numerical
sequence $\{x_s(n)\}$ alone is enough to reconstruct the continuous-time
signal. The sampling theorem specifies the conditions that need to be met for
perfect reconstruction to be possible.

1.1 The sampling theorem

Let $x(t)$ be a continuous signal, with $X(F)$ its Fourier transform, which will
also be called the spectrum. The sample sequence measured at the frequency
$F_s = 1/T_s$ is denoted by $x_s(n) = x(nT_s)$.

**Definition 1.1** When $X(F) ≠ 0$ for $F \in (B_1, B_2)$ and $X(F) = 0$ everywhere
else, $x(t)$ is said to be $(B_1, B_2)$ band-limited. If $x(t)$ is real, its Fourier transform
has a property called Hermitian symmetry, meaning that $X(F) = X^*(-F)$, and
the frequency band’s expression is $(-B, +B)$. A common misuse of language
consists of referring to the signal as a $B$-band signal.

**Perfect reconstruction**

Our goal is to reconstruct $x(t)$, at every time $t$, using the sampling sequence
$x_s(n) = x(nT_s)$, while imposing a “reconstruction scheme” defined by the expression (1.1):

$$y(t) = \sum_{n=-\infty}^{+\infty} x(nT_s)h(t - nT_s)$$ (1.1)
where $h(t)$ is called a reconstruction function. Notice that (1.1) is linear with respect to $x(nT_s)$. In order to reach this objective, two questions have to be answered:

1. is there a class of signals $x(t)$ large enough for $y(t)$ to be identical to $x(t)$?
2. if that is the case, what is the expression of $h(t)$?

The answers to these questions are provided by the sampling theorem (1.1).

**Theorem 1.1 (Sampling theorem)**

Let $x(t)$ be a $(B_1, B_2)$ band-limited signal, real or complex, and let $\{x(nT_s)\}$ be its sample sequence, then there are two possible cases:

1. If $F_s = 1/T_s$ is such that $F_s \geq B_2 - B_1$, then $x(t)$ can be perfectly reconstructed from its samples $x(nT_s)$ using the expression:

   $$x(t) = \sum_{n=-\infty}^{+\infty} x(nT_s) h_{(B_1,B_2)}(t - nT_s)$$

   (1.2)

   where the FT of the reconstruction function $h_{(B_1,B_2)}(t)$ is:

   $$H_{(B_1,B_2)}(F) = \frac{1}{F_s} \mathbf{1}(F \in (B_1,B_2))$$

   (1.3)

2. If $F_s = 1/T_s < B_2 - B_1$, perfect reconstruction turns out to be impossible because of the “spectrum aliasing” phenomenon.

The proof uses the Poisson summation formula which gives the relation between $X(F)$ and the values of $x(t)$ at sampling times $nT_s$, and makes it possible to determine the expression of the spectrum of the signal $y(t)$ defined by equation (1.1).
Lemma 1.1 (Poisson formula) Let \( x(t) \) be a signal, and \( X(F) \) its Fourier transform. Then for any \( T_s \):

\[
\frac{1}{T_s} \sum_{k=-\infty}^{+\infty} X(F - kF_s) = \sum_{n=-\infty}^{+\infty} x(nT_s) \exp(-2j\pi nFT_s) \tag{1.4}
\]

where the left member is assumed to be a continuous function of \( F \).

We will use the following definition for the discrete-time Fourier transform. We will see another completely equivalent expression of it (definition 1.3), but more frequently used in the case of numerical sequences.

Definition 1.2 (DTFT) The sum \( \sum_{n=-\infty}^{+\infty} x(nT_s) \exp(-2j\pi nFT_s) \) is called the Discrete-Time Fourier Transform (DTFT) of the sequence \( \{x(nT_s)\} \).

We now go back to the sampling theorem. By using the fact that the Fourier transform of \( h(t - nT_s) \) is \( H(F)e^{-2j\pi nFT_s} \), the Fourier transform of \( y(t) \), defined by (1.1), can be written:

\[
Y(F) = \sum_{n=-\infty}^{+\infty} x(nT_s) \times H(F)e^{-2j\pi nFT_s} = \frac{H(F)}{T_s} \sum_{k=-\infty}^{+\infty} X(F - kF_s) \tag{1.5}
\]

Therefore, if \( F_s \geq B_2 - B_1 \), the different contributions \( X(F - kF_s) \) do not overlap, and by simply assuming \( H_{(B_1, B_2)}(F) = T_s1(F \in (B_1, B_2)) \), \( Y(F) \) coincides exactly with \( X(F) \). Figure 1.3 illustrates this case for a real signal. In this case, \( B_1 = -B \) and \( B_2 = B \).

Figure 1.3 – Real signal reconstruction
Except if specified otherwise, we will assume from now on that $x(t)$ is real. The sufficient reconstruction condition can be written as follows:

$$F_s \geq 2B$$  \hspace{1cm} (1.6)

The limit frequency $2B$ is called the Nyquist frequency. Still in the same case, the Fourier transform of a possible reconstruction function is $H_B(F) = T_s \text{rect}_2B(F)$, and therefore:

$$h_B(t) = \frac{\sin(2\pi Bt)}{\pi F_s t}$$  \hspace{1cm} (1.7)

It should be noted that the function $H_B(F) = T_s \text{rect}_2B(F)$ is not the only possible function. If $F_s$ is assumed to be strictly greater than $2B$, then we can choose a filter with larger transition bands (see Figure 1.3), making it easier to design.

When there is no possible doubt, we will not indicate the dependence on $B$, and simply write $h(t)$ instead of $h_B(t)$.

**Anti-aliasing filter**

The reconstruction formula (1.1), is, according to the Poisson formula (1.4), associated with the periodization of the spectrum $X(F)$ with the period $F_s$. It follows that, for $F_s < 2B$, the different non-zero parts of the spectrum overlap, making perfect reconstruction impossible. The overlapping phenomenon is called spectrum aliasing.

Figure 1.4 illustrates the spectrum aliasing phenomenon for a real signal whose frequential content is of the “low-pass” type, implicitly meaning that it “fills up” the band $(-F_s/2, +F_s/2)$.

Except in some particular cases, we will assume that spectrum signals are of this type, or that they can be modified to fit this description.
For a real signal, showing aliasing means that the frequencies beyond the frequency $F_s/2$ can be “brought back” to the $(-F_s/2, +F_s/2)$ band.

In practice, the following cases will occur:

1. the sampling frequency is imposed: if, knowing how the data is used, the aliasing phenomenon is considered to “cause damage”, the appropriate procedure for sampling a real signal requires the use of low-pass filtering called anti-aliasing filtering which eliminates the components of the frequencies higher than $F_s/2$;

2. the sampling frequency is not imposed: in this case, it can be chosen high enough so that the aliased components of the signal do not alter the expected results. If this is not possible, $F_s$ is set, and the situation becomes the same as in the first case.

Speech signals are a good example. If they are sampled at 8,000 Hz, an extremely common value, high enough to make the person speaking recognizable and understandable, and if no anti-aliasing filtering is done, the reconstructed signal contains a “hissing” noise. This alone justifies the use of an anti-aliasing filter. The irretrievable loss of high frequency components is actually better than the presence of aliasing.

Figure 1.5 illustrates the case of a “low-pass”, prefiltered, real signal to prevent aliasing.

In general, it is important to understand that anti-aliasing filtering must be done in the band that is considered essential (useful band) to the unaliased signal reconstruction. The low-pass filtering mentioned here corresponds to a low-pass sampled signal.

The following general rule can be stated:
The sampling operation of a signal at the frequency $F_s$ must be preceded by an anti-aliasing filtering with a gain equal to 1 and with a width of $F_s$ in the useful band.

**Spectrum aliasing and ambiguity**

For a given signal, for any integer $k$, it is not possible to distinguish $F_0$ from $F_1 = F_0 + kF_s$, $k \in \mathbb{Z}$, which is called the image frequency of $F_0$ relative to $F_s$. Hence, $x_1(t) = \sin(2\pi F_0 t)$ and $x_2(t) = \sin(2\pi(F_0 + kF_s)t)$, with $k \in \mathbb{Z}$ take exactly the same values if both are collected at frequency $F_s$. This is the ambiguity due to the spectrum aliasing phenomenon (or generally speaking to the Poisson formula).

### 1.2 Spectral contents

#### 1.2.1 Discrete-time Fourier transform (DTFT)

The sampling period $T_s$ appears in the DTFT’s expression in definition 1.3.

**Definition 1.3 (DTFT)** The discrete-time Fourier transform of a sequence $\{x(n)\}$ is the function of the real variable $f$, periodic with period 1, defined by:

$$X(f) = \sum_{n=-\infty}^{+\infty} x(n) \exp(-2j\pi nf) \quad (1.8)$$

As you can see, we need only impose $FT_s = f$ and replace $x(nT_s)$ by $x(n)$ to go from (1.4) to (1.8)\(^{(1)}\).

Definition (1.3) calls for a few comments: it can be proven that if $\{x(n)\}$ is summable ($\sum_n |x(n)| < +\infty$), the series (1.8) converges uniformly to a continuous function $X(f)$. However, if $\{x(n)\}$ is square summable ($\sum_n |x(n)|^2 < +\infty$) without having a summable modulus, then the series converges in quadratic mean. There can be no uniform convergence.

Because of its periodicity, the DTFT is plotted on an interval of length 1, most often the intervals $(-1/2, +1/2)$ or $(0, 1)$.

Starting off from $X(f)$, how can we go back to $x(n)$? One possible answer is given in the following result.

**Theorem 1.2 (Inverse DTFT)** If $X(f)$ is a periodic function with period 1, and if $\int_0^1 |X(f)|^2 df < +\infty$, then $X(f) = \sum_n x(n)e^{-2j\pi nf}$, where the $x(n)$ coefficients are given by:

$$x(n) = \int_{-1/2}^{1/2} X(f)e^{2j\pi nf} df \quad (1.9)$$

\(^{(1)}\) $X(F)$, which refers to the FT in (1.4) must not be confused with $X(f)$, the DTFT.
As in the continuous-time case, we have the Parseval formula:

\[
\sum_{n=-\infty}^{+\infty} |x(n)|^2 = \int_{-1/2}^{1/2} |X(f)|^2 df \tag{1.10}
\]

and the conservation of the dot product:

\[
\sum_{n=-\infty}^{+\infty} x(n)y^*(n) = \int_{-1/2}^{1/2} X(f)Y^*(f)df \tag{1.11}
\]

Because the left member of (1.10) is, by definition, the signal’s energy, \(|X(f)|^2\) represents the energy’s distribution along the frequency axis. It is therefore called the energy spectral density (esd), or spectrum. In the literature, this last word is associated with the function \(|X(f)|\). If \(X(f)\) is included, this adds up to three definitions for the same word. But in practice, this is not important, as the context is often enough to clear up any ambiguity. It should be pointed out that the two expressions \(|X(f)|\) and \(|X(f)|^2\) become proportional if the decibel scale is used, by imposing:

\[
S_{dB}(f) = 20 \log_{10} |X(f)| \tag{1.12}
\]

### 1.2.2 Discrete Fourier transform (DFT)

**Definition of the discrete Fourier transform**

A computer calculation of the DTFT, based on the values of the samples \(x(n)\), imposes an infinite workload, because the sequence is made up of an infinity of terms, and because the frequency \(f\) varies continuously on the interval \((0, 1)\). This is why, digitally speaking, the DTFT does not stand a chance against the Discrete Fourier Transform, or DFT. The DFT calculation is limited to a finite number of values of \(n\), and a finite number of values of \(f\).

The digital use of the DFT has acquired an enormous and undisputed practical importance with the discovery of a fast calculation method known as the Fast Fourier Transform, or FFT.

Consider the finite sequence \(\{x(0), \ldots, x(P-1)\}\). Using definition (1.8), its DTFT is expressed \(X(f) = \sum_{n=0}^{P-1} x(n)e^{-2j\pi nf}\) where \(f \in (0, 1)\). In order to obtain the values of \(X(f)\) using a calculator, only a finite number \(N\) of values for \(f\) are taken. The first idea that comes to mind is to take \(N\) values, uniformly spaced-out in \([0, 1]\), meaning that \(f = k/N\) with \(k \in \{0, \ldots, N-1\}\). This gives us the \(N\) values:

\[
X(k/N) = \sum_{n=0}^{P-1} x(n)e^{-2j\pi nk/N} \tag{1.13}
\]
In this expression, $P$ and $N$ play two very different roles: $N$ is the number of points used to calculate the DTFT, and $P$ is the number of observed points of the temporal sequence. $N$ influences the precision of the plotting of $X(f)$, whereas $P$ is related to what is called the frequency resolution.

In practice, $P$ and $N$ are chosen so that $N \geq P$. We then impose:

$$\tilde{x}(n) = \begin{cases} x(n) & \text{for } n \in \{0, \ldots, P-1\} \\ 0 & \text{for } n \in \{P, \ldots, N-1\} \end{cases}$$

Obviously:

$$X(k/N) = \sum_{n=0}^{P-1} x(n) e^{-2\pi j nk/N} = \sum_{n=0}^{N-1} \tilde{x}(n) e^{-2\pi j nk/N}$$

Because the sequence $x(n)$ is completed with $(N - P)$ zeros, an operation called zero-padding, in the end we have as many points for the sequence $\tilde{x}(n)$ as we do for $X(k/N)$. Choosing to take as many points for both the temporal sequence and the frequential sequence does not restrict in any way the concepts we are trying to explain. This leads to the definition of the discrete Fourier transform.

**Definition 1.4** Let \(\{x(n)\}\) be a $N$-length sequence. Its discrete Fourier transform or DFT is defined by:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}, \quad k \in (0, 1, \ldots, N - 1)$$  \hspace{1cm} (1.14)

where $W_N = e^{-2\pi j / N}$  \hspace{1cm} (1.15)

is an $N$-th root of unity, that is to say such that $W_N^N = 1$. The inverse formula, leading from the sequence \(\{X(k)\}\) to the sequence \(\{x(n)\}\), is:

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W_N^{-nk}$$  \hspace{1cm} (1.16)

**Properties of the DFT**

The properties of the DFT show strong similarities with those of the DTFT. However, there is an essential difference. In the formulas associated with the DFT, all the index calculations are done modulo $N$.

**Fast Fourier transform**

The fast Fourier transform, or FFT, first published in 1965 by J. W. Cooley and J. W. Tuckey [8], is a fast DFT calculation technique. The basic algorithm,
many versions of which can be found, calculates a number of points \( N \), equal to a power of 2, and the time saved compared with a direct calculation is roughly:

\[
\text{gain} = \frac{N}{\log_2(N)}
\]

To get a better idea, if \( N = 1,024 \), the FFT is about 100 times faster than the direct calculation based on the definition of the DFT.

### 1.3 Case of random signals

In the case of a random process, also referred to as a “time series”, the notion of “spectral content” needs to be treated with caution. More specifically, the Fourier transform of a single random process trajectory generally does not exist. However, theoretical developments [3] lead us to define the spectrum associated with a random process by the DTFT of the covariance function:

\[
S_{XX}(f) = \text{DTFT} (R_{XX}(n)) = \sum_{n=-\infty}^{+\infty} R_{XX}(n)e^{-2\pi jnf}
\] (1.17)

\( S_{XX}(f) \) is called the power spectral density (psd).

A fairly simple alternative idea is to use the Fourier transform of that single record of length \( N \). This leads us to the definition of a periodogram. A periodogram is the random function of \( f \in (0,1) \) defined by:

\[
I_N(f) = \frac{1}{N} \left| \sum_{n=0}^{N-1} X(n)e^{-2\pi jnf} \right|^2
\] (1.18)

Unfortunately the periodogram is not a good estimator of the spectral density [3]. However a consistent estimate can be derived by averaging or smoothing periodograms.

The filtering equations also exhibit a fundamental difference in comparison to the deterministic case \( (Y(f) = H(f)X(f)) \). In the random case, the psds are linked by the relation:

\[
S_{YY}(f) = |H(f)|^2 S_{XX}(f)
\] (1.19)

Note that there is a second filtering equation which can be used:

\[
S_{YX}(f) = H(f)S_{XX}(f)
\] (1.20)

where \( S_{YX}(f) \), which is called the interspectrum, refers to the Fourier transform of \( \{R_{YX}(n)\} \).
1.4 Example of the Dual Tone Multi-Frequency (DTMF)

On a Dual Tone Multi-Frequency (DTMF) phone keyboard, each key is associated with the sending of a signal. This signal is the sum of two sines the frequencies (in Hz) of which are given in the correspondence Table 1.1.

<table>
<thead>
<tr>
<th>Hz</th>
<th>1,209</th>
<th>1,336</th>
<th>1,477</th>
</tr>
</thead>
<tbody>
<tr>
<td>697</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>770</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>852</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>941</td>
<td>*</td>
<td>0</td>
<td>#</td>
</tr>
</tbody>
</table>

**Table 1.1 – Frequency correspondence table**

This means that when you dial “5” on your phone, the signal \( x(t) = \cos(2\pi \times 1,336 \times t) + \cos(2\pi \times 770 \times t) \) is sent through the phone line.

**Comments:**

- These frequencies belong to the (300 Hz - 3,400 Hz) band, the phone band for the switched network (fixed phones). The frequencies associated with the columns are all greater than those associated with the lines. This layout can help to find the phone number using the signal. Finally, the frequencies were chosen so as not to have frequency ratios equal to integers.

As we have seen, a non-linear operation can cause multiples (harmonics) of the fundamental frequency to appear, causing some confusion;

- the keyboard is designed to always send signals for periods longer than \( \tau_1 = 65 \) milliseconds. This value was chosen so that the two frequencies contained in the signal could easily be separated. In the worst case, the difference in frequency is \( \Delta F_{\text{min}} = 1,209 - 941 = 268 \) Hz (corresponding to the * key), therefore \( \tau_1 \) needs to be such that \( \Delta F_{\text{min}} \tau_1 \gg 1 \). With the values that were chosen, \( \Delta F_{\text{min}} \tau_1 > 17 \);

- finally, it must be possible to tell the difference between the number \( C \) being sent for a duration of \( T \) and the number \( CC \) being sent for the same duration. This is why, after each key is released, a zero signal is sent for at least 80 ms (even if you can dial faster than that!).

**Exercise 1.1 (DTMF signal processing) (see p. 187)**
We are going to try to find a 10 digit phone number using the signal sent
by the phone. We will start by sampling the signal at a frequency of 8,000 samples per second, a speed much higher than twice the highest frequency, that is $2 \times 1,477 = 2,954$ Hz.

The following program creates such a signal:

```matlab
%===== genekey.m
clear
Fs=8000; % sampling freq.
tel='0145817178'; lt=length(tel); % seq. of numbers
%%% coding table
keys=['123456789*0#']; nbkeys=length(keys);
FreqB=[697 770 852 941]; FreqH=[1209 1336 1477];
Freqskeys=...
[FreqB(1) FreqH(1); FreqB(1) FreqH(2); % 1 et 2
 FreqB(1) FreqH(3); FreqB(2) FreqH(1); % 3 et 4
 FreqB(2) FreqH(2); FreqB(2) FreqH(3); % 5 et 6
 FreqB(3) FreqH(1); FreqB(3) FreqH(2); % 7 et 8
 FreqB(3) FreqH(3); FreqB(4) FreqH(1); % 9 et *
 FreqB(4) FreqH(2); FreqB(4) FreqH(3)];% 0 et #
%%% constraints
firstname=0.065; tsil=0.080; % in seconds
%%% construction of the seq. of frequencies
FREQS=zeros(lt,2);
for k=1:lt
 ind=find(keys==tel(k)); % test of the number
 FREQS(k,:)=Freqskeys(ind,:); % associated Freq.
end
freqs=FREQS/Fs; % normalized freq.
%%% construction of the signal
y=zeros(100+fix(100*rand),1); % starting with signal=0
dton=fix(1000*rand(1t,1)+tton*Fs); % number duration
dsail=fix(1000*rand(1t,1)+tsil*Fs); % silence duration
for k=1:lt
 sigu=cos(2*pi*(0:dton(k))'*freqs(k,:))*ones(2,1);
y=[y;sigu;zeros(dsail(k),1)];
end
%%% some noise is added
lx=length(y); py=y'*y/lx;
SNR=30; % signal to noise ratio
pb=py*10^(-SNR/10); x=y+sqrt(pb)*randn(lx,1);
%%% plotting of the signal
tps=(0:lx-1)/Fs; plot(tps,x); grid
set(gca,'xlim',[0 (lx-1)/Fs])
```

In order to simulate the perturbations on an actual phone call, the program adds noise created by $\sqrt{pb} \times \text{randn}(L,1)$. SNR is the signal-to-noise ratio (in dB) chosen equal to 30 dB. The resulting signal is shown in Figure 1.6.

We are going to find the 10 digit number in this signal in two steps. First, we will determine the beginning and the end of the signal’s active zones, then
we will analyze each of the intervals to extract the frequencies and therefore the corresponding digit. To determine the beginning and the end of the active zones of the signal, we are going to estimate the “instantaneous power” and compare it to a threshold value. We will see later on as we study random phenomena what we mean exactly by “estimating the instantaneous power”. Here, we will merely be considering the quantity:

$$P_n = \frac{1}{N} \sum_{k=n-N+1}^{n} x_k^2$$

which gives a relevant indication on the signal’s fluctuations. The choice of $N$ is done as a compromise. Consider, for example, the signal $x(n)$ represented in Figure 1.6. If $N$ is very small, $P_n$ will be very close to the amplitude $x_n^2$. The risk would be to make the conclusion that the power is equal to zero whenever the amplitude is close to 0 (which happens every period). If, on the contrary, $N$ is very high, we might include a silence and miss the beginning or the end of an active part. Quantitatively, $N$ must therefore be much greater than the longest of the periods of the active parts, and much smaller than the duration of the wanted signal, that is to say 65 ms. This can be expressed:

$$\frac{F_s}{697} \ll N \ll 65 \times 10^{-3} F_s$$

For $F_s = 8,000$, and with $N = 100$, this double inequality is satisfied.

1. write a program that measures the “instantaneous power” and determines the beginning and the end of the signals associated with a digit;

2. write a program that determines the digit associated with each portion of the signal.
Chapter 2

Additional Information About Filtering

Mathematically speaking, using the filter with the transfer function $H(z)$ for filtering the sequence $x(n)$ leads to a perfectly determined result. However, depending on the practical implementation of the filter, the results can vary in terms of precision, speed, etc. This chapter deals with the technical aspects of filtering. If you restrict yourself to a “simulation” approach, as we have up until now, the filter function is everything you will ever need. However, if this filtering operation has to be implemented, its effectiveness requires some additional knowledge that will be detailed in this chapter.

2.1 Filter implementation

Linear invariant filters as we consider in this section perform the convolution of the input sequence $\{x(n)\}$ with a sequence $\{h(n)\}$ called the impulse response of the filter:

$$y(n) = (x \ast h)(n) = \sum_{k=-\infty}^{+\infty} x(k)h(n-k) = \sum_{k=-\infty}^{+\infty} x(n-k)h(k)$$ (2.1)

When the series $\{h(n)\}$ is of finite length, we speak of a finite impulse response filter or FIR filter; otherwise, we have an infinite impulse response filter or IIR filter.

The DTFT of $\{h(n)\}$ is called the complex gain of the filter, and its modulus is the gain.

The ZT of $\{h(n)\}$ is called the transfer function.
2.1.1 Examples of filter structures

In this section, we will study the implementation of the filtering function, in other words its programming. Figure 2.1 shows a particular implementation called the canonical direct form of a general recursive filter with the transfer function:

\[ H(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_p z^{-p}}{1 + a_1 z^{-1} + \cdots + a_p z^{-p}} \]

Choosing the same degree for both the numerator and the denominator does not restrict us in any way; you need only consider that some of the coefficients can be equal to zero.

This “implementation” first performs the calculation of:

\[ t(n) = i(n) - a_1 t(n - 1) - \cdots - a_p t(n - p) \]

then the calculation of:

\[ o(n) = b_0 t(n) + b_1 t(n - 1) + \cdots + b_p t(n - p) \]

where \( \{i(n)\} \) and \( \{o(n)\} \) are the input and output sequences respectively.

For this algorithm, the vector \( \mathbf{x}(n) \triangleq [x_0(n) \ x_1(n) \ \ldots \ x_{p-1}(n)]^T \) (\( p \times 1 \)) is called the filter state:

\[ \mathbf{x}(n) = [t(n) \ t(n - 1) \ \ldots \ t(n - p + 1)]^T \]

Its components, referred to as state variables, are the input values of the “delay” cells denoted \( z^{-1} \) in Figure 2.1. By introducing the vectors:

\[ \mathbf{a} = [a_1 \ a_2 \ \ldots \ a_p]^T \text{ and } \mathbf{b} = [b_1 \ b_2 \ \ldots \ b_p]^T \]
we get the following expression for the algorithm:

\[
\begin{align*}
t(n) &= i(n) - a^T x(n - 1) \\
o(n) &= b_0 t(n) + b^T x(n - 1) \\
x(n) &= \begin{bmatrix} t(n) & x_0(n-1) & x_1(n-1) & \cdots & x_{p-2}(n-1) \end{bmatrix}
\end{align*}
\]  

(2.2)

The following `filtrer` function implements this algorithm:

```matlab
function [ys, xs] = filtrer(num, den, xe, xi)
%! Filter (direct canonical structure)!
%! SYNOPSIS: [ys, xs] = FILTRER(num, den, xe, xi)!
%! num = [b0 b1 ... bp]
%! den = [1 a1 a2 ... aP]
%! xe = input sequence
%! xi = initial state
%! ys = output sequence
%! xs = final state
%!==========================================!
lden = length(den); lnum = length(num);
if (lden < lnum), den(lnum) = 0; lden = lnum; end
if (lnum < lden), num(lnum) = 0; end
ld = lden - 1; N = length(xe);
av = zeros(1, ld); bv = av;
av(:) = den(2:lden); bv(:) = num(2:lden);
if (nargin == 3), zzi = zeros(ld, 1); end
if (nargin == 4),
    if length(xi) < ld, xi(ld) = 0; end
    zzi = zeros(ld, 1); zzi(:) = xi;
end
b0 = num(1); xs = zzi; ys = zeros(ld, 1);
for k = 1:N,
    x0n = xe(k) - av * xs;
    ys(k) = b0 * x0n + bv * xs;
    xs = [x0n; xs(1:ld-1)]; % new state
end
```

Determining the initial state leading to a given input-output sequence is another problem altogether. Using the recursive equations (2.2) that lead to \( t(n) \) and \( o(n) \), we can also write:

\[
\begin{bmatrix}
o(p - 1) \\
i(p - 1) \\
\vdots \\
o(0) \\
i(0)
\end{bmatrix} =
\begin{bmatrix}
b_0 & b_1 & \cdots & b_p & 0 & \cdots & 0 \\
1 & a_1 & \cdots & a_p & 0 & \cdots & 0 \\
0 & b_0 & \cdots & b_{p-1} & b_p & 0 & \cdots & 0 \\
0 & 1 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
o(0) \\
i(0)
\end{bmatrix}
\begin{bmatrix}
t(p - 1) \\
t(0) \\
\vdots \\
t(p - 2)
\end{bmatrix} = OT
\]
This expression shows that the initial state $T = [t(-1) \ldots t(-p)]^T$ can be reconstructed so long as the matrix $O$ is invertible. The system theory demonstrates that the possibility of reconstruction is related to the concept of observability. A great number of observability criteria exist, based on the state representations associated with a system [22].

The filtric function detailed hereafter carries out the reconstruction of the state associated with the processing architecture implemented by filter:

```matlab
function zi=filtric(num,den,xi,yo)
%!---------------------------------------------------------!
%! Initial state reconstruction for a direct !
%! canonical structure !
%! SYNOPSIS: zi=FILTRIC(num,den,xi,yo) !
%! num = [b0 b1 ... bP] !
%! den = [1 a1 a2 ... aP] !
%! xi = input sequence !
%! yo = output sequence !
%! zi = reconstructed initial state !
%!---------------------------------------------------------!
lden=length(den); lnum=length(num);
if (lden<lnum), den(lnum)=0; lden=lnum; end
if (lnum<lden), num(lden)=0; end
ld=lden-1;
numv=zeros(lden,1); denv=numv;
numv(:)=num; denv(:)=den;
lx=length(xi); ly=length(yo);
if lx<ld, xi(ld)=0; end
if ly<ld, yo(ld)=0; end
ysv=zeros(1,ld); xev=zeros(1,ld);
ysv(:)=yo(ld:-1:1); xev(:)=xi(ld:-1:1);
x=[ysv;xev]; vec=zeros(2*ld,1); vec(:)=x;
v0=[numv; zeros(ld-1,1); denv; zeros(ld,1)];
A=[]; for ii=1:ld, A=[A v0]; end
A=A(1:4*ld*ld);
Ax=zeros(2*ld,2*ld); Ax(:,)=A; Ax=Ax';
zi=inv(Ax) * vec; zi=zi(ld+1:2*ld);
```

The state reconstruction function is inseparably related to the filtering function, which is itself based on a particular processing architecture.

The reconstruction function is rarely used. The state vector $x_s$, final state of the first block’s processing, is transmitted as the initial state of the second block. The result $y_p$ is identical to the one obtained for the filtering of the entire block $e_t$.

```matlab
%===== fil2blocks.m
inp1=randn(100,1); inp2=randn(100,1); etot=[inp1;inp2];
b=[1 .3]; a=[1 -.8 .9];
```
MATLAB®’s filtering function, \texttt{filter}, uses the \textit{Transpose-Form IIR} structure [28], different from the previous one, represented in Figure 2.2. As in our example, \texttt{filter} transmits the state vector. The reconstruction function \texttt{filtic.m} is available as part of the \textit{Signal Toolbox}. Exercise 2.1 is a study of this structure.

\textbf{Exercise 2.1 (Filter architecture)} (see p. 189)

Consider the Transpose-Form IIR structure (Figure 2.2) of a rational filter.

1. Determine the filter’s transfer function.

---

```
%===== global filtering (null initial state)
y=filtrer(b,a,etot);
%===== filtering the 2 blocks
[y1 xs]=filtrer(b,a,inp1);  % null initial state
y2=filtrer(b,a,inp2,xs);  % initial state xs
yp=[y1;y2];  
%===== drawing for the transition between 2 blocks
[y(90:110) yp(90:110)]
```
By defining the state \( x = [x_1(n) \ldots x_p(n)] \) at the time \( n \), determine the state representation and express it as follows:

\[
\begin{align*}
x(n) &= Ax(n-1) + b_i(n) \\
o(n) &= c^T x(n-1) + d_i(n)
\end{align*}
\]

Use this to find the filtering program. It might be useful to notice that the matrix \( A \) is the transpose of the companion matrix (\texttt{companion} function) associated with the denominator polynomial \([1 \ a_1 \ a_2 \ldots \ a_p]\).

Find the associated reconstruction function using only the filtering function. In order to do this, express \( x_k(0) \) as the sum of an input filtering and an output filtering.

### 2.1.2 Distributing the calculation load in an FIR filter

We wish to distribute the calculation load for an FIR filtering algorithm among several processors. Only two methods will be presented. The first one consists of distributing the number of multiplication/accumulation operations (MAC operations) among \( M \) branches without changing the processing rate. The second one consists of organizing the calculation in different units, so as to reduce this speed, at the cost of a certain delay.

#### Paralleled calculations

Consider the filtering equation \( y(n) = \sum_{k=-\infty}^{+\infty} h(k)x(n - k) \). For a given \( M \), we define \( k = mM + r \) where \( r \in \{0, \ldots, M - 1\} \). We obtain:

\[
y(n) = \sum_{r=0}^{M-1} \sum_{m=-\infty}^{+\infty} h(mM + r)x(n - mM - r)
\]

This expression shows \( y(n) \) as the sum of \( M \) terms:

\[
\begin{align*}
r = 0 & \quad \sum_{m=-\infty}^{+\infty} h(mM)x(n - mM) \\
r = 1 & \quad \sum_{m=-\infty}^{+\infty} h(mM + 1)x(n - mM - 1) \\
& \quad \vdots \\
r = M - 1 & \quad \sum_{m=-\infty}^{+\infty} h(mM + M - 1)x(n - mM - M + 1)
\end{align*}
\]

The first term is the filtering of a sequence \( \ldots, x(n-M), x(n), x(n+M), \ldots \) by the filter with the impulse response \( h(0), h(M), \ldots \). The next terms correspond to translated sequences filtered by the filters \( h_r(m) = \{h(r), \ldots, \} \).
Additional Information About Filtering

$h(r + mM), \ldots \}$. The filter $h_r(m)$ is called the $r$-th $M$-polyphase component of $h(n)$.

Figure 2.3 illustrates this processing method.

![Figure 2.3 – A representation of the paralleled process](image)

Exercise 2.2 (Parallel implementation of the FIR filtering) (see p. 192)
Write a program designed to simulate the process described by Figure 2.3. Choose $M = 4$ and a low-pass, $(-0.3, +0.3)$ band FIR filter with 25 coefficients. The result will be compared to the one obtained through direct filtering.

This method for paralleling does not reduce the processing speed in intermediate filters. Only the number of multiplications per filter is reduced.

2.1.3 FIR block filtering

Let us again consider the FIR filtering equation $y(k) = h(0)x(k) + h(1)x(k - 1) + \cdots + h(P)x(k - P)$. Let:

$$y = \begin{bmatrix} y(nN) \\ y(nN - 1) \\ \vdots \\ y(nN - (N - 1)) \end{bmatrix} = X \times \begin{bmatrix} h(0) \\ h(1) \\ \vdots \\ h(P) \end{bmatrix}$$

with:

$$X = \begin{bmatrix} x(nN) & x(nN - 1) & \cdots & x(nN - P) \\ x(nN - 1) & x(nN - 2) & \cdots & x(nN - 1 - P) \\ \vdots \\ x(nN - (N - 1)) & x(nN - N) & \cdots & x(nN - (N - 1) - P) \end{bmatrix}$$
By organizing the inputs modulo $M$, we obtain:

$$
y = \begin{bmatrix}
    [x(nN) & x(nN - M) & \cdots ] \\
    [x(nN - 1) & x(nN - 1 - M) & \cdots ] \\
    \vdots \\
    [x(nN - (N - 1)) & x(nN - (N - 1) - M) & \cdots ] \\
\end{bmatrix} \times \begin{bmatrix}
    h(0) \\
    h(M) \\
    \vdots \\
    h(1) \\
    h(M + 1) \\
    \vdots \\
\end{bmatrix}$$. 

By restricting ourselves to the case $M = N = 2$, the previous expression can be written:

$$
\begin{bmatrix}
    y(2n) \\
    y(2n - 1)
\end{bmatrix} = \begin{bmatrix}
    [x(2n) & x(2n - 2) & \cdots ] \\
    [x(2n - 1) & x(2n - 3) & \cdots ] \\
\end{bmatrix} \times \begin{bmatrix}
    h(0) \\
    h(2) \\
    \vdots \\
    h(1) \\
    h(3) \\
    \vdots \\
\end{bmatrix} 
$$

If we assume $x_0(n) = [x(2n), x(2n-2), \ldots]^T$ and $x_1(n) = [x(2n-1), x(2n-3), \ldots]^T$, we can also write:

$$
\begin{bmatrix}
    y(2n) \\
    y(2n - 1)
\end{bmatrix} = \begin{bmatrix}
    x_0(n) \\
    x_1(n)
\end{bmatrix} \times \begin{bmatrix}
    x_0(n-1) \\
    \end{bmatrix} \times \begin{bmatrix}
    h_0 \\
    h_1
\end{bmatrix}
$$

where $h_0 = [h(0), h(2), \ldots]^T$ and $h_1 = [h(1), h(3), \ldots]^T$. If we develop $y(2n)$ and $y(2n - 1)$, we obtain:

$$
y(2n) = x_0(n)h_0 + x_1(n)h_1 = x_1(n)(h_0 + h_1) + (x_0(n) - x_1(n))h_0$$

$$
y(2n - 1) = x_1(n)h_0 + x_0(n - 1)h_1 = x_1(n)(h_0 + h_1) + (x_0(n - 1) - x_1(n))h_1$$

Therefore, the calculation of the two terms $y(2n)$ and $y(2n - 1)$ requires the calculation of a total of four terms. However, one of them, $x_1(n)(h_0 + h_1)$,
appears twice. If $P$ refers to the length of the filter $h$, the lengths of $h_0$ and $h_1$ are at the most equal to $P/2$. Hence the three terms of the calculation of $y(2n)$ and $y(2n-1)$ correspond to $P/2$ length filtering. Figure 2.4 illustrates all these calculations.

To sum up, in order to calculate $y(2n)$ and $y(2n-1)$, the number of MAC operations is roughly $3 \times P/2$. This value should be compared to the $2 \times P$ MAC operations of the direct calculations. You may, as an exercise, simulate the process described in Figure 2.4. There is more than one method organizing the process. Consider for example:

\[
\begin{align*}
y(2n) &= x_0(n)h_0 + x_1(n)h_1 \\
y(2n-1) &= (x_0(n) + x_1(n))(h_0 + h_1) - x_0(n)h_1 - x_1(n)h_0
\end{align*}
\]

Notice that $x_0(n-1)h_1$ was calculated previously. Hence there are indeed only three MAC operations at this stage of the calculation. We can also consider parallel block processing for values of $M$ and $N$ different from 2.

### 2.1.4 FFT filtering

A possibility for accelerating filtering operations is to work in the frequency domain, using Fourier transforms to take advantage of the FFT algorithm’s speed. Unfortunately, this is not as simple as it seems, because linear filtering uses a linear convolution:

\[
y(n) = \sum_{m=-\infty}^{+\infty} x(m)h(n-m)
\]

the DTFT of which is $H(f)X(f)$, whereas the product of the DFTs is the DFT of the circular convolution. As a reminder, here is its expression (2.4):

\[
\sum_{m=0}^{N-1} x(m)h((n-m) \mod N)
\]
A simple calculation shows that expressions (2.3) and (2.4) lead to completely different results. Consider a finite impulse response filter \( \{h_N(n)\} \) and a sequence \( \{x(n)\} \). The output value at the time \( n \) is:

\[
y(n) = h_N(0)x(n) + \cdots + h_N(N-1)x(n-N+1) \tag{2.5}
\]

For a clearer picture, let us assume \( N = 8 \). We are going to calculate the terms resulting from a circular convolution of the length 8 block \( \{x(n), \ldots, x(n-7)\} \) with a filter with the coefficients \( \{h(0), \ldots, h(7)\} \). The following table describes the operations modulo 8.

<table>
<thead>
<tr>
<th>( m )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>( h_m )</td>
<td>( h_0 )</td>
<td>( h_1 )</td>
<td>( h_2 )</td>
<td>( h_3 )</td>
<td>( h_4 )</td>
<td>( h_5 )</td>
<td>( h_6 )</td>
<td>( h_7 )</td>
</tr>
</tbody>
</table>

\[
x = x_{n-7} \quad x_{n-6} \quad x_{n-5} \quad x_{n-4} \quad x_{n-3} \quad x_{n-2} \quad x_{n-1} \quad x_n
\]

\[
h_{-m \mod 8} = \begin{bmatrix} h_0 & h_7 & h_6 & h_5 & h_4 & h_3 & h_2 & h_1 \end{bmatrix}
\]

\[
h_{1-m \mod 8} = \begin{bmatrix} h_1 & h_0 & h_7 & h_6 & h_5 & h_4 & h_3 & h_2 \end{bmatrix}
\]

\[
h_{7-m \mod 8} = \begin{bmatrix} h_7 & h_6 & h_5 & h_4 & h_3 & h_2 & h_1 & h_0 \end{bmatrix}
\]

Notice that among the 8 results of the circular convolution, only the last one, \( h_0x_n + h_1x_{n-1} + \cdots + h_7x_{n-7} \), corresponds to one of the terms from the linear convolution, making this approach completely hopeless. This is actually downright wrong, as we are going to see now.

**The overlap-save algorithm**

Consider an \( N = 5 \) length filter with its coefficients \( h(0), \ldots, h(4) \) completed by three zeros. Let \( \{x(n), \ldots, x(n-7)\} \) be the \( L = 8 \) length input block. As we did before, we can build the sequence of the 8 output values by using the following table:

<table>
<thead>
<tr>
<th>( m )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>( h_m )</td>
<td>( h_0 )</td>
<td>( h_1 )</td>
<td>( h_2 )</td>
<td>( h_3 )</td>
<td>( h_4 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
x = x_{n-7} \quad x_{n-6} \quad x_{n-5} \quad x_{n-4} \quad x_{n-3} \quad x_{n-2} \quad x_{n-1} \quad x_n
\]

\[
h_{-m \mod 8} = \begin{bmatrix} h_0 & 0 & 0 & 0 & h_4 & h_3 & h_2 & h_1 \end{bmatrix}
\]

\[
h_{1-m \mod 8} = \begin{bmatrix} h_1 & h_0 & 0 & 0 & 0 & h_4 & h_3 & h_2 \end{bmatrix}
\]

\[
h_{7-m \mod 8} = \begin{bmatrix} 0 & 0 & 0 & h_4 & h_3 & h_2 & h_1 & h_0 \end{bmatrix}
\]
The resulting circular convolution outputs are:

\[ y_c(0) = h_0 x_{n-7} + h_4 x_{n-3} + h_3 x_{n-2} + h_2 x_{n-1} + h_1 x_n \]
\[ y_c(1) = h_1 x_{n-7} + h_0 x_{n-6} + h_4 x_{n-2} + h_3 x_{n-1} + h_2 x_n \]
\[ y_c(2) = h_2 x_{n-7} + h_1 x_{n-6} + h_0 x_{n-5} + h_4 x_{n-1} + h_3 x_n \]
\[ y_c(3) = h_3 x_{n-7} + h_2 x_{n-6} + h_1 x_{n-5} + h_0 x_{n-4} + h_4 x_n \]
\[ y_c(4) = h_4 x_{n-7} + h_3 x_{n-6} + h_2 x_{n-5} + h_1 x_{n-4} + h_0 x_{n-3} \]
\[ y_c(5) = h_4 x_{n-6} + h_3 x_{n-5} + h_2 x_{n-4} + h_1 x_{n-3} + h_0 x_{n-2} \]
\[ y_c(6) = h_4 x_{n-5} + h_3 x_{n-4} + h_2 x_{n-3} + h_1 x_{n-2} + h_0 x_{n-1} \]
\[ y_c(7) = h_4 x_{n-4} + h_3 x_{n-3} + h_2 x_{n-2} + h_1 x_{n-1} + h_0 x_n \]

Notice that the four last expressions correspond to terms found with the linear convolution:

\[ y_c(4) = y(n - 3) \]
\[ y_c(5) = y(n - 2) \]
\[ y_c(6) = y(n - 1) \]
\[ y_c(7) = y(n) \]

In order to calculate the next four values, we have to choose \( \{ x(n + 4), \ldots, x(n - 3) \} \) as our input block. This block partly overlaps the previous one (see Figure 2.5), hence the word overlap in the name “overlap-save algorithm”.

All of the operations can be summed up as follows:
Overlap-save

1. calculation (performed only once) of the DFT of the $L$ length sequence $h(n)$ completed by $(L - N)$ zeros. $L$ (the DFT’s length) is usually a power of 2;

2. calculation of the DFT of an $L$ length block extracted from the input data with an overlap of the $(N - 1)$ last values of the previous block;

3. term-by-term multiplication of the two DFTs, followed by an IDFT;

4. the $(L - N + 1)$ terms corresponding to the linear convolution (2.3) are saved.

Let us compare the number of operations for the overlap-save algorithm with that of a direct calculation. The direct calculation of one convolution point for an $N$ length impulse response requires a loop comprising $N$ MAC operations.

Using the overlap-save algorithm, the impulse response’s DFT is calculated in advance. There are two $L$ length FFTs left for each step (one direct, one inverse) and $L$ complex multiplications, in all a calculation load of roughly $2 \times L \log_2(L) + L = 2L \log_2(L\sqrt{2})$ MAC operations. This calculation provides us with a block of $(L - N + 1)$ convolution points, equivalent to a load of about $2L \log_2(L)/(L - N + 1)$ MAC operations per calculation point. Hence the gain is roughly:

$$G(N, L) = \frac{(L - N + 1)N}{2L \log_2(L\sqrt{2})}$$

Thus, for $N = L/2$ and $N \geq 32$, the FFT technique is quicker.

Other parameters have to be considered. The FFT calculation implies the use of array pointers, which cause a considerable increase in the calculation time. The FFT also requires memory space to save the data arrays that are too large for the filter’s memory. This is why convolution calculations that use the FFT are usually undertaken only with filters with a length of more than a hundred coefficients. In acoustics, impulse response of a quarter-second sampled at 8,000 Hz lead to lengths of 2,000 samples. You also have to add to that the delay caused by block processing, a delay roughly equal the block’s length. For some applications, this delay is reason enough to discard these techniques.
Overlap-add algorithm

Consider once again the previous example of a filter \(\{h(0), \ldots, h(4)\}\). We still hope to obtain the output:

\[
y(n) = h_0 x_n + h_1 x_{n-1} + h_2 x_{n-2} + h_3 x_{n-3} + h_4 x_{n-4}
\]

Consider the convolutions concerning two consecutive length 8 blocks labelled \(x_p\) and \(x_{p+1}\).

<table>
<thead>
<tr>
<th>(m)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>(h_m)</td>
<td>(h_0)</td>
<td>(h_1)</td>
<td>(h_2)</td>
<td>(h_3)</td>
<td>(h_4)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
x_p = x_{n-3} \quad x_{n-2} \quad x_{n-1} \quad x_n\]

\[
x_{p+1} = x_{n+1} \quad x_{n+2} \quad x_{n+3} \quad x_{n+4}\]

\[
h_{-m \mod 8} = h_0 \quad 0 \quad 0 \quad 0 \quad h_4 \quad h_3 \quad h_2 \quad h_1\]

\[
h_{1-m \mod 8} = h_1 \quad h_0 \quad 0 \quad 0 \quad 0 \quad h_4 \quad h_3 \quad h_2\]

\[
h_{7-m \mod 8} = 0 \quad 0 \quad 0 \quad h_4 \quad h_3 \quad h_2 \quad h_1 \quad h_0\]

The values obtained from the first block are:

\[
y_{c,p}(0) = h_0 x_{n-3}
\]
\[
y_{c,p}(1) = h_1 x_{n-3} + h_0 x_{n-2}
\]
\[
y_{c,p}(2) = h_2 x_{n-3} + h_1 x_{n-2} + h_0 x_{n-1}
\]
\[
y_{c,p}(3) = h_3 x_{n-3} + h_2 x_{n-2} + h_1 x_{n-1} + h_0 x_n
\]
\[
y_{c,p}(4) = h_4 x_{n-3} + h_3 x_{n-2} + h_2 x_{n-1} + h_1 x_n
\]
\[
y_{c,p}(5) = h_4 x_{n-2} + h_3 x_{n-1} + h_2 x_n
\]
\[
y_{c,p}(6) = h_4 x_{n-1} + h_3 x_n
\]
\[
y_{c,p}(7) = h_4 x_n
\]

and the values obtained from the next block are:

\[
y_{c,p+1}(0) = h_0 x_{n+1}
\]
\[
y_{c,p+1}(1) = h_1 x_{n+1} + h_0 x_{n+2}
\]
\[
y_{c,p+1}(2) = h_2 x_{n+1} + h_1 x_{n+2} + h_0 x_{n+3}
\]
\[
y_{c,p+1}(3) = h_3 x_{n+1} + h_2 x_{n+2} + h_1 x_{n+3} + h_0 x_{n+4}
\]
\[
y_{c,p+1}(4) = h_4 x_{n+1} + h_3 x_{n+2} + h_2 x_{n+3} + h_1 x_{n+4}
\]
\[
y_{c,p+1}(5) = h_4 x_{n+2} + h_3 x_{n+3} + h_2 x_{n+4}
\]
\[
y_{c,p+1}(6) = h_4 x_{n+3} + h_3 x_{n+4}
\]
\[
y_{c,p+1}(7) = h_4 x_{n+4}
\]
As you can see:

\[
\begin{align*}
    y(n + 1) &= y_{c,p}(4) + y_{c,p+1}(0), \\
    y(n + 2) &= y_{c,p}(5) + y_{c,p+1}(1), \\
    y(n + 3) &= y_{c,p}(6) + y_{c,p+1}(2), \\
    y(n + 4) &= y_{c,p}(7) + y_{c,p+1}(3).
\end{align*}
\]

The conclusion is that if the input block is completed with \(N - 1 = 4\) zeros, the circular convolution will calculate incomplete sums. These sums will then be completed with values obtained from the next block translated by \(N - 1\) values. The sequence of operations can be summed up in the following way:

**Overlap-add**

1. calculation of the DFT of the \(N\) length sequence \(h(n)\) completed with \((L - N)\) zeros. Usually the length \(L\) of the DFT is \(2^P\);

2. calculation of the DFT of a length \((L - N + 1)\) block extracted from the input data without any overlap and completed with \((N - 1)\) zeros;

3. term-by-term multiplication of the two DFTs, followed by an IDFT;

4. sum of the current block and of the next block with an overlap of \((N - 1)\) values.

You can check that the overlap algorithm leads to basically the same calculation load as the overlap-save algorithm.

To sum everything up, the overlap-save performs an overlap on the inputs then delivers the result, whereas the overlap-add technique performs an overlap not on the input but on the output (see Figure 2.6).

**Exercise 2.3 (FFT filtering)** (see p. 192)

Let \(x(n)\) be a signal such that \(x(n) = \sin(2\pi f_0 n) + \sin(2\pi f_1 n)\), where \(f_0 = 0.15\) and \(f_1 = 0.3\) and let \(h(n)\) be the following impulse response filter:

\[
h(n) = [0.0002 \ 0.0134 \ 0.0689 \ 0.1676 \ 0.2498 \ 0.2498 \ 0.1676 \ 0.0689 \ 0.0134 \ 0.0002].
\]

1. Normalize the filter’s coefficients so as to have the gain at the frequency 0 equal to 1.

2. Display on the same graph the original signal and the filtered signal obtained with the `filter` function.

3. Display the filter’s complex gain and the spectra of the original signal and of the filtered signal.
4. Perform the filtering using an FFT on the entire signal.

5. Perform the same process with length 32 blocks. Notice that this requires an overlap of consecutive blocks.

2.2 Filter banks

The idea of using several parallel filters to “simultaneously” analyze several frequency bands is very old. That is how some analog spectrum analyzers work. Several filters, forming what is called a “filter bank”, with slightly overlapping frequency responses, cover the entire extent of the frequency band we wish to analyze. The short term Fourier transform time-frequency analysis is another example.

Signal spectrum analysis is not the only application of filter banks. For the purpose of processing improvements, we can imagine performing operations on signals coming from different filters. This is what is represented in Figure 2.7.

There are two advantages to this method: on the one hand, the calculations are parallel, and on the other hand, the processes can be adapted to the various channels. Among the main applications of subband filtering techniques, a few are worth mentioning, such as subband coding, multicarrier modulations, analog-to-digital conversion ($\Sigma$-$\Delta$), etc. In this section, we will only present some results concerning filter banks that can be encountered when dealing with processing architectures.
In digital processing, the fact that each channel operates on narrower frequency bands allows the possibility, according to the sampling theorem, of reducing the sampling rate at the filter’s output, as it is shown in Figure 2.7. We end up with a system for which different processing frequencies are used simultaneously in different points of the calculation chain. The oversampling and undersampling operations are examples. Two operations form the basis of these techniques: decimation and expansion.

2.2.1 Decimation and expansion

**Decimation:** an operation that takes one out of every \( M \) samples. Symbolically, it is represented by an arrow pointing down (Figure 2.8). We have:

\[
X_{\downarrow M}(z) = \frac{1}{M} \sum_{k=0}^{M-1} X(z^{1/M}W_M^k) \text{ where } W_M = \exp(-2j\pi/M) \quad (2.6)
\]

**Expansion:** an operation that inserts \( M - 1 \) zeros between two samples of the original sequence. Symbolically, it is represented by an arrow pointing up (Figure 2.9). We have:

\[
X_{\uparrow M}(z) = X(z^M) \quad (2.7)
\]

**Property 2.1 (Filtering and decimation)**

We have the property illustrated by Figure 2.8.

**Hints:** on the right-hand side of Figure 2.8, we have:

\[
Y(z) = U_{\downarrow M}(z) = \frac{1}{M} \sum_{n=0}^{M-1} U(z^{1/M}W_M^n)
\]
By choosing \( U(z) = H(z^M)X(z) \), we obtain:

\[
Y(z) = \frac{1}{M} \sum_{n=0}^{M-1} X\left(z^{1/M}W_n^{-n}\right)H\left[(z^{1/M}W_n^{-n})^M\right] \\
= \sum_{n=0}^{M-1} X\left(z^{1/M}W_n^{-n}\right)H(z) = H(z)X_{\downarrow M}(z) \quad (2.8)
\]

which corresponds to the expression for the process on the left-hand side of Figure 2.8.

**Property 2.2 (Filtering and expansion)**

We have the properties illustrated by Figure 2.9.

HINTS: starting off with the diagram on the left of Figure 2.9, and with \( U(z) = H(z)X(z) \), we have:

\[
Y(z) = U(z^M) = H(z^M)X(z^M) = H(z^M)X_{\downarrow M}(z) \quad (2.9)
\]

corresponding to the expression for the process on the right of Figure 2.9.

**Application: the comb decimation filter**

Consider the filter represented in Figure 2.10. It is composed of the filter with the transfer function \(1/(1 - z^{-1})\) cascaded with the filter with the transfer function \((1 - z^{-M})\).

Hence its transfer function has the expression:

\[
H(z) = \frac{1}{1-z^{-1}} \times (1-z^{-M})
\]
A simplification leads to $H_z(z) = 1 + z^{-1} + \cdots + z^{-(M-1)}$. This is therefore an FIR filter with the impulse response $h(n) = 1$ for $n \in \{0, \ldots, (M-1)\}$ and 0 otherwise. Notice that $1/(1-z^{-1})$ has a pole in $z = 1$ (zero frequency) and that $(1-z^{-M})$ has $M$ zeros placed on the unit circle in $W_m = e^{2j\pi m/M}$ where $m \in \{0, \ldots, (M-1)\}$.

Because of the location of the zeros of $(1-z^{-M})$, regularly spread out on the unit circle, the filter is called a comb filter. After a simplification, $H_z(z)$ has $(M-1)$ zeros in $e^{2j\pi m/M}$ where $m \in \{1, \ldots, (M-1)\}$, and no poles.

The frequency response of the filter $H_z(z)$ is represented in Figure 2.11 for $M = 16$. It has a main lobe centered at 0 with a width of $2/M$. It is therefore a low-pass filter. By placing the first cell’s pole on another of the second cell’s zeros, we can obtain a low-pass filter (see exercise (2.4)).

**Exercise 2.4 (Band-pass filter based on a comb filter)** (see p. 193)

In a method similar to the one used for the low-pass comb filter in Figure 2.10, design a real band-pass filter centered at the frequency $f_m = m/M$. 
Although the filter in Figure 2.11 is, after simplification, an FIR filter that can therefore be achieved by a stable structure, the diagram in Figure 2.10 is unstable because the first cell can produce an unbounded output even if the input is bounded. For systems that use comb filters (see Figure 2.14), it is ensured that this never happens. We will now see an application of the comb filter.

The comb filter represented in Figure 2.10 can be used for designing a low-pass filter (but not a very selective one) in the undersampling operation by placing it before an $M$ order decimator. The result is the first system represented in Figure 2.12.

\[
\begin{align*}
\frac{1}{1 - z^{-1}} & \rightarrow \frac{1}{1 - z^{-M}} \rightarrow \downarrow M \\
\frac{1}{1 - z^{-1}} & \rightarrow \downarrow M \rightarrow 1 - z^{-1}
\end{align*}
\]

**Figure 2.12 – Permutation of the decimator and the derivative filter**

According to property 2.1, the filtering and decimation operations can be performed in any order. We end up with the second system in Figure 2.12. In this system $I(z) = 1/(1 - z^{-1})$ performs the operation that associates the output $u(n)$ with the input $x(n)$ as follows:

\[ u(n) = u(n - 1) + x(n) \]

which is an accumulation/integration and $D(z) = 1 - z^{-1}$ performs the operation that associates the output $y(n)$ with the input $v(n)$ as follows:

\[ y(n) = v(n) - v(n - 1) \]

which can be seen as the approximation of a derivative filter.

The selectivity can be increased by cascading several integrators and several derivative filters, as is done on certain audio CD players using what is called the one-bit stream technique. Figure 2.13 explains how this works.

Starting off with a binary flux at 6.4 MHz, we accumulate input values, 0 or 1, then we decimate by a factor of 4. We obtain a flux of values represented on 20 bits and sampled at 400 kHz. As we have already said, the integrator cascade is, by nature, unstable. However, it can be shown that if we use a modulo $M$ summer, the “integrator, decimator, derivative filter” set does not cause any overflow, so long as the summer contains $M$ bits. Because the cascade is comprised of four of these systems, an $M + 4 = 20$ bits summer is
used. Finally, the calculations performed by the fourth band output filter are processed with 38 bits.

### 2.2.2 Filter banks

An analysis filter bank is a group of parallel digital filters, the input signals of which are $x(n)$, that cuts up the frequency band in $K$ subbands. The synthesis filter bank is a group of $K$ filters placed after the the analysis filter bank and generating the signal $\hat{x}(n)$.

The processing system can be represented by a group of filters connected by undersampling and expansion operators as shown in Figure 2.14.

We now reconsider the problem of perfect reconstruction: is there a filter bank such that the aliasing effects for each band compensate each other exactly on the entire band? This question is justified by the fact that if no processing is done, the least that can be expected of this structure is to produce an output signal identical to the input signal. To make the rest of this discussion simpler, we will consider the case where $M = 2$ according to Figure 2.15, a case of important practical use.
Obvious solutions

Note that the perfect reconstruction problem has at least two obvious solutions. The first one consists of taking two ideal low-pass filters in the $(0, 1/4)$ band for $G_0(z) H_0(z)$, and two ideal high-pass filters in the $(1/4, 1/2)$ band for $G_1(z) H_1(z)$.

The second solution simply consists of choosing $G_0(z) = H_1(z) = 1$ and $G_1(z) = H_0(z) = z^{-1}$. In this case, the analysis filter bank is merely a demultiplexer distributing the even index values of $x(n)$ to one channel and the odd index values to the other channel, while the synthesis filter bank is a multiplexer that interlaces the two channels. This solution uses FIR filters (with only one coefficient!). However, these filters are unfortunately band-pass filters with a gain of 1 without any frequency selectivity.

One of the major problems we are faced with when using filter banks is the difficulty of finding a solution that has a finite impulse response, a good selectivity and the ability to perform perfect reconstruction, all at once.

Perfect reconstruction equations

By referring to the diagram in Figure 2.15, and by using formulae (2.6) and (2.7), we can write:

\[
Z_0(z) = (G_0 X) \downarrow 2 \uparrow 2 = \frac{[G_0(z)X(z) + G_0(-z)X(-z)]}{2}
\]

\[
Z_1(z) = (G_1 X) \downarrow 2 \uparrow 2 = \frac{[G_1(z)X(z) + G_1(-z)X(-z)]}{2}
\]

and:

\[
Y_0(z) = H_0(z)[G_0(z)X(z) + G_0(-z)X(-z)]/2
\]

\[
Y_1(z) = H_1(z)[G_1(z)X(z) + G_1(-z)X(-z)]/2
\]

This leads us to the reconstructed sequence $\hat{x}(n)$ by $\hat{X}(z) = Y_0(z) + Y_1(z)$ and therefore:

\[
2\hat{X}(z) = [G_0(z)H_0(z) + G_1(z)H_1(z)]X(z) + [G_0(-z)H_0(z) + G_1(-z)H_1(z)]X(-z)
\]

\[
= T(z)X(z) + A(z)X(-z)
\]
Perfect reconstruction is ensured when \( \hat{X}(z) = z^{-r}X(z) \). This gives us the following two conditions:

\[
\begin{align*}
T(z) &= G_0(z)H_0(z) + G_1(z)H_1(z) = 2z^{-r} \quad (2.10) \\
A(z) &= G_0(-z)H_0(z) + G_1(-z)H_1(z) = 0 \quad (2.11)
\end{align*}
\]

The first condition ((2.10)) expresses the absence of distortion, due to the fact that the transfer function \( T(z) \) has a gain equal to 1 and a linear phase. The second one ((2.11)) ensures that the term \( X(-z) \), characterizing the spectrum aliasing, is zeroed out.

**Quadrature filters**

A first solution consists of imposing:

\[
H_0(z) = G_1(-z) \text{ and } H_1(z) = -G_0(-z) \quad (2.12)
\]

which ensures condition (2.11). Condition (2.10) then becomes:

\[
H_0(z)G_0(z) - G_0(-z)H_0(-z) = 2z^{-r} \quad (2.13)
\]

If \( H_0(z) \) and \( G_0(z) \) are two polynomials in \( z^{-1} \), equation (2.13) can be expressed \( \alpha(z) - \alpha(-z) = 2z^{-r} \), where we have defined \( \alpha(z) = H_0(z)G_0(z) \).

In \( \alpha(z) - \alpha(-z) \), only the odd degree coefficients of \( \alpha(z) \) remain. Therefore, all the odd degree coefficients of \( \alpha(z) \) must be equal to 0, except for the \( r \)-th degree coefficient. This implies, incidentally, that \( r \) is odd. We then have to factorize \( \alpha(z) = H_0(z)G_0(z) \). Because the two constraints (2.12) are supposed to be obeyed at all times, two simple solutions arise:

- we impose \( G_0(z) = G_1(-z) \). If we change over to the DTFTs, we get \( G_0(e^{2 j \pi f}) = G_1(e^{2 j \pi(f-1/2)}) \). Because the filters are real, this expression implies that the frequency responses of the filters have a “mirror” symmetry about the frequency 1/4. This is called a QMF filter bank, short for Quadrature Mirror Filters. Unfortunately, there are very few solutions, and they are not selective. In order to show this, we replace \( G_0(z) = G_1(-z) \) in the first expression of (2.12), meaning \( H_0(z) = G_1(-z) \), and we get \( G_0(z) = H_0(z) \). Replacing it in (2.13) leads us to:

\[
H_0^2(z) - H_0^2(-z) = 2z^{-r}
\]

For this solution to be satisfied, \( H_0(z) \) cannot have more than two non-zero coefficients, in other words \( H_0(z) = h_0z^{-k_0} + h_1z^{-k_1} \). If we identify the terms, we obtain:

\[
4h_0h_1z^{-(k_0+k_1)} = 2z^{-r}
\]
and therefore $k_0$ and $k_1$ can have any value so long as the sum is odd, for example, $k_0 = 0$ and $k_1 = 1$, and $h_0 h_1 = 1/2$. By imposing that the filters be linear-phase filters, and therefore $h_0 = h_1$, we obtain $h_0 = h_1 = 1/\sqrt{2}$. This result is not satisfactory because the obtained filters are very poorly selective. Thus, the frequency response of $H_0(z)$, for $k_0 = 0$ and $k_1 = 1$, is $|H_0(e^{2j\pi f})|^2 = \cos^2(\pi f)/2$;

- the condition $G_1(z) = G_0(-z)$ is now replaced by $G_1(z) = (-z)^{-N}G_0(-z^{-1})$ or $g_1(n) = (-1)^ng_0(N - n)$, which is equivalent. This is called a CQF filter bank, short for Conjugate Quadrature Filters. By replacing $G_1(z) = (-z)^{-N}G_0(-z^{-1})$ in (2.12), that is to say $H_0(z) = G_1(-z)$, we have $H_0(z) = (-z)^{-N}G_0(z^{-1})$. Replacing it in (2.13) leads us to:

$$z^{-N} \left( G_0(z)G_0(z^{-1}) + G_0(-z)G_0(-z^{-1}) \right)$$

and a sufficient condition on the phase is provided by:

$$G_0(z)G_0(z^{-1}) + G_0(-z)G_0(-z^{-1}) = 1$$

The transfer function $D(z) = G_0(z)G_0(z^{-1})$ is sometimes called a zero-phase half-band. Because $G_0(e^{2j\pi f})$ satisfies $|G_0(e^{2j\pi f})|^2 + |G_0(e^{2j\pi (f - 1/2)})|^2 = 1$, $G_0$ is said to be “power symmetric”. Searching for a solution can be summed up as follows:

**Steps:**

1. Find an odd order, “power symmetric” filter $D(z)$, approximately half-band.

   In order to do this, we can start with the window method (using a triangular window for which positivity is ensured), or with an iterative method such as the Parks-McClellan algorithm [2] (the problem is that there is no guarantee that the phase will be linear).

2. Perform a spectral decomposition of $D(z)$ in $G_0(z)G_0(z^{-1})$.

3. Construct the filter bank using $G_1(z) = (-z)^{-N}G_0(-z^{-1})$, then:

$$H_0(z) = G_1(-z) H_1(z) = -G_0(-z)$$
Orthogonal filters

We will again use equations (2.10) and (2.11), written below:

\[
\begin{align*}
G_0(z)H_0(z) + G_1(z)H_1(z) &= 2z^{-r} \\
G_0(-z)H_0(z) + G_1(-z)H_1(z) &= 0
\end{align*}
\]

and solve this linear system in order to determine the expressions of \( H_0(z) \) and \( H_1(z) \) as an expression of \( G_0(z) \) and \( G_1(z) \). We obtain:

\[
\begin{align*}
H_0(z) &= \frac{2z^{-r}G_1(-z)}{G_0(z)G_1(-z) - G_1(z)G_0(-z)} \\
H_1(z) &= -\frac{2z^{-r}G_0(-z)}{G_0(z)G_1(-z) - G_1(z)G_0(-z)}
\end{align*}
\]

**Property 2.3** For the two channel filter bank, the perfect reconstruction property is obtained if and only if:

\[
\begin{align*}
\sum_k g_0(k)h_0(2n - k) &= \delta(2n - r) \\
\sum_k g_1(k)h_1(2n - k) &= \delta(2n - r) \\
\sum_k g_1(k)h_0(2n - k) &= 0
\end{align*}
\]

Hints: let \( P(z) = H_0(z)G_0(z) \). Using (2.14), we have:

\[ P(z) = 2z^{-r}G_1(-z)G_0(z)D(z) \]

where \( D(z) \) refers to the denominator of \( H_0(z) \) in (2.14). Likewise:

\[ H_1(z)G_1(z) = -2z^{-r}G_0(-z)G_1(z)D(z) \]

Because \( D(z) = -D(-z) \), we have \( H_1(z)G_1(z) = P(-z) \), and we can write:

\[ P(z) + P(-z) = 2z^{-r} \]

This condition implies that \( r \) is even, and that \( p(2n) = \delta(2n - r) \). By noticing that \( P(z) = H_0(z)G_0(z) \) is the \( z \)-transform of the convolution product of \( h_0(n) \) with \( g_0(n) \), we obtain:

\[ \sum_k g_0(k)h_0(2n - k) = \delta(2n - r) \]

Now let \( Q_0(z) = H_1(z)G_0(z) \):

\[ Q_0(z) = -2z^{-r}G_0(z)G_0(-z)D(z) \]
Because $r$ is even, $Q_0(z)$ is odd. And hence $q_0(2n) = 0$. By noticing that $Q_0(z) = H_1(z)G_0(z)$ is the $z$-transform of the convolution of $h_1(n)$ with $g_0(n)$, we get $\sum_k g_1(k)h_0(2n - k) = 0$.

The sequences $g_0(n)$, $g_1(n)$ on one hand, and $h_0(n)$ and $h_1(n)$ on the other, lead to the definition of two sets of orthogonal sequences. Let:

$$\phi_{2n}(k) = g_0(2n - k), \quad \phi_{2n+1}(k) = g_1(2n - k)$$

$$\psi_{2n}(k) = h_0(k - 2n), \quad \psi_{2n+1}(k) = h_1(k - 2n)$$

Property (2.3) shows that the two sequences $\{\phi_n(k)\}$ and $\{\psi_n(k)\}$ verify for any $n \neq n'$:

$$\sum_k \phi_n(k)\psi_{n'}(k) = 0$$

The two sets $\{\phi(n)\}$ and $\{\psi(n)\}$ are said to have the bi-orthogonality property. This is the equivalent for infinite dimension of the property of two matrices such that $\Psi^T\Phi = \text{diag}(d_1, \ldots, d_K)$ where $\text{diag}(d_1, \ldots, d_K)$ is a diagonal matrix, the identity being a particular case.

We will now discuss the orthogonal case [40], where the sequences $\phi_n(k)$ and $\psi_n(k)$ coincide, that is where one is equal to the other translated. A sufficient condition is to have $h_0(n) = g_0(r - n)$ and $h_1(n) = g_1(r - n)$. Hence perfect reconstruction and orthogonality require the impulse responses of the synthesis filters to be reversed copies of the impulse responses of the analysis filters. Changing over to the $z$-transforms, this leads to:

$$H_i(z) = z^{-r}G_i(1/z) \quad \text{where } i = \{0, 1\} \quad (2.15)$$

This means, first of all, that $P(z)$, defined by $P(z) = H_0(z)G_0(z)$ can be written:

$$P(z) = z^{-r}G_0(1/z)G_0(z)$$

This relation implies that if $z_0$ is a root of $P(z)$, then $1/z_0$ is also a root of $P(z)$. Hence, the roots of $P(z)$ are pairs of inverse values, one inside and one outside the unit circle.

By replacing (2.15) in the second equation of (2.14), we get:

$$G_0(-1/z)G_0(z) + G_1(-1/z)G_1(z) = 0$$

If the polynomials $G_0(z)$ and $G_1(z)$ share the same finite degree (FIR filters of the same length) and are different from one another, then the roots of $G_1(z)$ have to be roots of $G_0(-1/z)$. Therefore, $G_1(z) = -z^{2K-1}G_0(-1/z)$. This relation can be expressed, in the temporal domain, as:

$$g_1(n) = (-1)^n g_0(2K - 1 - n)$$
To sum up, calculating analysis filter banks using orthogonal filters is achieved using the following method: starting off with $P(z) = G_0(z)G_0(1/z)$ which verifies $P(z) + P(-z) = 2$:

- we associate with $G_0(z)$ the roots of $P(z)$ that are inside the unit circle, then we calculate $g_0(n)$;
- we calculate $h_0(n) = g_0(-n)$;
- we calculate $g_1(n) = (-1)^n g_0(2K - 1 - n)$;
- we calculate $h_1(n) = g_1(-n)$.

We still have to find a function $P(z) = G_0(z)G_0(1/z)$ such that $P(z) + P(-z) = 2$.

A first crude method consists of imposing the relation $P(z) + P(-z) = 2$ by choosing a sequence $p(n)$ such that $p(2n) = 0$ in the following manner: we start off with an even sequence $w_n$, for example, the one obtained by the FIR filter design method (window method, Parks-McClellan method), and all the even index terms are replaced by zero, except for the zero index term. This can be expressed as follows:

$$p(n) = w_n c_n$$

where $c_{2n} = \delta(n)$. However, this does not guarantee that $P(z)$ can be expressed as $G_0(z)G_0(1/z)$, or that $P(e^{2j\pi f})$ is positive, which is equivalent. We can then determine the sequence $c_n$ such that $P(e^{2j\pi f}) > 0$. As a consequence, the relation $P(z) + P(-z) = 2$ is not quite true anymore, and becomes even less true as the minimum negative value of the DTFT of $w_n$ becomes smaller.

Let us now see an important example related to the Daubechies wavelets. We start with a polynomial $P(z)$, such that it is at the frequency $1/2$. As a consequence, this introduces in the sequence $p(n)$ a kind of regularity similar to the signal smoothing property when the energy of the high frequencies is reduced, hence the idea to place a great number of zeros in $z = -1$. For this we assume:

$$P(z) = (1 + z)^k(1 + z^{-1})^k R(z)$$

where $R(z)$ can be expressed as $R_1(z)R_1(1/z)$. $R(z)$ is therefore a symmetrical polynomial for which the degrees of its terms vary from $-s$ to $+s$. Therefore, $P(z)$ has $2k + 2s$ roots and is dependent on $2k + 2s + 1$ coefficients, $(k + s)$ of which have to be equal to zero ($p(2n) = 2\delta(k)$). This leads to $(k + s)$ equations. Yet we have $2s + 1$ linearly independent coefficients in $R(z)$. This means we have to set $k + s = 2s + 1$, or $s = k - 1$. Thus, for $k = 2$, we get $s = 1$. Hence the length of the filter $G_0(z)$ is 4. Generally speaking, this method leads to FIR filters with lengths of $L = 2k$. 

Let us calculate the coefficients for $k = 2$. For this we assume $R(z) = (\alpha z + \beta + \alpha^{-1})$. The expression of the condition $P(z) + P(-z) = 2$ will give us two equations with two unknowns $\alpha$ and $\beta$. First we have:

$$P(z) = \alpha z^{-3} + (4\alpha + \beta)z^{-2} + (4\beta + 7\alpha)z^{-1} + (8\alpha + 6\beta) + (4\beta + 7\alpha)z + (4\alpha + \beta)z^2 + \alpha z^3$$

The condition:

$$P(z) + P(-z) = 2((4\alpha + \beta)z^{-2} + (8\alpha + 6\beta) + (4\alpha + \beta)z^2) = 2$$

is met if $4\alpha + \beta = 0$ and $8\alpha + 6\beta = 1$. This leads to $\alpha = -1/16$ and $\beta = 1/4$. If we factorize $R(z)$, then associate with $G_0(z)$ the roots inside the unit circle, we get:

$$G_0(z) = \frac{1}{4\sqrt{2}} \left((1 + \sqrt{3}) + (3 + \sqrt{3})z^{-1} + (3 - \sqrt{3})z^{-2} + (1 - \sqrt{3})z^{-3}\right)$$

This leads to $h_0(n) = g_0(-n)$, then $g_1(n) = (-1)^ng_0(3 - n)$ and $h_1(n) = g_1(-n)$.

The following program calculates the coefficients of $G_0(z)$, plots the gains of the analysis filters, and checks the perfect reconstruction property on a trajectory.

```matlab
%===== daub4.m
clear
r=4; % delay due to the bank
g0=[1+sqrt(3);3+sqrt(3);3-sqrt(3);1-sqrt(3)]/4/sqrt(2);
h0=g0(r:-1:1); g1=-h0 .* ((-1) .^ (0:r-1)');
h1=g1(r:-1:1);
%===== gains
Lfft=1024; freq=[0:Lfft-1]/Lfft;
G0f=abs(fft(h0,Lfft)); G1f=abs(fft(h1,Lfft));
subplot(311); plot(freq,[G0f G1f]); grid;
set(gca,'Xlim',[0 .5])
%===== verification
N=1000; x=randn(N,1);
%===== analysis
x0=filter(g0,1,x); x1=filter(g1,1,x);
%===== decimation/expansion
v0=x0; v0(1:2:N)=zeros(N/2,1);
v1=x1; v1(1:2:N)=zeros(N/2,1);
%===== synthesis
y0=filter(h0,1,v0); y1=filter(h1,1,v1);
xchap=y0+y1; max(abs(xchap(r:N)-x(1:N-r+1)))
subplot(312); plot(x(100:120)); grid
subplot(313); plot(xchap(100+r-1:120+r-1)); grid
```
Comments

- We often only restrict ourselves to the two-branch symmetrical filter, because the same segmentation can be applied to both branches (Figure 2.16).

\[ x(n) \rightarrow G_0(z) \downarrow 2 \rightarrow G_{00}(z) \downarrow 2 \rightarrow \ldots \]
\[ G_1(z) \downarrow 2 \rightarrow G_{10}(z) \downarrow 2 \rightarrow \ldots \]

**Figure 2.16 – Decomposition of each branch**

A particular decomposition in *octaves* (Figure 2.17) can be associated with the *wavelets* using multi-scale analysis.

\[ x(n) \rightarrow G_0(z) \downarrow 2 \rightarrow G_{10}(z) \downarrow 2 \rightarrow \ldots \]
\[ G_1(z) \downarrow 2 \rightarrow G_{110}(z) \downarrow 2 \rightarrow \ldots \]

**Figure 2.17 – Decomposition in octaves**

- A commonly used approach in sub-band decomposition techniques uses the FFT calculation structure. Analysis and synthesis filter banks consist of inverse and direct “FFT blocks”. Any reader curious for more information on this method should read [9, 39].

2.3 Ripple control

2.3.1 Principle

The Remez algorithm [32] [33] [31], and its subsequent application to the synthesis of FIR filters by Parks and McClellan [29], is based on the alternance theorems (Borel, 1905) [7] [17] and an optimal error control (La Vallée Poussin,
1910) at each step of an iterative calculation. The objective is to obtain as faithful an approximation as possible of the template we can use for the synthesis of an FIR filter\(^{(1)}\), whose frequency response is a polynomial in \(e^{2j\pi f}\). MATLAB\(^\circ\) provides the function \verb|firpm| in its signal toolbox to perform a synthesis of this type.

Hereinafter, we examine the approximation of continuous functions \(f(x)\) over the interval \([a,b]\), and use the notation \(||.||\) for the norm sup. We look for a polynomial \(p(x)\) which minimizes the error \(\varepsilon(x) = p(x) - f(x)\).

The polynomial approximation theorem establishes the fact that there is a unique polynomial \(p_n\) of degree \(n\) and a subdivision of at least \(n+2\) points \(x_k\), \(a \leq x_0 < x_1 < \cdots < x_{n+1} \leq b\), such that, if, \(\forall k, 0 \leq k \leq n:\)

\[
\frac{f(x_{k+1}) - p_n(x_{k+1})}{f(x_k) - p_n(x_k)} = -1
\]

and 
\[
|f(x_k) - p_n(x_k)| = ||f - p_n||
\]

then \(p_n\) is a better uniform approximation of \(f\) in the sense of the norm sup. We denote it as \(p_n^\ast\).

These are sufficient and also necessary conditions, and \(p_n\) is unique (Chebyshev theorem).

The approximation of the error is expressed as follows: the error \(\varepsilon(x) = p(x) - f(x)\) exhibits at least \(n+2\) oscillations. Let \(x_0, x_1, \ldots, x_{n+1}\) be the abscissa values of the local maxima of the error, and suppose that the expression (2.16) is satisfied (alternance). It can be shown that the optimal error \(\mu\) is such that:

\[
\min_k |p(x_k) - f(x_k)| \leq \mu \leq \max_k |p(x_k) - f(x_k)|
\]

More generally speaking, we can introduce a weighting function \(W(x)\) for the criterion over the interval \([a,b]\), and thus control the ripples of \(W(x)(p(x) - f(x))\).

\(^{(1)}\)eeweb.poly.edu/iselesni/EL713/remez/remez.pdf
2.3.2 Programming

We will look at the case of a low-pass FIR filter with \(2n+1\) coefficients \(h_k\) such that \(h_k = h_{-k}\). The frequency response is of the form:

\[
H(f) = \sum_{k=-n}^{n} h_k \exp(2k\pijf) = h_0 + 2 \sum_{k=1}^{n} h_k \cos(2k\pi f) \tag{2.18}
\]

which we can write as \(H(f) = \sum_{k=0}^{n} \alpha_k \cos^k(2\pi f)\). We will use the notation \(G(f)\) for the desired frequency response, i.e. \(G(f) = 1\) for \(0 \leq f < f_c\) and \(G(f) = 0\) for \(f_c < f \leq 1/2\).

If the polynomial is optimal, the ripples have the same value \(\delta_a\) for the frequencies \(f_0, f_1, \ldots, f_{n+1}\). If we introduce a weighting function \(W(f)\), this gives us:

\[
\begin{align*}
G(f_0) &= h_0 + \cdots + 2h_n \cos(2n\pi f_0) + \frac{\delta_a}{W(f_0)} \\
G(f_1) &= h_0 + \cdots + 2h_n \cos(2n\pi f_1) - \frac{\delta_a}{W(f_1)} \\
&\vdots \\
G(f_{n+1}) &= h_0 + \cdots + 2h_n \cos(2n\pi f_{n+1}) + (-1)^{n+1} \frac{\delta_a}{W(f_{n+1})} \tag{2.19}
\end{align*}
\]

system of \(n+2\) equations with \(n+2\) unknowns (the \(h_k\) and \(\delta_a\)) which we can solve (see function calcak). It may actually be possible to calculate \(\delta_a\) directly.

In fact, we do not know the series of \(f_k\). We take \(n+2\) values including the terminal values (0, 1/2, the limits of the transition band \(f_c - \Delta f\) and \(f_c + \Delta f\)) and the result of the system (2.19) will lead to a polynomial whose ripples are greater than \(\delta_a/W(f)\). However, we know that the optimal error \(\mu\) is such that:

\[
|\delta_a/W| \leq \mu \leq ||H - G|| = M \tag{2.20}
\]

Therefore, we need to choose the \(f_k\) values such that \(|\delta_a/W|\) is maximum. If, during the calculation, \(|\delta_a/W|\) and \(||H - G||\) are sufficiently close to one another, then we can consider that the optimum value has been reached.

The stop test for the computational procedure may use a quality factor \(q_a\) as follows. Consider the relative errors. On the basis of (2.20), we obtain the relation:

\[
\frac{||H - G|| - \mu}{\mu} \leq \frac{||H - G|| - \varepsilon}{\varepsilon} = q_a \tag{2.21}
\]
When $q_a$ falls below a pre-defined value, the procedure is halted. It is noteworthy that, on the basis of (2.20) and (2.21), we also obtain:

$$||H^* - G|| \leq ||H - G|| \leq ||H^* - G||(1 + q_a)$$

This relation was contributed by de la Vallée Poussin.

**Example 2.1 (Synthesis of a low-pass filter)** Here we initialize the $n + 2$ frequencies by taking $f = 0, f = 1/2, f_c - \Delta f$ and $f_c + \Delta f$ where $[-\Delta f, +\Delta f]$ defines the width of the transition band. We also suppose that the weighting function is equal to 1 in the passband and in the attenuation band, equal to 0 in $[f_c - \Delta f, f_c + \Delta f]$. At step 1, the result obtained for the calculation of the polynomial is illustrated in Figure 2.19. We can see that for the frequencies $f_k$ we have an error equal to $\delta_a$ and that error is not optimal. The maximum value of the error $M$ is $> \delta_a$.

![Figure 2.19](image-url)

*Figure 2.19 – Result of the initial calculation for $f_c = 0.2$, $n = 6$ and $\Delta f = 0.02$*

The error function is forced at 0 over the interval $[-f_c, f_c]$. The choice of a new sequence $\{f_k\}$ respects the following principles (Figure 2.20):

- the sign of the error changes for each new point of frequency $f_k$;
- the error for each $f_k$ is $|\varepsilon(f_k)| \geq |\delta|$;
- $|\varepsilon(f_k)| = |\varepsilon_k| > |\delta|$ for at least one of the $f_k$.

At step 1 of the algorithm, the alternance conditions are fulfilled (Figure 2.20). The new frequencies are indicated by $\circ$ corresponding to the maximum and minimum points of the error function.

The maximum values are obtained from the zeros (Figure 2.21) of the derivative of the gain.
Figure 2.20 – Error function and choice of new frequencies

function [ffreq,fk,gainfk,mark,nn]=selnewfr(ir,n,fc,Df)
%!=========================================================================
%! SYNOPSIS: [ffreq,fk,gainfk,mark,nn]=SELNEWFR(ir,n,fc,Df) !
%! ir = coeff. of FIR !
%! fc = cutoff freq. !
%! Df = defines the transition band [fc-Df,fc+Df] !
%! ffreq = freq. sequence <> edge freq. !
%! fk = new freq. sequence !
%! gainfk = gain(fk) !
%! mark = error(fk) !
%!=========================================================================
myeps=10^(-10);
fk=[0;1/2;fc-Df;fc+Df]; % initial values (edge freq.)
%==
nt=fix(length(ir)/2);
pdifcoefs=ir .* [nt:-1:-nt]'*2*pi*1i; % dP/df
rtspol=roots(pdifcoefs); nn=nt;
icond=(1-myeps < abs(rtspol)) & (abs(rtspol) < 1+myeps);
idx=find(icond); rr=rtspol(idx);
ffreq=atan2(imag(rr),real(rr))/pi/2;
idx=find(ffreq>myeps & ffreq<1/2-myeps);
ffreq=ffreq(idx);
fk=[fk;ffreq]; fk=sort(fk);
expc=exp(-2*pi*1i*fk);
gainf=polyval(ir,expc) .* exp(2*pi*1i*nt*fk);
mark=real(gainf); idx=find(fk<=fc-Df); mark(idx)=mark(idx)-1;
if length(fk)== n+2
    if sign(mark(1))==sign(mark(2))
        fk(1)=[]; mark(1)=[];
    elseif sign(mark(end))==sign(mark(end-1))
    end
end
At step 2 of the algorithm we have an additional maximum (Figure 2.22) and alternance is no longer assured. The first frequency is “eliminated”. If alternance were verified, the order of the filter would be altered \((n \rightarrow n + 1)\).

The value of \(n\) can be initialized by using Kaiser's approximation, which gives the theoretical length \(N_t\) of the filter:

\[
N_t = \frac{-10 \log_{10}(\delta_p \delta_s) - 13}{14.6 \times 2 \times \Delta f} + 1;
\]  

\[2.23\]

where \(\delta_p\) and \(\delta_s\) are the ripples in the passband and attenuation band. In our case, \(\delta_p = \delta_s = \delta\). If we arbitrarily take \(\delta = 0.1\):

```
delta=.1; mdelta=1.177479e-01;
Nt=fix((-20*log10(abs(mdelta)) -13)/14.6/2/Df) + 1
```

which gives \(N_t \approx 13\) and \(n = 6\). This generally constitutes a minimum value.
% Weighting function W=Wp=Ws=1
clear
verb=true;
nf=2048; nfs2=nf/2; fr=[0:nfs2-1]/nf;
f=.2; Df=.01; n=15; q0=10^(-3);
[ir,lpcnt,mdelta,gainmat]=LPrems(n,nf,fc,Df,qf0,verb);
sprintf('loop count: %d',lpcnt-1)
sprintf('delta: %d, %d dB',abs(mdelta),20*log10(abs(mdelta))
sprintf('FIR order: %d',length(ir))
if verb
    plot(fr,gainmat(:,end)); grid on
    hold on
    plot([fc-Df fc-Df], [0 1+abs(mdelta)],'r')
    plot([fc+Df fc+Df], [0 1+abs(mdelta)],'r')
    plot([0 fc+Df], [1+mdelta 1+mdelta],'r')
    plot([0 fc+Df], [1-mdelta 1-mdelta],'r')
    plot([fc-Df 1/2],abs([mdelta mdelta]),'r')
    hold off
end

%! function [ir,lpcnt,mdelta,gainmat]=LPrems(n,nf,fc,Df,qf0,verb)
%! %! Low-pass filter
%! %! SYNOPSIS:
%! %! [ir,lpcnt,mdelta,gainmat]=LPREMS(n,nf,fc,Df,qf0,verb) !
%! %! fc = cutoff freq.
%! %! Df = defines the transition band [fc-Df,fc+Df]
%! %! nf = number of FFT points
%! %! n = 2n+1 FIR coefficients
%! %! lpcnt = loop count
%! %! mdelta = ripple amplitude
%! %! gainmat = (nf/2,lpcnt) gain matrix
\begin{verbatim}

%%!===================================================================================================!
fr0=[0:nf-1]/nf; fr=fr0(1:nf/2);
%==== initial freq. sequence f
n1=fix(2*fc*(n+2)); st1=(fc-Df)/(n1-1);
f1=[0:st1:fc-Df];
n2=(n+2-n1); st2=(1/2-fc-Df)/(n2-1);
f2=[fc+Df:st2:1/2];
f=[f1,f2]'; H=[ones(1,n1),zeros(1,n2)];
fk=f;
Hf=zeros(1,nf/2); idx=find(fr<=fc); Hf(idx)=1;
%===========
qfact=inf; gainmat=[]; lpcnt=1;
%==========
while (qfact>qf0)
    %==== best approximation
    [ak,mdelta]=calcak(n,f,fc,Df);
    ir=[ak(n+1:-1:2);ak]; Lir=length(ir); % impulse resp.
    gainf=fft(ir,nf);
    gainf=real(gainf .* exp(2*pi*1i*n*fr0'));
    if verb, gainmat=[gainmat,abs(gainf(1:nf/2))]; end
    lpcnt=lpcnt+1;
    dk=abs(mdelta);
    %====
    merrf=gainf(1:nf/2)-Hf.';
    idx=find(fr>=fc-Df & fr<=fc+Df); merrf(idx)=0;
    %==== selection of a new sequence
    [ffreq,fk,gk,mrk,nn]=selnewfr(ir,n,fc,Df);
    n=nn; M=max(abs(mrk)); f=fk;
    %===========
    qfact=(M-dk)/M;
end

Figure 2.23 – Result for an order equal to 33 with ripples in amplitude −18 dB, Δf = 0.01, a quality factor of 10^{-3} and five iterations (see program testremes)
\end{verbatim}
function [ak,mdelta]=calcak(n,f,fc,Df)
calt=(-1).^(1:n+2)';
mM=[cos(2 * pi * f * [0:n]),calt];
idx=find(f<=fc-Df); n1=length(idx); n2=(n+2-n1);
H=[ones(1,n1),zeros(1,n2)];
coeffs=mM \ H'; ak=coeffs(1:end-1);
ak(2:end)=ak(2:end)/2; mdelta=coeffs(end);
end

Figure 2.24 – Evolutions of $\delta$ (top figure) and of the real number of coefficients of the filter (bottom figure) as a function of $n$. 
Chapter 3

Image Processing

The basic version of MATLAB® offers a number of functions for the processing of images without actually needing to have the toolbox image processing. In particular, it includes the functions image and imagesc, filter2 and conv2, fft2, etc., and the software is also able to deal with numerous formats for opening and saving images.

3.1 A little geometry

3.1.1 3D object

This section discusses the representation of a “3D” object on a screen. The basic idea is to “project” the object onto a screen $P$ placed between the object itself and the observer’s eye (point $C$ in Figure 3.1).

We choose a reference system $(x, y, z)$, and suppose that the screen is orthogonal to the $z$ axis. Given a point $M(x_M, y_M, z_M)$, the point projected onto the screen is $P(x_P, y_P, z_P)$, with $z_P = D_r - D_s$. The eye $C$ of the observer is at $(0, 0, D_r)$. We define $\delta$:

$$\delta = \frac{D_s}{D_r - z_M} \Rightarrow x_P = \delta \times x_M, \ y_P = \delta \times y_M$$ (3.1)

Exercise 3.1 (Projection onto a screen (see p. 194))

Consider a 3D object defined by the function newimg. That function defines a filter referred to as a DoG (Difference of Gaussians) filter.

function [tx,ty,tz,zz,xx]=newimg()
    tx = linspace(-8,8,32)'; tz=tx;
    [zz,xx] = meshgrid(tz,tx);
    yy=(-zz .^ 2 - xx .^ 2);
We are not seeking to obtain a realistic view of the object. Rather we will content ourselves with merely projecting the points which define the object, without worrying about the “facets” which may make it up.

1. Write a function which projects the object onto the screen (Figure 3.2). We will take $D_r = 20$ and $D_s = 12$. The synopsis will be as follows:

```
function [pos2D,lm]=projconique(Ds,Dr,tb3N)
% Dr = distance object reference origin - observer!
% Ds = distance from screen to observer!
% tb3N = (3*N)-array [[x1;y1;z1],...,[xN;yN;zN]]!
% pos2D = coordinates in the P-plane!
```

2. Rather than modifying the position of the observer or of the screen, we will simply move the object. Give the three matrices of rotation around the three axes as a function of the angles $\varphi_x$, $\varphi_y$ and $\varphi_z$, expressed in degrees. Write three functions implementing these transformations. Take the following description as a starting point:
Figure 3.2 – *Construction of the projection*

function vxp=rotx(vp,phix)
    \%!=========================================!
    \%! phix = rotation around x-axis (degrees)!
    \%! vp=(3*N)-array, vxp=(3*N)-array!
    \%!=========================================!

3. Write a program which moves the 3D object (rotational and translational motion) and projects its image onto the screen.

3.1.2 Calibration of cameras

Numerous domains – e.g. robotics or face recognition – use cameras which provide 2-dimensional images which are used to reconstruct a 3D image of an object (Figure 3.3).
Figure 3.3 – System with three cameras: each camera $k$ is modeled by an optical center $C_k$ and a projection plane $\mathcal{P}_k$ representing the sensor. The point $M$ is projected into $m_1$, $m_2$ and $m_3$ on the sensors $\mathcal{P}_1$, $\mathcal{P}_2$ and $\mathcal{P}_3$ of the cameras.

**Calibration parameters**

From the moment an image is captured, efforts are made to deal, as best we can, with the deformations due to the projection of real images on the CCD or CMOS sensor used. The problems may be due:

- to deformations caused by the lens system;
- to the spatial and temporal image sampling rates;
- to different interpretations of the coordinates on each camera, etc.

Hence, before using them, it is helpful to “calibrate the cameras” (in fact we identify the projection model). There are numerous methods in existence [38, 13, 18, 14], as well as numerous software packages, devoted to the operation of calibration.

The projection model uses a **optical center** $C$ which plays the role of the eye mentioned in section 3.1.1. This model, known as the **pinhole camera model**, simplifies the role of the lenses of the objective somewhat, but in practice the results that it yields are acceptable. The distance between the optical center and the image plane is the **focal length** $f$, which we have previously denoted $D_s$ in section 3.1.1.
The calibration parameters come in the form of intrinsic parameters – i.e. those linked to the camera itself – and extrinsic parameters – which correspond to the camera’s position:

- Intrinsic parameters: if we consider a reference space linked to the camera with the origin $C$ and the camera’s own axis as its axis $Oz$ (Figure 3.4), we obtain:

$$\frac{f}{z_M} = \frac{x_m}{x_M} = \frac{y_m}{y_M}$$

![Figure 3.4 – Frame of reference linked to the camera](image)

By using homogeneous coordinates, we find:

$$\begin{pmatrix} x_m \\ y_m \\ 1 \end{pmatrix} = \begin{pmatrix} \chi_m \\ \psi_m \\ \zeta_m \end{pmatrix} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_M \\ y_M \\ z_M \end{pmatrix}$$

(3.2)

At the point of the sensor, the “sensor coordinates” (Figure 3.4), expressed in pixels, are given in the form:

$$\begin{cases} x_s = k_x x_m + x_0 \\ y_s = k_y y_m + y_0 \end{cases} \text{ so that } \begin{pmatrix} x_s \\ y_s \\ 1 \end{pmatrix} = \begin{pmatrix} k_x & 0 & x_0 \\ 0 & k_y & y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_m \\ y_m \\ 1 \end{pmatrix}$$

(3.3)

where $k_x$ and $k_y$ are scaling factors and $(x_0, y_0)$ is the origin of the coordinates in the image plane (sensor). In addition, in order to take account of the “skew” deformations, we introduce a parameter $s/f$ into the above
matrix:
\[
\begin{bmatrix}
\chi_s \\
\psi_s \\
\zeta_s
\end{bmatrix} = z_M 
\begin{bmatrix}
k_x & s/f & x_0 \\
0 & k_y & y_0 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x_m \\
y_m \\
1
\end{bmatrix}
= k_x & s/f & x_0 \\
0 & k_y & y_0 \\
0 & 0 & 1
\begin{bmatrix}
\chi_m \\
\psi_m \\
\zeta_m
\end{bmatrix}
\] (3.4)

We obtain the relation between the coordinates of the point \( M \) in the frame of reference \((C,x,y,z)\) and the sensor coordinates in that same frame of reference:
\[
\begin{bmatrix}
\chi_s \\
\psi_s \\
\zeta_s
\end{bmatrix} = f_k x_s 
\begin{bmatrix}
x_0 \\
y_0 \\
0
\end{bmatrix}
\begin{bmatrix}
\alpha_x & s & x_0 & 0 \\
0 & \alpha_y & y_0 & 0 \\
0 & 0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
x_M \\
y_M \\
z_M \\
1
\end{bmatrix}
= \alpha_x & s & x_0 \\
0 & \alpha_y & y_0 \\
0 & 0 & 1
\begin{bmatrix}
x_M \\
y_M \\
z_M \\
1
\end{bmatrix}
\] (3.5)

We rewrite this as follows:
\[
\begin{bmatrix}
\alpha_x & s & x_0 & 0 \\
0 & \alpha_y & y_0 & 0 \\
0 & 0 & 1 & 0
\end{bmatrix} = \begin{bmatrix}
\alpha_x & s & x_0 \\
0 & \alpha_y & y_0 \\
0 & 0 & 1
\end{bmatrix} \times \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0
\end{bmatrix} = K \times \begin{bmatrix}
I_3 \\
0
\end{bmatrix}
\] (3.6)

\( K \) is called the calibration matrix.

Extrinsic parameters: we still need to express the relation giving \([x_M\ y_M\ z_M]^T\) as a function of \([X_M\ Y_M\ Z_M]^T\) given in a different frame of reference from that of the camera (Figure 3.5).

![Figure 3.5 – Frames of reference of the camera – (C,\(\hat{i},\hat{j},\hat{k}\)) – and of the surrounding universe – (O,\(\hat{I},\hat{J},\hat{K}\))](image)

We suppose that the frames of reference are orthonormal. The conversion from one frame of reference to the other is made by three rotations – by
angles $\alpha$ around $OX$, $\beta$ around $OY$ and $\gamma$ around $OZ$ – followed by a translation. We use the notations $R_x$, $R_y$ and $R_z$ to represent the three rotation matrices associated with these motions.

For the rotations, we have:

$$
\begin{bmatrix}
x_M \\
y_M \\
z_M
\end{bmatrix}
=\begin{bmatrix}
R_z & R_y & R_x
\end{bmatrix}
\begin{bmatrix}
X_M \\
Y_M \\
Z_M
\end{bmatrix}
= R
\begin{bmatrix}
X_M \\
Y_M \\
Z_M
\end{bmatrix}
= \begin{bmatrix}
\cos(\gamma) & \sin(\gamma) & 0 \\
-\sin(\gamma) & \cos(\gamma) & 0 \\
0 & 0 & 1
\end{bmatrix}
\times
\begin{bmatrix}
\cos(\beta) & 0 & -\sin(\beta) \\
0 & 1 & 0 \\
\sin(\beta) & 0 & \cos(\beta)
\end{bmatrix}
\times
\begin{bmatrix}
\cos(\alpha) & 0 & \sin(\alpha) \\
0 & 1 & 0 \\
-\sin(\alpha) & 0 & \cos(\alpha)
\end{bmatrix}
\begin{bmatrix}
X_M \\
Y_M \\
Z_M
\end{bmatrix}
(3.7)
$$

Taking account of the translation $\vec{OC}$, writing $v^T_M = [X_M \ Y_M \ Z_M]$, $V^T_C = [X_C \ Y_C \ Z_C]$, and in homogeneous coordinates we obtain:

$$
\begin{bmatrix}
v_M \\
0 \\
1
\end{bmatrix}
= \begin{bmatrix}
R & -RV_C \\
0 & 1
\end{bmatrix}
\begin{bmatrix}
V_M \\
1
\end{bmatrix}
(3.8)
$$

In summary:

$$
\begin{bmatrix}
X_M \\
Y_M \\
Z_M
\end{bmatrix}
\begin{bmatrix}
R & -RV_C \\
0 & 1
\end{bmatrix}
\begin{bmatrix}
X_M \\
Y_M \\
Z_M
\end{bmatrix}
= \begin{bmatrix}
0 & f & 0 & 0 \\
f & 0 & 0 & 0 \\
0 & 0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
\chi_m \\
\psi_m \\
\zeta_m
\end{bmatrix}
\times
\begin{bmatrix}
k_x & s/f & x_0 \\
k_y & y_0 & 0 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\chi_s \\
\psi_s \\
\zeta_s
\end{bmatrix}
(3.9)
$$

where $\chi_s$, $\psi_s$ and $\zeta_s$ are the coordinates in the sensor plane ((3.5)).

From (3.6) and (3.8) we deduce:

$$
\begin{bmatrix}
\chi_s \\
\psi_s \\
\zeta_s
\end{bmatrix}
= K \times \left[ I_3 \right] \times \begin{bmatrix}
R & -RV_C \\
0 & 1
\end{bmatrix}
\begin{bmatrix}
X_M \\
Y_M \\
Z_M
\end{bmatrix}
= KR \left[ I_3 \right] \times \begin{bmatrix}
V_M \\
1
\end{bmatrix}
(3.9)
$$

It is noteworthy that, in the product $M = KR$, the last row is given by the last row of $R$ which gives the three components of the vector $k$ in the frame
of reference \((O, I, J, K)\) (and thus \([k^T I \quad k^T J \quad k^T K]\)). As \(k\) is a unit vector, we must have \(m_{31}^2 + m_{32}^2 + m_{33}^2 = 1\). In light of (3.9) we deduce that we also have \(p_{31}^2 + p_{32}^2 + p_{33}^2 = 1\).

Thus, the problem of calibration involves estimating the components \(p_{ij}\) of the matrix \(P = KR \begin{bmatrix} I_3 & -V_C \end{bmatrix}\).

In total, there are 11 unknowns: five for the matrix \(K\), three for the rotation of the axes (three angles), and three for the translation.

In our discussion above, we have ignored numerous aspects – particularly the nonlinear effects such as the distortions introduced by the optical system. There are many others, including faults in the centering of the lenses, lack of perpendicularity of the optical axis and the sensor, etc.

### Computation of the calibration parameters

There are many ways to find the calibration parameters. All these methods use “test patterns” which may include checkered patterns, circles, ellipses, etc. The reference [38] describes a calibration grid known as a Tsai grid, often used to calculate the parameters of the system on the basis of the coordinates of the “corners” of each tile on the grid.

It is possible, using a “least-squares’-type method’, to obtain the \(p_{ij}\) of the matrix \(P\). This method, known as the Faugeras and Toscani method, is presented in [13].

Based on (3.9), we can write:

\[
\begin{align*}
x_s &= \frac{\chi_s}{\zeta_s} = \frac{p_{11}X_M + p_{12}Y_M + p_{13}Z_M + p_{14}}{p_{31}X_M + p_{32}Y_M + p_{33}Z_M + p_{34}} \\
y_s &= \frac{\psi_s}{\zeta_s} = \frac{p_{21}X_M + p_{22}Y_M + p_{23}Z_M + p_{24}}{p_{31}X_M + p_{32}Y_M + p_{33}Z_M + p_{34}}
\end{align*}
\]

As we have 11 elements to evaluate, we need at least 6 couples \((m_i, M_i)\). Each such couple gives two linear equations with respect to the parameters \(p_{ij}\):

\[
\begin{align*}
p_{11}X_M + p_{12}Y_M + p_{13}Z_M + p_{14} - x_sp_{34} &= 0 \\
-x_s(p_{31}X_M + p_{32}Y_M + p_{33}Z_M) &= 0 \\
p_{21}X_M + p_{22}Y_M + p_{23}Z_M + p_{24} - y_sp_{34} &= 0 \\
y_s(p_{31}X_M + p_{32}Y_M + p_{33}Z_M) &= 0
\end{align*}
\]

However, it should not be forgotten that the data obtained are subject to noise – estimation of the positions, spatial quantification in the sensors, etc. In
practice, we take around thirty points ($M_n \approx 30$).

\[
\begin{bmatrix}
X_{M_1} & Y_{M_1} & Z_{M_1} & 1 & 0 & 0 & 0 & 0 & -x_{s_1} \\
0 & 0 & 0 & 0 & X_{M_1} & Y_{M_1} & Z_{M_1} & 1 & -y_{s_1} \\
\vdots & & & & \vdots & & & & \vdots \\
X_{M_n} & Y_{M_n} & Z_{M_n} & 1 & 0 & 0 & 0 & 0 & -x_{s_n} \\
0 & 0 & 0 & 0 & X_{M_n} & Y_{M_n} & Z_{M_n} & 1 & -y_{s_n}
\end{bmatrix}
\begin{bmatrix}
p_{11} \\
p_{12} \\
p_{13} \\
p_{14} \\
p_{21} \\
p_{22} \\
p_{23} \\
p_{24} \\
p_{31} \\
p_{32} \\
p_{33} \\
p_{34}
\end{bmatrix} = 0_{2n}
\]

which we write as:

\[
A_p + B_q = 0_{2n} \text{ with } q^T q = 1 \tag{3.10}
\]

We seek to minimize $(A_p + B_q)^T(A_p + B_q)$ with the constraint $q^T q = 1$. For the sake of simplicity of the formulation, we shall set:

\[
\begin{align*}
C &= \begin{bmatrix} A & B \end{bmatrix}, \\
M &= \begin{bmatrix} 0 & 0 \\
0 & I \end{bmatrix}, \\
r &= \begin{bmatrix} p \\
q \end{bmatrix}
\end{align*}
\]

The problem boils down to looking for $r$ such that $||Cr||$ is minimal, where $||Mr|| = 1$. We apply the Lagrange multiplier method, which consists of seeking the extreme values of the function $\mathcal{L}(r, \lambda)$ (see section (6.3)):

\[
\mathcal{L}(r, \lambda) = r^T C r + \lambda(1 - r^T M r) \tag{3.11}
\]

By derivation, we obtain:

\[
\frac{\partial \mathcal{L}}{\partial r} = 2C^T r - 2\lambda M r = 0 \tag{3.12}
\]

We put the value of $\lambda$ found into the criterion $\mathcal{L}$:

\[
\mathcal{L}(r, \lambda) = r^T \lambda M r + \lambda(1 - r^T M r) = \lambda \tag{3.13}
\]

Hence, our problem involves minimizing $\lambda$. With the notations used:

\[
\begin{align*}
A^T A_p + A^T B_q &= 0 \\
B^T B_q + B^T A_p - \lambda q &= 0
\end{align*}
\]
which gives us:

\[
\lambda_q = B^T B q + B^T A p \\
= B^T B q - B^T A (A^T A)^{-1} A^T B q \\
= B^T [I - A (A^T A)^{-1} A^T] B q
\]

(3.14)

We can verify that \( \lambda \) is positive. Indeed, by multiplying (3.14) on the left by \( q^T \), we obtain:

\[
q^T \lambda q = \lambda = q^T B^T B q + q^T B^T A p = q^T B^T B q + p^T A^T A p > 0
\]

The expression obtained by (3.14) shows that \( \lambda \) is an eigenvalue of the matrix \( B^T [I - A (A^T A)^{-1} A^T] B \) and that therefore we need to choose the smallest of them. The corresponding eigenvector gives us \( q \), and then \( p = -(A^T A)^{-1} A^T B q \).

function [r]=calibparam(m,M)
%!================================================!
%! SYNOPSIS [r]=CALIBPARAM(m,M) !
%! m : (2*N) (xs(k);ys(k)), k=1:N matrix (pixels) !
%! M : (3*M) (XM(k);YM(k);ZM(k)), k=1:N matrix !
%! r : (11,1) parameter vector !
%!================================================!

n=size(M,2);
A=zeros(2*n,8);
for k=1:n
    A(2*k-1:2*k,:)=kron(eye(2,2),[M(:,k);1]');
end
MM=[M;M]; MMM=zeros(3,2*n); MMM(:,:,)=MM; MMM(:,:,)=MMM';
mm=-reshape(m,2*n,1)*ones(1,3); B=mm.*MMM;
Q=inv(A'*A)*A';
P=B*(eye(2*n,2*n)-A*Q)*B;
[V,D]=eig(P); [Y,idx]=min(diag(D));
q=V(:,idx); p=-Q*B*q;
r=[p;q];

Separation of the intrinsic and extrinsic parameters

It is possible to obtain \( \mathbf{V}_C \), \( \mathbf{K} \) and \( \mathbf{R} \) in the following manner:

- given \( \mathbf{P} \), we obtain \( \mathbf{V}_C \) by using (3.9):

\[
\mathbf{P} \begin{bmatrix} \mathbf{V}_C \\ 1 \end{bmatrix} = 0 \Rightarrow \mathbf{V}_C = \mathbf{P}^{-1}_{(1:3,1:3)} \mathbf{P}_{(:,4)}
\]

using MATLAB\textsuperscript{\textregistered}-type notations;
- given \( P = KR \left[ \begin{array}{c} I_3 \\ -V_G \end{array} \right] \) we wish to obtain \( K \) and \( R \). Taking the \( M \) (3×3) matrix formed by the first three columns of \( P \), we must perform the factorization \( M = KR \), where \( K \) is an upper triangular matrix and \( R \) an orthogonal base-change matrix whose expression we have already seen \( R = R_z R_y R_x \) ((3.7)).

We can also write \( K = M R^T = M R_z^T R_y^T R_x^T \). We have, successively:

1. \( k_{32} = 0 = m_{32} \cos(\alpha) + m_{33} \sin(\alpha) \):

\[
M R_x^T = M \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} = M' = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & 0 & \cdot \end{bmatrix}
\]

so that:

\[
\cos(\alpha) = \frac{m_{33}}{\sqrt{m_{32}^2 + m_{33}^2}} \quad \text{and} \quad \sin(\alpha) = -\frac{m_{32}}{\sqrt{m_{32}^2 + m_{33}^2}} \quad (3.15)
\]

2. \( k_{31} = 0 = m_{31}' \cos(\beta) - m_{33}' \sin(\beta) \):

\[
M'R_y^T = M' \times \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix} = M'' = \begin{bmatrix} \cdot & \cdot & \cdot \\ 0 & 0 & \cdot \end{bmatrix}
\]

thus:

\[
\cos(\beta) = \frac{m_{33}'}{\sqrt{m_{31}'^2 + m_{33}'^2}} \quad \text{and} \quad \sin(\beta) = \frac{m_{31}'}{\sqrt{m_{31}'^2 + m_{33}'^2}} \quad (3.17)
\]

3. \( k_{21} = 0 = m_{21}'' \cos(\gamma) + m_{22}'' \sin(\gamma) \):

\[
M'' R_z^T = M'' \times \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cdot & \cdot & \cdot \\ 0 & 0 & 1 \end{bmatrix}
\]

hence:

\[
\cos(\gamma) = \frac{m_{22}''}{\sqrt{m_{21}''^2 + m_{22}''^2}} \quad \text{and} \quad \sin(\gamma) = -\frac{m_{21}''}{\sqrt{m_{21}''^2 + m_{22}''^2}} \quad (3.19)
\]

We obtain:

\[
K = M R_z^T R_y^T R_x^T \quad \text{with} \quad R = R_z R_y R_x \quad (3.20)
\]
3.2 Pyramidal decompositions

The pixels in an image are generally strongly correlated with their neighbors. It is easy to see that, in order to generate an acceptable representation of an image, it is sufficient to use an under-sampled version of that image. The operation of under-sampling causes the loss of information, which it is possible to compensate for by using the error rate committed during that operation. Hence, we obtain the scheme of the principle shown in Figure 3.6.

![Figure 3.6 – Decomposition by iterative filtering](image)

The symbol \( \circledast \) represents a decimation by a factor \( 1/2 \). Based on the image...
$P_0$, which is under-sampled by a factor 2, we obtain $P_1$, and the prediction error $L_0 = P_0 - Q_0$ which contains the high-frequency data from the original image. The process can be repeated in successive iterations.

One of the advantages to this type of operation (though not the only one) is image compression. The image $P_n$ is relatively small, and the “errors” $L_p$ can be encoded on a reduced number of bits.

The operation of reconstruction (Figure 3.7) introduces errors due to the interpolations (the symbol $\mathbf{1}$ represents an interpolation of ratio 2).

\[
P_3 = L_3 \rightarrow \hat{P}_3 \approx Q_2 \approx P_2 \rightarrow \hat{P}_2 \approx Q_1 \approx P_1 \rightarrow \hat{P}_1 \approx Q_0 \approx P_0
\]

**Figure 3.7 – Reconstruction**

The program `tstpyramide.m` illustrates the decomposition with the “low-frequency” and “high-frequency” parts represented in Figure 3.8.

```matlab
function wfltrBurtAdelson=BurtAdelsonInit
%============================
% Un des filtres passe-bas
```

The coefficients of the low-pass filter are given by the initialization function `BurtAdelsonInit.m`:

```matlab
function wfltrBurtAdelson=BurtAdelsonInit
%============================
% Un des filtres passe-bas
```
%========================================
rac2=sqrt(2);
cHBurtAdelson = [rac2*[-1 5 12 5 -1]/20];
cHtildeBurtAdelson = [rac2*[-3 -15 73 170 73 -15 -3]/280];
wfltrBurtAdelson=struct('cH',cHBurtAdelson, ...
    'cHt',cHtildeBurtAdelson);

Figure 3.8 – Pyramidal decomposition

3.2.1 Pyramidal decomposition given by Burt and Adelson

The construction scheme presented above does not take account of the fact that it is not useful to actually compute all of the points (low-pass filtering is
applied to the whole image). It is sufficient to compute only those points which are necessary [4]. However, the “high-frequency” part cannot be performed directly. Hence, the result must be interpolated as shown in Figure 3.9.

![Figure 3.9 – Decomposition according to Burt and Adelson](image)

\[
\begin{align*}
P_0 & \quad Q_0 \\
L_0 & \quad \downarrow \\
+ & \quad 1 \\
\downarrow & \quad L_0 \\
P_1 & \quad Q_1 \\
L_1 & \quad \downarrow \\
+ & \quad 1 \\
\downarrow & \quad L_1 \\
P_2 & \quad Q_2 \\
L_2 & \quad \downarrow \\
+ & \quad 1 \\
\downarrow & \quad L_2 \\
P_3 &= L_3
\end{align*}
\]

Figure 3.9 – Decomposition according to Burt and Adelson

\[
\begin{align*}
P_3 &= L_3 \\
\hat{P}_3 & \quad \times \quad P_2 \quad \downarrow \\
\hat{P}_2 & \quad \times \quad P_1 \quad \downarrow \\
\hat{P}_1 & \quad \times \quad P_0 \quad \downarrow \\
\downarrow & \quad L_2 \\
\downarrow & \quad L_1 \\
\downarrow & \quad L_0
\end{align*}
\]

Figure 3.10 – Reconstruction

3.2.2 Pyramidal decomposition using a Haar transformation

We apply a transformation which tends to decompose the image into four frequency bands. Consider the transformation:

\[
T = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \end{bmatrix}
\]

Each \(X(2 \times 2)\) block is transformed as follows:

\[
Y = TXT^T = T \begin{bmatrix} a & b \\ c & d \end{bmatrix} T^T = \begin{bmatrix} a + b + c + d & (a + c) - (b + d) \\ (a + b) - (c + d) & (a + d) - (b + c) \end{bmatrix} / 2
\]

Here, we notice four terms corresponding to a mean \((a + b + c + d)\), a horizontal derivative \(((a + c) - (b + d))\), a vertical derivative \(((a + c) - (b + d))\) and a second derivative \(((a + d) - (b + c))\). The program `dwtproc.m` gives a direct realization of this.

```matlab
%----- dwtproc.m -----
figure(1), clf
pixc=imread('boats.jpg','JPG'); diml=size(pixc);
cmap=[0:1/255:1]'*ones(1,3);
ha=axes('units','pix','position',[30 30 fliplr(diml)]);
```

```matlab
T = 1/sqrt(2) * [1];

Each X(2 x 2) block is transformed as follows:

Y = TXT* = T * [a b; c d] * T' = [a + b + c + d (a + c) - (b + d); (a + b) - (c + d) (a + d) - (b + c)] / 2

Here, we notice four terms corresponding to a mean (a + b + c + d), a horizontal derivative ((a + c) - (b + d)), a vertical derivative ((a + c) - (b + d)) and a second derivative ((a + d) - (b + c)). The program `dwtproc.m` gives a direct realization of this.
66 Digital Signal and Image Processing using MATLAB

```matlab
imagesc(pixc), axis('square'), colormap(cmap)
%===== Construction directe
A=double(pixc(1:2:end,1:2:end)); B=double(pixc(1:2:end,2:2:end));
C=double(pixc(2:2:end,1:2:end)); D=double(pixc(2:2:end,2:2:end));
pixcLL=(A+B+C+D)/2; pixcHL=(A+C-B-D)/2;
pixcLH=(A+B-C-D)/2; pixcHH=(A-C-B+D)/2;
pixcf=[normM(pixcLL,cmap),normM(pixcHL,cmap);
      normM(pixcLH,cmap),normM(pixcHH,cmap)];
figure(2), clf
hc=axes('units','pix','position',[30 30 fliplr(diml)]);
imagesc(pixcf), axis('square'), colormap(cmap)
```

In writing this program, it would have been possible to use the 2D filtering functions in MATLAB®, but the result would have taken 4-5 times as long to obtain. The results are normalized so they can be displayed in a single table:

```matlab
%===== dwtprocf.m =====
pixc=imread('boats.jpg','JPG'); diml=size(pixc);
cmap=[0:1/255:1]'*ones(1,3);
%===== Construction par filter2
T=[1 1; -1, -1]; Tt=T';
pixcLL=filter2(ones(2,2),pixc); pixcLH=filter2(T,pixc);
pixcHL=filter2(Tt,pixc); pixcHH=filter2(T*Tt,pixc);
%===== Normalisation pour affichage
pixcLL=normM(pixcLL,cmap); pixcHL=normM(pixcHL,cmap);
pixcLH=normM(pixcLH,cmap); pixcHH=normM(pixcHH,cmap);
pixc3=[pixcLL(1:2:end,1:2:end) pixcHL(1:2:end,1:2:end);...
      pixcLH(1:2:end,1:2:end) pixcHH(1:2:end,1:2:end)];
clf
hb=axes('units','pix','position',[30 30 fliplr(diml)]);
imagesc(pixc3), axis('square'), colormap(cmap)
```

```matlab
function pixct=normM(pixc,cmap)
Lc=size(cmap,1);
Mx=max(max(pixc)); mx=min(min(pixc));
pixct=1+(pixc-mx)./(Mx-mx)*(Lc-1);
```

### 3.2.3 Stepwise decomposition (lifting scheme)

Up until now, the decomposition has been based on the spectral content. We have separated out the low- and high-frequency components. The lifting scheme is based on a spatial decomposition (e.g. a decimation). The two separate elements are generally strongly correlated. By exploiting this correlation, we are able to “compress” the representation.

Consider $P_0$ and the two components $P_1$ and $Q_1$. We can use $P_1$ to predict $Q_1$. Rather than predicting $Q_1$, we predict $\hat{Q}_1 = Q_1 - \mathcal{P}(P_1)$, where $\mathcal{P}$ denotes the predictor. We then attain the setup illustrated in Figure 3.13.
The scheme (3.13) yields an image $P_n$ obtained on the basis of successive separations which exhibit a phenomenon of aliasing. In order to limit that aliasing, we “update” the separate component on the basis of the prediction error (operator $\mathcal{W}$ in Figure 3.14).

Example 3.1 (Lifting scheme)
[36] gives the following example, in which the operator $\mathcal{P}$ is a simple interpolation (the mean of two neighbors):

\begin{align*}
R_0(k) &= P_0(2k) \\
Q_0(k) &= P_0(2k + 1) \\
\overline{P}_1(k) &= Q_0(k) - (R_0(k) + R_0(k + 1))/2 \\
&= P_0(2k + 1) - (P_0(2k) + P_0(2k + 2))/2
\end{align*}

which is tantamount to filtering the sequence $P_0$ with the impulse response filter $\{-\frac{1}{2}, 1, -\frac{1}{2}\}$ (high-pass).
Figure 3.12 – Pyramidal decomposition using a Haar transformation

Figure 3.13 – Setup using only prediction (I performs the separation)

Figure 3.14 – Setup using prediction and updating
Updating consists of using the $P_1(k)$ and $P_1(k-1)$ as follows:

$$P_1(k) = R_0(k) + A(P_1(k) + P_1(k-1))$$

$$= -\frac{A}{2}P_0(2k+2) + AP_0(2k+1) + (1 - A)P_0(2k) + AP_0(2k-1) - \frac{A}{2}P_0(2k-2)$$

corresponding to an impulse response filter \{-\frac{A}{2}, A, (1 - A), A, -\frac{A}{2}\} (low-pass). The choice of $A$ is made by supposing that the sum $\sigma = \sum_k P_0(k)$ is kept constant. Using (3.22) and then (3.21):

$$\sum_k P_0(k) = \sum_k P_1(k) = \sum_k R_0(k) + 2A \sum_k P_1(k)$$

$$= \sum_k P_0(2k) + 2A \sum_k P_0(2k+1) - A \sum_k P_0(2k) - A \sum_k P_0(2k+2)$$

$$= (1 - 2A) \sum_k P_0(2k) + 2A \sum_k P_0(2k+1)$$

Equality is assured only if $A = 1/4$. The gains of the two filtering operations are given by Figure 3.15.

From a spectral point of view, the operations of decimation and reconstruc-
tion can be described by:

\[
R_0(z) = \frac{P_0(z^{1/2}) + P_0(-z^{1/2})}{2} \quad \text{or} \quad R_0(f) = \frac{P_0(f/2) + P_0((f - 1)/2)}{2}
\]

\[
Q_0(z) = \frac{z^{1/2}}{2} \left( P_0(z^{1/2}) - P_0(-z^{1/2}) \right)
\]

or \( Q_0(f) = \frac{e^{\pi j f}}{2} (P_0(f/2) - P_0((f - 1)/2)) \)

The reconstruction \( \mathcal{J} \) is expressed as:

\[
P_0(z) = R_0(z^2) + z^{-1}Q_0(z^2)
\]

or \( P_0(f) = R_0(2f) + e^{-2\pi j f}Q_0(2f) \)

We can write (we suppose that \( P \) and \( U \) are Finite Impulse Responses (FIRs), with the respective transfer functions \( B(z) = \sum_{k=0}^{M-1} b_k z^k \) and \( A(z) = \sum_{k=0}^{N-1} a_k z^{-k} \):

\[
\mathcal{P}_1(z) = Q_0(z) - \left( \sum_{k=0}^{M-1} b_k z^k \right) R_0(z) = Q_0(z) - \mathcal{B}(z)R_0(z)
\]

and \( P_1(z) = R_0(z) + \left( \sum_{k=0}^{N-1} a_k z^{-k} \right) \mathcal{P}_1(z) = R_0(z) + \mathcal{A}(z)\mathcal{P}_1(z) \)

\[
Q_0(z) = \frac{z^{1/2}}{2} \left( P_0(z^{1/2}) - P_0(-z^{1/2}) \right) / 2
\]

If the construction were perfect, then for any value of \( P_0(z) \) we should have:

\[
P_0(z) = P_1(z^2) + z^{-1}\mathcal{P}_1(z^2)
\]

\[
= R_0(z^2) + (\mathcal{A}(z^2) + z^{-1}) \mathcal{P}_1(z^2)
\]

\[
= R_0(z^2) + (\mathcal{A}(z^2) + z^{-1}) (Q_0(z^2) - \mathcal{B}(z^2)R_0(z^2))
\]

\[
= R_0(z^2) \left[ 1 - \mathcal{B}(z^2) (\mathcal{A}(z^2) + z^{-1}) \right] + Q_0(z^2) \left( \mathcal{A}(z^2) + z^{-1} \right)
\]

so:

\[
P_0(z) = \frac{1}{2} (P_0(z) + P_0(-z)) \left[ 1 - \mathcal{B}(z^2) (\mathcal{A}(z^2) + z^{-1}) \right]
\]

\[+ \frac{z}{2} (P_0(z) - P_0(-z)) \left[ \mathcal{A}(z^2) + z^{-1} \right] \]
Chapter 4

Numerical Calculus and Simulation

This chapter is devoted to the problems of numerical calculus. Although numerous functions – the conversion from a continuous-time system to a discrete-time system, seeking of zeros, of minima and maxima, solving of differential equations, etc. – are provided in the basic version of MATLAB®, it is useful to have some understanding of the methods used. An abundant body of literature is available on the subject, as are numerous publications in FORTRAN, C or other languages. Thus, we will content ourselves here with describing a number of techniques commonly used in devoted software packages.

4.1 Simulation of continuous-time systems

4.1.1 Simulation by approximation

Design methods based on continuous-discrete time changes actually consist of constructing a discrete-time simulator of a linear differential equation. This method provides satisfying results because the simulated systems are linear. Many precautions would have been needed had they not.

Exercise 4.1 illustrates the implementation of an RC filter simulator subjected to a periodic input.

Exercise 4.1 (Full-wave rectifier and simulation) (see p. 197)

Consider a full-wave rectifier followed by an RC filter (Figure 4.1).

1. The input signal \( s(t) = A\sin(2\pi F_0 t) \) with \( F_0 = 50 \) Hz is fed to the rectifier. Determine the Fourier series expansion of the rectified signal.
Figure 4.1 – Full-wave rectifier

\[ x(t) = |s(t)|. \] What is the amplitude of the continuous component of \( x(t) \)?

2. The output voltage \( y(t) \) of the RC filter verifies the differential equation:

\[ RC \frac{dy(t)}{dt} + y(t) = x(t) \]

Using the properties of the Fourier transform, determine the expression of this filter’s complex gain \( H(F) \);

3. \( 1/RC \) is chosen to be much greater than \( F_0 \) so that only the continuous component and the first harmonics remain in the output signal. What is, in this case, the expression of \( y(t) \)?

4. We wish to simulate this system. In order to do so, we perform a sharp enough discretization of time by choosing a sampling frequency \( F_s = 1/T_s = 5,000 \) Hz much greater than the input sine’s frequency.

We define \( x_s(n) = x(nT_s) \) \( y_s(n) = y(nT_s) \). By using the Euler approximation \( dy/dt \approx (y(nT_s) - y((n-1)T_s)) \times F_s \), show that the differential equation is equivalent to the recursive equation \( y_s(n) + (\tau - 1)y_s(n-1) = \tau x_s(n) \). Determine the expression of \( \tau \) as a function of \( RC \) and \( T_s \).

5. Write a program that simulates the system’s output voltage when the input is a sinusoidal voltage with a frequency of 50 Hz and an RMS (Root-Mean-Square) voltage of 220V. The filter’s time constant is \( RC = 0.01 \) s. The filter function will be used to generate the output signal.

### 4.1.2 Exact model simulation

Consider a continuous-time filter with a frequency response that tends to 0 when the frequency tends to \(+\infty\) (this is called a **strictly proper filter**). The relation between the input \( i(t) \) and the output \( o(t) \) is then assumed to be
described by a constant coefficient linear differential equation. This system can be represented with the use of state equations as follows:

\[
\begin{align*}
\frac{dx(t)}{dt} &= Ax(t) + bi(t) \\
o(t) &= c^T x(t) + di(t)
\end{align*}
\]  

(4.1)

where \(b, c\) and \(x(t)\) are \(n \times 1\) vectors and \(d\) is a scalar, equal to zero. \(A = [a_{ij}]\) is an \(n \times n\) matrix with its \([a_{ij}]\) time-independent. \(x(t)\) is called the state vector of this representation, which is far from being the only possible one.

It can be shown that the solution to the first of the system’s equations is:

\[
x(t) = e^{A(t-t_0)} x(t_0) + \int_{t_0}^{t} e^{A(t-u)} b i(u) du
\]

(4.2)

where the matrix exponential (the MATLAB® function \texttt{expm}) is given by:

\[
e^{At} = I + \frac{t}{1!} A + \frac{t^2}{2!} A^2 + \cdots + \frac{t^n}{n!} A^n + \cdots
\]

Given this definition, you can check that:

\[
d e^{A t} \over dt = A e^{A t} = e^{A t} A
\]

and:

\[
A \int_{0}^{t} e^{A u} du = \int_{0}^{t} e^{A u} du \times A = (e^{A t} - I)
\]

The input-output relation is obtained by applying the Laplace transform to the system (4.1). We get:

\[
O(s) = c^T (sI - A)^{-1} b I(s)
\]  

(4.3)

where \(I(s)\) and \(O(s)\) are the Laplace transforms of \(i(t)\) and \(o(t)\) respectively.

**Example 4.1 (Second order systems)** Consider a continuous-time system, with the input \(i(t)\) and the output \(o(t)\), both scalar, described by the differential equation:

\[
\frac{d^2 o(t)}{dt^2} + a_1 \frac{do(t)}{dt} + a_2 o(t) = i(t)
\]  

(4.4)

Let:

\[
x(t) = \begin{bmatrix} o(t) & \frac{do(t)}{dt} \end{bmatrix}^T
\]
With this choice of the state vector, equation (4.4) leads us to a state representation:

\[
\begin{align*}
\frac{d\mathbf{x}(t)}{dt} & = \begin{bmatrix} 0 & 1 \\ -a_2 & -a_1 \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} i(t) \\
o(t) & = [1 \\ 0] \mathbf{x}(t)
\end{align*}
\]

similar to that of (4.1). We have to check that the input-output relation is:

\[ O(s) = \frac{1}{s^2 + a_1 s + a_2} I(s) \]

by applying the Laplace transform to (4.4) (the initial conditions are assumed to be equal to zero) or by (4.3).

Exercise 4.2 illustrates the implementation, in the particular case of a simulation based on a state representation. Such a simulation is particularly useful in automatic control where the filter’s output (called a compensator in this context) is applied to a continuous-time process through a digital-to-analog converter.

**Exercise 4.2 (Simulation in the presence of a ZOH)** (see p. 200)
We wish to simulate the behavior of a continuous-time system described by a state representation. The input signal is obtained with a ZOH (Zero-Order Hold) DAC that maintains the input value during the sampling period (Figure 4.2).

**Figure 4.2 – Continuous-time system fed through a ZOH DAC**

1. Give, based on expression (4.2), the relation between the state vector at the time \((k+1)T\) and the state vector at the time \(kT\). Show that we can find a relation similar to:

\[ \mathbf{x}((k+1)T) = \Phi(T) \mathbf{x}(kT) + i(kT) \Psi(T) \mathbf{b} \]

where \(\Psi(T)\) is obtained as a \(k\)-independent integral of \(\Phi(t) = e^{At}\). Notice that the integral \(\int_0^t e^{Au} du\) does not require the calculation of \(A^{-1}\), the
invertibility of which is not certain. It can actually be calculated directly by taking the exponential $\exp(A_e t)$ with:

$$A_e = \begin{bmatrix} A & b \\ 0^T & 0 \end{bmatrix}$$

2. Give $x(t)$ as a function of $\Phi(t)$ and $\Psi(t)$, for $t \in (kT, (k+1)T)$.

3. Consider the filter defined by the differential equation (4.5):

$$\frac{d^2 o(t)}{dt^2} + 1.4 \frac{do(t)}{dt} + o(t) = 1(t \in [0, +\infty))$$  \hspace{1cm} (4.5)

with the unit step as its input. Starting with the initial conditions:

$$o(0) = 0 \quad \frac{do(t)}{dt} \bigg|_{t=0} = 0$$

simulate the response to the unit step.

Notice that, in this exercise, the unit step response given in question 3 is correct. This means that the calculated values of the output sequence coincide with the values $o(nT)$ for all $T$. Any other technique involving a transformation (Euler or bilinear) on the continuous-time system can only yield an approximation.

**Exercise 4.3 (Non-minimal system)**  \hspace{1cm} (see p. 202)

The use of linear, time-invariant, continuous-time models in the form of state representations has led us the following expressions:

$$A = \begin{bmatrix} -11/4 & -11/8 & -5/4 \\ 27/4 & 11/8 & 21/4 \\ 15/8 & 19/16 & 5/8 \end{bmatrix}$$

$$b = \begin{bmatrix} 1 \\ -1 \\ -1/2 \end{bmatrix}$$

$$c^T = \begin{bmatrix} 3/8 & 1/2 & -1/4 \end{bmatrix}, d = 0$$  \hspace{1cm} (4.6)

1. We wish to “digitally” simulate the response of this system when fed by a unit step through a ZOH. Perform this simulation using the results of the previous exercise, for several sampling values and for a minimum simulation duration of 10 s.

2. Observe the evolution of the $\|\cdot\|_\infty$ norm of the state vector during the simulation. What conjecture can be made concerning the stability?
3. Find the transfer function (Laplace transform) associated with the system (4.6). What can be said of the stability? Comment on the role played by initial conditions in a system's behavior (we have the same result in continuous and discrete time cases).

4. Perform the simulation for a period of about 3 minutes. What happens? What is this phenomenon caused by? It can be verified that the discrete TF can be calculated by \( \frac{\text{poly}(\phi - \psi \cdot \text{c}')}{\text{poly}(\phi)} - 1 \).

4.2 Solving of ordinary differential equations (ODEs)

So-called ordinary differential equations of order \( n \) are defined by an equation such that:

\[
\mathcal{F}(x, y, y', \ldots, y^{(n)}) = 0
\]

where \( y \) is a vector in normed vector space \( E \), \( \mathcal{F} \) is continuous over an opening in \( \mathbb{R} \times E^{n+1} \).

4.2.1 Conversion from continuous to discrete time

The conversion from a continuous to a discrete regime can be made by way of a bilinear transformation. Control engineers use the term “Tustin method” to speak of that conversion. The Tustin method involves replacing \( s \) in the transfer function (Laplace transform) with \( \frac{2}{T}(1 - z^{-1})/(1 + z^{-1}) \), where \( T \) is the computation step and \( z^{-1} \) is the delay operator. This approximation corresponds to the approximate computation of an integral by the trapeze method. Consider the linear equation:

\[
\tau \frac{dy}{dt} + y(t) = x(t)
\]

with which the following transfer function is associated:

\[
G(s) = \frac{1}{1 + \tau s}
\]

Changing the variables gives:

\[
H(z) = \frac{T(1 + z^{-1})}{T + 2\tau + (T - 2\tau)z^{-1}}
\]

which yields the recurrent equation:

\[
(T + 2\tau)y_n + (T - 2\tau)y_{n-1} = Tx_n + Tx_{n-1}
\]

It can be implemented in the following simple way:
function ctime2dtime()
% continuous time to discrete time
tau=0.6; T=.03; t=(0:T:7);
% discrete time linear system
numd=[T,T], dend=[(T+2*tau),(T-2*tau)]
[myinp]=f(t);
y=filter(numd,dend,myinp);
clf, plot(t,myinp,'r',t,y,'b'), grid
end

function myinp=f(t)
f0=0.5; y=sin(2*pi*f0*t); myinp=sign(y);
end

or, using the function nbilin (see [1]):

function [B,A]=nbilin(pol,Ts)
% Bilinear transform of a polynomial
% SYNOPSIS: [B,A]=NBILIN(pol,Ts)
% pol = polynomial (decreasing powers of s)
% Ts = sampling period
% B,A = numerator and denominator of the result
if nargin<2, Ts=1; end
NX=[1 -1]*2/Ts; DX=[1 1];
nP=length(pol); PP=zeros(nP,1); PP(:)=pol;
B=pol(1); A=[1];
for k=2:nP
    A=conv(A,DX); B=conv(B,NX) + pol(k)*A;
end

with the test program:

function testbilin()
tau=.6; Ts=.03; t=(0:T:7);
um=[1]; den=[tau,1];
[Bn,An]=nbilin(num,Ts); [Bd,Ad]=nbilin(den,Ts);
B=conv(Bn,Ad); A=conv(An,Bd);
myinp=f(t); y=filter(B,A,myinp);
clf, plot(t,myinp,'-r',t,y,'-b'), grid
end

function myinp=f(t)
f0=0.5; y=sin(2*pi*f0*t); myinp=sign(y);
The transformation may be modified somewhat. The modification is known as *pre-warping*, and is performed as follows:

\[
\frac{2 \frac{1 - z^{-1}}{T}}{1 + z^{-1}} \rightarrow \frac{1 - z^{-1}}{1 + z^{-1}} \times \frac{\omega_a}{\tan \left(\omega_a \frac{T}{2}\right)}
\]

where \(\omega_a\) is called the *critical frequency*.

**Remarks:**

- the functions of continuous-to-discrete-time conversion are available in toolboxes linked to the field of automation engineering;
- the bilinear transform is available in the toolbox *robust control* with the function *bilin* and the toolbox *bilinear* with *bilinear*;
- the solution in the above program is obtained by numerical filtering (with the function *filter*). This function solves the recurrent equation obtained for the given input. Here, we apply a “square wave” signal, constructed by \(\text{sign}(\sin(2\pi f_0 t))\);
- this method remains problematic when there are discontinuities, either in the system or in the input signal, as is the case here.

### 4.2.2 Linear case, continuous-time solution

MATLAB® has numerous functions for solving differential equations, be they linear or otherwise. The function *ode45* is one such tool, and can be used to solve the differential equation \(\frac{dy}{dt} = f(t, y)\).

**Example 4.2 (Use of the function *ode45*)** At the input to an RC filter, we apply a “square wave” signal, and we wish to display the output. The transfer function for the filter is:

\[
G(s) = \frac{1}{1 + RCs} = \frac{1}{1 + \tau s}
\]  

(4.12)

In the linear case, the functions *tf2ss* and *ss2tf* can be used to make the conversion between transfer functions and representations of state:

\[
\frac{dv}{dt} = Av + Bx \\
y = Cv + Dx
\]

where \(v(N \times 1)\) is the state vector, \(x(M \times 1)\) the input vector and \(y(P \times 1)\) the output vector. The matrix \(A(N \times N)\) is the state matrix.

In the above example, the order of the system is \(N = 1\).
>> [A,B,C,D]=tf2ss(num,den)
A =
   -1.6667
B =
    1
C =
   1.6667
D =
    0

The simulation can be performed as follows:

%===== repcren.m
%===== linear system
num=1; tau=0.6; den=[tau 1];
[A,B,C,D]=tf2ss(num,den); % CT linear system
y0=sqrt(tau)/2;
%===== simulation
[t,y,myinp]=solve_myls(A,B,y0);
%===== drawing
myoutp=C*y+D*myinp;
plot(t,myinp,'r',t,myoutp,'b'), grid on

which uses the function:

function [t,y,myinp]=solve_myls(A,B,y0)
%===== simulation
tspan=[0:.03:7]'; myinp=sign(sin(pi*tspan));
[t,y] = ode45(@(myls,tspan,y0)
%===== local function
    function yp=myls(t,y)
    yp=A*y+B*sign(sin(pi*t));
end

end

The function solve_myls uses a pointer, @myls, to the function describing the ODE. The input parameters for that pointer, @myls, are a vector y and a scalar t. We can see that the function myls is local in solve_myls. Thus, it is able to pass the relevant parameters (here A and B) to myls.

In the program, we have defined an initial state which renders a certain output myoutp equal to 1/2 at the initial time. Here, we have a first order, and the scalar linking the output to the state can be obtained in the following manner:

>> [A,B,C,D]% state representation
ans =
   -1.6667    1.0000    1.6667    0
It is possible to verify that the constant linking the output to the state is \( \frac{1}{\sqrt{\tau}} \). Indeed, the state representation used is:

\[
\begin{align*}
\frac{dx}{dt} &= -\frac{1}{\tau} x(t) + \frac{1}{\sqrt{\tau}} u(t) \\
y(t) &= \frac{1}{\sqrt{\tau}} x(t)
\end{align*}
\] (4.13)

which is obtained by defining the state by \( x(t) = \sqrt{\tau} y(t) \).

**Example 4.3 (Solving of a “predator–prey” system)** Consider the following system of nonlinear equations, known as Lotka–Volterra equations, to be solved:

\[
\begin{align*}
\frac{dx_1}{dt} &= ax_1 - bx_1 x_2 \\
\frac{dx_2}{dt} &= -cx_2 + dx_1 x_2
\end{align*}
\] (4.14)

with \( a, b, c \) and \( d > 0 \). It is possible to simulate the system for \( a = b = c = d = 1 \) as follows:

```matlab
% fixed points: [0;0] and [c/d;a/b] (omeg^2=ac)
a=1; b=1; c=1; d=1;
xinit=[3/2 1.1 1.1 1 1;3 1.3 1.1 2 4];
t0=0; t=[0:0.1:2.35*pi];
for k=1:size(xinit,2)
y0=xinit(:,k);
[t,y] = solve_lv(a,b,c,d,y0);
plot(y(:,1),y(:,2),'b',y0(1),y0(2),'or'), hold on
end
set(gca,'xlim',[0 4.5],'ylim',[0 4.5]), grid on; hold off
```
where the function `solve_lv` is:

```matlab
function [t,y]=solve_lv(a,b,c,d,y0)
tspan=[0:0.1:2.35*pi]';
[t,y] = ode45(@lv,tspan,y0);
% === Lotka-Volterra equation
function ydot=lv(t,y)
ydot=zeros(2,1);
ydot(1)=a*y(1)-b*y(1)*y(2);
ydot(2)=-c*y(2)+d*y(1)*y(2);
end
```

We obtain Figure 4.4 for several initial values.

**Figure 4.4 – Phase trajectories for various initial values**

### 4.2.3 Remarks on the Runge–Kutta methods

Here, again, the problem is to calculate the solution to \( \frac{dy}{dt} = f(t, y) \) by estimation of \( y(kT + T) \) as a function of \( y(kT) \). The Runge–Kutta (RK) methods \( pq \) \([5]\) consist of:

1. Estimating the value of the function \( y(t) \) at intermediary points \( kT + h_1, kT + h_2, \ldots, kT + h_q = (k + 1)T \ (h_0 = 0 \text{ and } h_q = T) \) and the \( h_n \) are not
necessarily distinct), in the range \([kT, (k + 1)T]\), as follows:

\[
y(kT + h_1) = y(kT) + a_{1,0} f(kT, y(kT)) \\
y(kT + h_2) = y(kT) + a_{2,0} f(kT, y(kT)) + a_{2,1} f(kT + h_1, y(kT + h_1)) \\
\vdots \\
y(kT + h_q) = y(kT) + a_{q,0} f(kT, y(kT)) + \cdots + a_{q,q-1} f(kT + h_{q-1}, y(kT + h_{q-1}))
\]

given that \(y(kT + h_q) = y((k + 1)T)\) is what we are seeking to estimate.

2. Ensuring that the \(p\)-order Taylor expansions are verified at all those intermediary points \((kT + h_n)\):

\[
y(kT + h_n) = y(kT) + h_n \dot{y}(kT) + \cdots + \frac{h_n^p}{p!} y^{(p)}(kT) + O(h_n^{p+1})
\]

**RK\(_{22}\) or RK-2 algorithms**

The application to the case \(p = q = 2\) begins by writing:

\[
y(kT + h_1) = y(kT) + a_{1,0} f(kT, y(kT)) = y(kT) + h_1 f(kT, y(kT)) \\
y(kT + h_2) = y(kT) + a_{2,0} f(kT, y(kT)) + a_{2,1} f(kT + h_1, y(kT + h_1)) \\
\quad = y(kT) + a_{2,0} \frac{dy}{dt}(kT) + a_{2,1} \frac{dy}{dt}(kT + h_1) \\
\quad = y((k + 1)T)
\]

The first expression is already in the form of a first-order Taylor expansion. The second should give:

\[
y(kT + h_2) = y(kT) + T \dot{y}(kT) + \frac{T^2}{2} y^{(2)}(kT) + O(T^3)
\]

We have:

\[
\frac{d^2 y}{dt^2} = \frac{df(t, y)}{dt} = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial y} \frac{dy}{dt}
\]

\[
\frac{dy}{dt}(kT + h_1) = \frac{dy}{dt}(kT) + h_1 \frac{d^2 y}{dt^2}(kT) + \frac{h_1^2}{2} \frac{d^3 y}{dt^3}(kT) + \cdots
\]

and:

\[
y((k + 1)T) = y(kT) + a_{2,0} \frac{dy}{dt}(kT) + a_{2,1} \left( \frac{dy}{dt}(kT) + h_1 \frac{d^2 y}{dt^2}(kT) + \cdots \right) \\
\quad = y(kT) + T \frac{dy}{dt}(kT) + \frac{T^2}{2} \frac{d^2 y}{dt^2}(kT) + \frac{T^3}{6} \frac{d^3 y}{dt^3}(kT) + \cdots
\]
By identification we obtain:

\[ a_{2,0} + a_{2,1} = T \quad \text{and} \quad a_{2,1} h_1 = \frac{T^2}{2} \]

\[ \Rightarrow \quad a_{2,1} = \frac{T^2}{2 h_1} \quad \text{and} \quad a_{2,0} = T - \frac{T^2}{2 h_1} \]

This finally gives us:

\[
y(kT + h_1) = y(kT) + h_1 f(kT, y(kT)) \]
\[
y(kT + T) = y(kT) + \left( T - \frac{T^2}{2 h_1} \right) f(kT, y(kT)) \ldots \]
\[
+ \frac{T^2}{2 h_1} f(kT + h_1, y(kT + h_1))
\]

There are three classic versions of the algorithm: improved tangent algorithm for \( h_1 = T/2 \), Euler–Cauchy for \( h_1 = T \) and Heun for \( h_1 = 2T/3 \).

Exercise 4.4 (RK-2 solver and ode15i function) (see p. 203) Returning to the example 4.3:

1. process the differential equation from example 4.4 using Heun’s algorithm, and compare the result to the solution given by ode45.

2. instead of ode45, use ode15i to perform the same process.

\textbf{RK}_{44} \text{ or } \textbf{RK}-4 \text{ algorithms}

Let us begin with an example relating to the widely-used “\( M = 4 \)th-order Runge–Kutta method”. The presentation of the algorithm is slightly different from what we have seen before.

\textbf{Example 4.4 (Fourth-order Runge–Kutta, or RK-4, method)} Consider the scalar case. We can write:

\[
y(kT + T) = y(kT) + \int_0^1 f(kT + uT, y(kT + uT)) Tu_f \]
\[
= y(kT) + T \sum_{1}^{N} \alpha_n f(kT + u_n T, y(kT + u_n T)) \quad (4.15)
\]

with \( \sum_{1}^{N} \alpha_n = 1 \). In certain conditions the error committed by the sum (4.15) is at \( T^{N+1} \).

In the program \texttt{RK4.m}, we solve the equation \( \dot{y}(t) = y^2(t) - y \sin(t) + \cos(t) \) using two fourth-order methods:
1. In the basic method we estimate the derivative at $kT + T/2$ twice. We successively calculate:

- the derivative $k_1$ at point $(kT, y(kT))$, so that $k_1 = f(kT, y(k))$,
- the derivative $k_2$ at point $((kT + T/2), y(k) + k_1T/2)$ using the previous result, so that $k_2 = f((k + 1/2)T, y(k) + k_1T/2)$,
- the derivative $k_3$ at point $((kT + T/2), y(k) + k_2T/2)$ using the previous result, so that $k_3 = f((k + 1/2)T, y(k) + k_2T/2)$,
- the derivative $k_4$ at point $kT + T$ using the previous result, so that $k_4 = f(kT + T, y(k) + k_3T)$,

Then we estimate a weighted-sum derivative of $k_1$, $k_2$, $k_3$ and $k_4$ by $[1]/6$ (in the initial presentation, it is the series $a_{q,0}$, $a_{q,1}$, ..., $a_{q-1,q-1}$), from which we deduce an estimation for $y(k+1)$.

2. In the second so-called “Kutta” method, we calculate:

- the derivative $k_1$ at point $kT, y(kT)$ so that $k_1 = f(kT, y(kT))$,
- the derivative at point $(k + 1/3)T, y((k + 1/3)T)$ using the previous result, so that $k_2 = f((k + 1/3)T, y(k) + k_1T/3)$,
- the term $k_3$ by $f((k + 2/3)T, y(k) - k_1T/3 + k_2T)$,
- the term $k_4$ by $f((k + 1)T, y(k) + k_1T - k_2T + k_3T)$

with a weighting function $[1 3 3 1]/8$, from which we deduce an estimation for $y(k+1)$.

```matlab
function RK4()
y0=0; h=.1; t=[0:h:2*pi]; Lt=length(t);
[t,y]=ode45(@fexple,t,y0); y1=y;
clf, subplot(211), plot(t,y1); grid on, hold on
%===== solution standard RK4
hs2=h/2; hw=[1;2;2;1]/6;
y(1)=y0;
for k=1:Lt-1
  k1=fexple(t(k),y(k));
  k2=fexple(t(k)+hs2,y(k)+hs2*k1);
  k3=fexple(t(k)+hs2,y(k)+hs2*k2);
  k4=fexple(t(k)+h,y(k)+h*k3);
  % weighting
  y(k+1)=y(k)+[k1,k2,k3,k4]*hw*h;
end
plot(t,y,'ro'); y2=y;
%===== solution RK4 - 3/8 rule
hw=[1;3;3;1]/8;
y(1)=y0; hs3=h/3; dhs3=2*hs3;
```
for \( k = 1: L_t - 1 \)
\[
\begin{align*}
  k_1 &= \text{fexple}(t(k), y(k)); \\
  k_2 &= \text{fexple}(t(k) + hs3, y(k) + k_1*hs3); \\
  k_3 &= \text{fexple}(t(k) + dhs3, y(k) - k_1*hs3 + k_2*h); \\
  k_4 &= \text{fexple}(t(k) + h, y(k) + k_1*h - k_2*h + k_3*h); \\
\end{align*}
\]
% weighting
\[
y(k+1) = y(k) + [k_1, k_2, k_3, k_4]*hw*h;
\]
end
plot(t, y, 'xk'), legend('ode45', 'standard RK4', 'RK4 3/8')
subplot(212), plot(t, y1'-y2', 'b', t, y1'-y', 'r');
legend('Diff. ode45-RK4 standard', 'Diff. ode45-RK4 3/8')
grid on
end
\]
function ydot=fexple(t,y)
ydot=y^2-y*sin(t)+cos(t);
end

Modification of the order of the error and estimation of the derivative

Here, we consider a quantity \( A \) whose approximate expression \( A_{n-1}(\varepsilon) \) of order \( (n - 1) \) is given by:
\[
A = A_{n-1}(\varepsilon) + k_0 \varepsilon^n + k_1 \varepsilon^{n+1} + \cdots
\]
\[
= A_{n-1}(\varepsilon) + k_0 \varepsilon^n + \mathcal{O}(\varepsilon^{n+1}) \tag{4.16}
\]

The Richardson method—also known as extrapolation—enables us to obtain an approximate \( n \)-order expression which we will denote as \( A_n(\varepsilon) \) or \( R_n(\varepsilon, r) \) [34].

Consider the parameter \( r \); we can write (4.16) as follows:
\[
A = A_{n-1}(\varepsilon/r) + k_0 \times (\varepsilon/r)^n + \mathcal{O}(\varepsilon^{n+1}) \tag{4.17}
\]
\[
(r^n \times ((4.17)) - ((4.16))) \text{ gives us:}
\]
\[
(r^n - 1)A = r^n A_{n-1}(\varepsilon/r) - A_{n-1}(\varepsilon) + \mathcal{O}(\varepsilon^{n+1}) \tag{4.18}
\]
so:
\[
A = \frac{r^n A_{n-1}(\varepsilon/r) - A_{n-1}(\varepsilon)}{r^n - 1} + \mathcal{O}(\varepsilon^{n+1}) \tag{4.19}
\]

The expression of \( A \) uses the approximation \( R_n(\varepsilon, r) \) of order \( n \):
\[
R_n(\varepsilon, r) = \frac{r^n A_{n-1}(\varepsilon/r) - A_{n-1}(\varepsilon)}{r^n - 1} \tag{4.20}
\]
at the cost of the calculation of \( A_{n-1}(\varepsilon/r) \).

\( ^{(1)} \)http://www.math.ubc.ca/~feldman/m256/richard.pdf
Example 4.5 (Estimation of the derivative) The derivative at a point $t_0$ can be estimated on the basis of the Taylor expansion:

$$f(t_0 + \varepsilon) = f(t_0) + \varepsilon \frac{d}{dt}f(t_0) + \frac{\varepsilon^2}{2} \frac{d^2}{dt^2}f(t_0) + O(\varepsilon^3)$$

$$\Rightarrow \frac{d}{dt}f(t_0) = \frac{f(t_0 + \varepsilon) - f(t_0)}{\varepsilon} - \frac{\varepsilon}{2} \frac{d}{dt}f(t_0) + O(\varepsilon^2)$$

The estimation $A_0(\varepsilon) = \frac{f(t_0 + \varepsilon) - f(t_0)}{\varepsilon}$ corresponds to the Euler approximation. Take $r = 2$. In this case:

$$R_1(\varepsilon, 2) = 2A_0(\varepsilon/2) - A_0(\varepsilon) \quad (= A_2(\varepsilon))$$  
$$= 2\frac{f(t_0 + \varepsilon/2) - f(t_0)}{\varepsilon/2} - \frac{f(t_0 + \varepsilon) - f(t_0)}{\varepsilon}$$  
$$= 4\frac{f(t_0 + \varepsilon/2)}{\varepsilon} - 3\frac{f(t_0)}{\varepsilon} - \frac{f(t_0 + \varepsilon)}{\varepsilon}$$

In the following order:

$$R_2(\varepsilon, 2) = \frac{1}{3} \left[ 4A_1(\varepsilon/2) - A_1(\varepsilon) \right]$$  
$$= \frac{1}{3} (8A_0(\varepsilon/4) - 6A_0(\varepsilon/2) + A_0(\varepsilon))$$

The approximate value of the derivative is then obtained by:

$$R_2(\varepsilon, 2) = \frac{8}{3} A_0(\varepsilon/4) - 2A_0(\varepsilon/2) + \frac{1}{3} A_0(\varepsilon)$$  
$$= \frac{8}{3} \frac{f(t_0 + \varepsilon/4) - f(t_0)}{\varepsilon/4} - 2\frac{f(t_0 + \varepsilon/2) - f(t_0)}{\varepsilon/2} + \frac{f(t_0 + \varepsilon) - f(t_0)}{3\varepsilon}$$  
$$= -\frac{7}{\varepsilon} f(t_0) + \frac{32}{3\varepsilon} f(t_0 + \varepsilon/4) - \frac{4}{\varepsilon} f(t_0 + \varepsilon/2) + \frac{1}{3\varepsilon} f(t_0 + \varepsilon)$$

$$R_3(\varepsilon, 2) = \frac{1}{7} \left[ 8A_2(\varepsilon/2) - A_2(\varepsilon) \right]$$  
$$= \frac{1}{21} (64A_0(\varepsilon/8) - 56A_0(\varepsilon/4) + 14A_0(\varepsilon/2) - A_0(\varepsilon))$$

We can summarize the above in the table:

<table>
<thead>
<tr>
<th>$(r = 2)$</th>
<th>$\frac{\varepsilon}{11}$</th>
<th>$\frac{\varepsilon}{9}$</th>
<th>$\frac{\varepsilon}{7}$</th>
<th>$\frac{\varepsilon}{5}$</th>
<th>$\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n = 1$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>-1</td>
</tr>
<tr>
<td>$n = 2$</td>
<td>0</td>
<td>0</td>
<td>$\frac{8}{3}$</td>
<td>-2</td>
<td>$\frac{1}{3}$</td>
</tr>
<tr>
<td>$n = 3$</td>
<td>0</td>
<td>$\frac{64}{21}$</td>
<td>$-\frac{8}{3}$</td>
<td>$\frac{2}{3}$</td>
<td>$-\frac{1}{21}$</td>
</tr>
<tr>
<td>$n = 4$</td>
<td>$\frac{1024}{315}$</td>
<td>$-\frac{64}{21}$</td>
<td>$\frac{8}{9}$</td>
<td>$-\frac{2}{21}$</td>
<td>$\frac{1}{315}$</td>
</tr>
</tbody>
</table>
and for the derivative:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>$\frac{1}{16}$</th>
<th>$\frac{1}{8}$</th>
<th>$\frac{1}{4}$</th>
<th>$\frac{1}{2}$</th>
<th>$\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n=0$</td>
<td>$-\frac{1}{\varepsilon}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$\frac{1}{\varepsilon}$</td>
</tr>
<tr>
<td>$n=1$</td>
<td>$-\frac{2}{\varepsilon}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$\frac{1}{4}$</td>
<td>$-\frac{1}{\varepsilon}$</td>
</tr>
<tr>
<td>$n=2$</td>
<td>$-\frac{7}{\varepsilon}$</td>
<td>0</td>
<td>0</td>
<td>$\frac{32}{3\varepsilon}$</td>
<td>$-\frac{4}{\varepsilon}$</td>
<td>$\frac{1}{3\varepsilon}$</td>
</tr>
<tr>
<td>$n=3$</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The calculation of the coefficients for the derivative `derivestimc` and construction `calcrn` and `derivestimc` of the corresponding tables are:

```matlab
%===== estimderiv.m
n=2; h=1; r=2;
mcoff=derivestimc(n,h,r);
disp(mcoff)

function [Rn,ve]=calcrn(n,h,r)
%!==================================!
%! SYNOPSIS [Rn,ve]=CALCRN(n,h,r) !
%! n = expansion order !
%! h = step value !
%! r = parameter used to divide h !
%! Rn = coefficients array !
%! ve = vector [h/r^n ... h/r h] !
%!==================================!
Rn=zeros(n,n+1);
vr=ones(1,n+1) ./ r.^(n:-1:0); ve=h * vr;
Rn(1,n)=r; Rn(1,n+1)=-1;
for k=2:n
    Rn(k,:)=(r^k * [Rn(k-1,2:end),0]-Rn(k-1,:))/(r^k-1);
end

function [mcoff,ve]=derivestimc(n,h,r)
%!================================================!
%! SYNOPSIS [mcoff,ve]=DERIVESTIMC(n,h,r) !
%! n = expansion order !
%! h = step value !
%! r = parameter used to divide h !
%! mcoff = coefficients for derivative estimation !
%! mcoff(1)*f(nT) + mcoff(2)*f(nT+h/r^n)+... !
%! ve = vector [h/r^n ... h/r h] !
%!================================================!
[Rn,ve]=calcrn(n,h,r);
Re=Rn(n,:) ./ ve;
tt=toeplitz(zeros(n+1,1),[0 1 zeros(1,n)]);
tt(:,:,1)=-1; mm=(Re.' * ones(1,n+2)) .* tt;
mcoff=sum(mm,1);
```
4.3 Systems of equations and zero-seeking

4.3.1 Zeros of a function using the Newton method

The principle of the method is illustrated by Figure 4.5. Given the graph $G$ of $y = f(x)$, we construct a series of values $x_n$ in the following way: we consider the tangent $(\Delta_n)$ in $x_n$ at $G$ and its intersection with the Ox axis which defines $x_{n+1}$. This process is repeated iteratively until $|x_{n+1} - x_n| < \varepsilon$.

![Figure 4.5 – Diagram of recurrence in the Newton method, moving from the point $M_n$ to the point $M_{n+1}$](image)

The convergence of the method depends on several conditions related to the function $h(x) = x - f(x)/f'(x)$ and on the choice of the initial value $x_0$. We choose to take that value in the vicinity $(V)$ containing a solution to the equation, ensuring that the function $y$ is doubly derivable and with non-null first and second derivatives.

Exercise 4.5 (Newton’s method) (see p. 205)

Here we consider the case of the polynomials $y = f(x)$. Given $x = x_n$, we use the notation $y_n = f(x_n)$ and $y_n'$ for the derivative at $x = x_n$:

1. give the recurrence relation linking $x_{n+1}$ to $x_n$;

2. we take a polynomial $\text{myp}$ by its coefficients:

   $\text{myp}=[1,(-4-\text{sqrt}(2)),(3+4\times\text{sqrt}(2)),-3\times\text{sqrt}(2)];$

Write a program calculating the roots of that polynomial using the Newton method. Note that the polynomial division is performed by the function $\text{deconv}$ and the derivation by $\text{polyder}$. If the derivative becomes null during the course of the execution of the program, we will simply take a different initial condition.
4.3.2 Roots of a polynomial with the Newton–Raphson method

The Newton–Raphson method applies to zero-seeking for \( F(z) = 0 \). We will use it here to calculate the roots of a polynomial \( P(z) \). The principle is as follows: we take an initial value \( z = x_0 + jy_0 \). Then we vary the value of \( z \) by \( \Delta z = \Delta x + j\Delta y \), \( \Delta x \) and \( \Delta y \), given by Newton’s equations, so that \( P(z) \) approaches the origin (see Figure 4.6). The stop test is given by \( |\Delta x| + |\Delta y| < \varepsilon \).

![Figure 4.6 – Principle of the method](image)

Thus, consider a polynomial \( P(z) \) of degree \( D \):

\[
P(z) = \sum_{n=0}^{D} a(n)z^n \quad (4.21)
\]

The steps involved in the method are as follows:

1. During the calculation, \( z^n \) is evaluated by recurrence. We set \( z^n = (x + jy)^n = X_n + jY_n \) and the \( X_n \) and \( Y_n \) are obtained by the recurrences:

\[
X_n = xX_{n-1} - yY_{n-1} \quad \text{and} \quad Y_n = xY_{n-1} + yX_{n-1} \quad (4.22)
\]

2. We set \( P(z) = U + jV \). We have:

\[
P(z) = \sum_{n=0}^{D} a(n)z^n = \sum_{n=0}^{D} a(n)X_n + j\sum_{n=0}^{D} a(n)Y_n \quad (4.23)
\]

from which we deduce:

\[
\frac{\partial U}{\partial x} = \frac{\partial V}{\partial y} = \sum_{n=1}^{D} na(n)X_{n-1} \quad (4.24)
\]

\[
\frac{\partial V}{\partial x} = -\frac{\partial U}{\partial y} = \sum_{n=1}^{D} na(n)Y_{n-1} \quad (4.25)
\]
3. Newton’s equations are obtained with the proviso that we are seeking to bring the point \((U_k, V_k)\) closer to the origin (Figure 4.6):

\[
\begin{align*}
\Delta U &= -U = \frac{\partial U}{\partial x} \Delta x + \frac{\partial U}{\partial y} \Delta y \\
\Delta V &= -V = \frac{\partial V}{\partial x} \Delta x + \frac{\partial V}{\partial y} \Delta y
\end{align*}
\]

\[
\Rightarrow \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = -\begin{bmatrix} \frac{\partial U}{\partial x} & \frac{\partial U}{\partial y} \\ \frac{\partial V}{\partial x} & \frac{\partial V}{\partial y} \end{bmatrix}^{-1} \begin{bmatrix} U \\ V \end{bmatrix} \tag{4.26}
\]

and from expressions (4.24) and (4.25):

\[
\Delta x = -\frac{U \frac{\partial U}{\partial x} + V \frac{\partial V}{\partial x}}{\left(\frac{\partial U}{\partial x}\right)^2 + \left(\frac{\partial V}{\partial x}\right)^2} \quad \text{and} \quad \Delta y = \frac{U \frac{\partial V}{\partial x} - V \frac{\partial U}{\partial x}}{\left(\frac{\partial U}{\partial x}\right)^2 + \left(\frac{\partial V}{\partial x}\right)^2} \tag{4.27}
\]

The expression (4.26) extends to all equations, linear or nonlinear, as we will see in section 6.5.

**Exercise 4.6 (The Newton–Raphson method)** (see p. 206)

We apply the Newton–Raphson method to the calculation of the roots of a polynomial:

1. Write a function to calculate a root \([xc, yc, cErr] = \text{CalcXY}(pol, x0, y0, nblps)\) where \(pol\) is the polynomial given in the form of a vector of its coefficients (in order of decreasing power), \((x0, y0)\) is the starting point, \(\text{teps}\) the \(\varepsilon\) the end-of-loop test, \(\text{nblps}\) the maximum acceptable number of loops, \((xc, yc)\) the solution found and \(cErr\) an error code (e.g. denoting an “excessive” number of iterations);

2. Write a function to calculate all the roots of a polynomial.

**4.3.3 Systems of nonlinear equations**

The function \(\text{fzero}\), which is available in the basic version of MATLAB®, can be used to solve the equation \(\mathcal{F}(x) = 0\) when \(x\) is scalar. \(\text{fsolve}\) from the\(\text{optim toolbox}\) is able to solve \(\mathcal{F}(x) = 0\) in the multivariable case.

The call syntax for \(\text{fzero}\) is:

\[
[x, fval, exitflag, output] = \text{fzero}(@myfunc, x0, options)
\]

**Example 4.6 (Solving of nonlinear equations)**

Consider the following equation to be solved:

\[
x + 1 + \tan(x) = 0 \tag{4.28}
\]

Type:
%===== explefzero.m
x0=-1; % init value
%==== using an anonymous function
[x,fval,exitflag,output]=fzero(@(x)x+1+atan(x),x0)

The execution of the program explefzero gives:

>> explefzero

x =
   -0.5203
fval =
   5.5511e-17
exitflag =
   1
output =
   intervaliterations: 10
   iterations: 6
   funcCount: 26
   algorithm: 'bisection, interpolation'
   message: 'Zero found in the interval [-0.36, -1.45255]'

Instead of giving an initial value, it is possible to define an interval within which a solution is sought. Thus, in the program explefzero2, we seek a solution between $\pi/4$ and $\pi/2$:

%===== explefzero2.m
x0=[pi/4 pi/2]; % interval
[x,fval,exitflag,output]=fzero(@(x)x+1-tan(x),x0)

4.4 Interpolation

The word interpolation is used to denote many different types of operations. We sometimes also speak of curve fitting. In signal-processing, interpolation is generally performed by zero-insertion, followed by low-pass filtering. The series obtained does not pass through the same points as the original series $S = \{ \ldots (x_0, f(x_0)), (x_1, f(x_1)), \ldots \}$. However, the spectra of the trajectories are practically identical. There are a wide range of other methods. The most conventional techniques use polynomials: Lagrange basis polynomials, Newton’s equations, cubic spline interpolation, etc. In this case, the trajectory obtained passes through the points of $S$. Functions other than polynomials can also be used: trigonometric functions, complex exponentials, Gaussians, etc.

Here, we are going to examine a method using a rational fraction written in the form of a continued fraction.
4.4.1 Thiele’s interpolation

Principle

Knowing a set of $N + 1$ points of a trajectory given in the form $\{ (x_0, f(x_0)), (x_1, f(x_1)), \ldots, (x_N, f(x_N)) \}$, we seek to approximate $f(x)$ as closely as possible. We write $\mathcal{S} = \{x_0, x_1, \ldots, x_N\}$. The so-called continued fractions decomposition or interpolation by rational interpolants proposed by Thiele [37] is of the form:

$$f(x) = f(x_0) + \frac{x - x_0}{\rho_1(x_0, x_1)} + \frac{x - x_1}{\rho_2(x_0, x_1, x_2)} + \frac{x - x_2}{\rho_3(x_0, x_1, x_2, x_3)} + \ldots \tag{4.29}$$

We use the notation $\rho_0(x) = f(x) = y$ and $\rho_n(x_i, \ldots, x_{i+n}) = \rho_n(x_{i:i+n})$ where $\rho_n(.) = 0$ if $n < 0$. We have the relation:

$$\rho_n(x_{0:n}) = \frac{x_{n-1} - x_n}{\rho_{n-1}(x_{0:n-1}) - \rho_{n-1}(x_{0:n-2}, x_n)} \tag{4.30}$$

and the fraction is constructed by the recurrence relation:

$$\begin{align*}
F_0(x) &= f(x) \\
F_n(x) &= \rho_{n-1}(x_{0:n-1}) + \frac{x - x_{n-1}}{F_{n+1}(x)} \\
F_N(x) &= 1
\end{align*} \tag{4.31}$$

Thus, we have:

$$\begin{align*}
F_0(x) &= \rho_0(x_0), \\
F_1(x) &= \rho_0(x_0) + \frac{x - x_0}{F_2(x)}, \\
F_2(x) &= \rho_1(x_0, x_1) + \frac{x - x_1}{F_3(x)}, \\
F_3(x) &= \rho_2(x_0, x_1, x_2) + \frac{x - x_2}{F_4(x)} \ldots
\end{align*}$$

The first values of the $\rho_n(.)$ are:

$$\begin{align*}
\rho_1(x_0, x_1) &= \frac{x_0 - x_1}{y_0 - y_1}, \\
\rho_1(x_0, x_2) &= \frac{x_0 - x_2}{y_0 - y_2}, \\
\rho_2(x_0, x_1, x_2) &= \frac{x_1 - x_2}{\rho_1(x_0, x_1) - \rho_1(x_0, x_2)}
\end{align*}$$
Indications regarding the method

Consider the rational fraction:

\[
R^{(M,N)}(x) = \frac{a_0 + a_1 x + \cdots + a_M x^M}{b_0 + b_1 x + \cdots + b_N x^N} = \frac{p_M(x)}{q_N(x)} \quad (4.32)
\]

We take a series of points \{ (x_0, y_0), (x_1, y_1), \ldots, (x_{2N}, y_{2N}) \}. The problem at hand is that of finding \( R^{(N,N)}(x) \) such that:

\[
R^{(N,N)}(x_k) = \frac{p_N(x_k)}{q_N(x_k)} = y_k \quad \text{for } k = 0, \ldots, 2N \quad (4.33)
\]

We consider the first point in the series, and we have:

\[
\frac{p_N(x_0)}{q_N(x_0)} = y_0
\]

and we write \( p_N(x)/q_N(x) \) in the form:

\[
\frac{p_N(x)}{q_N(x)} = y_0 + \frac{(x-x_0)}{\rho_1(x_0,x)} \frac{p_{N-1}(x)}{q_{N-1}(x)} = y_0 + \frac{x-x_0}{\rho_1(x_0,x)}
\]

\( \rho_1(x_0,x) \) is called the inverse difference. If we attribute to \( x \) the successive values \( x_k, k = 1 \ldots 2N \), we can write:

\[
\frac{x_k-x_0}{y_k-y_0} = \frac{q_N(x_k)}{p_{N-1}(x_k)} = \rho_1(x_0,x_k)
\]

We can continue with \( x_1 \):

\[
\frac{q_N(x_1)}{p_{N-1}(x_1)} = \rho_1(x_0,x_1) \Rightarrow \frac{q_N(x)}{p_{N-1}(x)} = \rho_1(x_0,x_1) + (x-x_1) \frac{q_{N-1}(x)}{p_{N-1}(x)} \quad (4.34)
\]

If we substitute \( x = x_k, k = 2 \ldots 2N \) back into (4.34):

\[
\frac{q_N(x_k)}{p_{N-1}(x_k)} = \rho_1(x_0,x_k) = \rho_1(x_0,x_1) + (x_k-x_1) \frac{q_{N-1}(x_k)}{p_{N-1}(x_k)} \quad (4.35)
\]

\[
\Rightarrow \frac{p_{N-1}(x_k)}{q_{N-1}(x_k)} = \frac{x_k-x_1}{\rho_1(x_0,x_k) - \rho_1(x_0,x_1)} = \rho_2(x_0,x_1,x_k)
\]

We continue, noting that:

\[
\frac{p_{N-1}(x_2)}{q_{N-1}(x_2)} - \rho_2(x_0,x_1,x_2) = 0 \Rightarrow \frac{p_{N-1}(x)}{q_{N-1}(x)} = \rho_2(x_0,x_1,x_2) + (x-x_2) \frac{p_{N-2}(x)}{q_{N-1}(x)}
\]

which enables us to obtain the construction scheme (4.29). It should be noted, however, that the values of \( \rho_n \) must not reach zero. If they do, we say that the associated points are unattainable and that interpolation is not possible, taking the series of points \( S \) in the order in which it is given. Werner's algorithm [42] provides a solution which, in certain cases, is able to yield a permutation to help calculate the fraction.
Programming

For programming, we organize the calculation of the $\rho_n^{(i)}$ values using a table:

<table>
<thead>
<tr>
<th>$x$</th>
<th>$\rho_0(.)$</th>
<th>$\rho_1(.)$</th>
<th>$\rho_2(.)$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_0$</td>
<td>$y_0$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_1$</td>
<td>$y_1$</td>
<td>$\rho_1(x_0, x_1)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_2$</td>
<td>$y_2$</td>
<td>$\rho_1(x_0, x_2)$</td>
<td>$\rho_2(x_0; 2)$</td>
<td></td>
</tr>
<tr>
<td>$x_3$</td>
<td>$y_3$</td>
<td>$\rho_1(x_0, x_3)$</td>
<td>$\rho_2(x_0; 3)$</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$x_N$</td>
<td>$y_N$</td>
<td>$\rho_1(x_0, x_N)$</td>
<td>$\rho_2(x_0; N)$</td>
<td>...</td>
</tr>
</tbody>
</table>

then we calculate the $F_k$ values using the relation (4.31).

**Example 4.7 (Thiele decomposition)** We consider the points $(0, 0)$, $(1, 1/2)$, $(2, 2)$, $(3, 1)$. The computed table gives us:

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$\rho(x_0; x_k)$</th>
<th>$\rho(x_0, x_1; x_k)$</th>
<th>$\rho(x_0, x_1, x_2; x_k)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$\phi$</td>
<td>$\phi$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>1</td>
<td>1/2</td>
<td>2</td>
<td>$\phi$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>$-1$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1/3</td>
<td></td>
</tr>
</tbody>
</table>

and the approximation function:

$$f(x) = \frac{2x}{4 + (x - 1)(3x - 8)}$$

![Figure 4.7 – Interpolation by “rational interpolants”](image)
clear all
xk=(0:3)'; yk=[0;1/2;2;1]; N=length(xk);

%== calculating of rho constants
mt=initthiele(xk,yk); dmt=diag(mt,1); disp(mt)

%== verification
[yk,(2*xk) ./ (4+(xk-1).*(3*xk-8))]
x=[0:.05:3]; fx=(2*x) ./ (4+(x-1).*(3*x-8));

%== calculating of interpolant function
Nx=length(x); ye=zeros(Nx,1);
for k=1:Nx
    ye(k)=calcthiele(x(k),xk,dmt);
end
plot(x,fx,'-r',x,ye,'.y',xk,yk,'xk'), grid
axis([-0.5,3.5,-0.5,2.5])

%===== testth01.m

function mt=initthiele(xk,yk)
%!======================================================!
%! Calculating rho functions for thiele's interpolation !
%! (xk,yk) are the original points to be !
%! interpolated by a continued fraction !
%!======================================================!
N=length(xk); mt=ones(N,N+1)*NaN;
mt(:,1)=xk; mt(:,2)=yk;
mt(2:N,3)=(mt(1,1)-mt(2:N,1))./(mt(1,2)-mt(2:N,2));
for ll=3:N
    for cc=4:ll+1
        mt(ll,cc)=...
        (mt(ll,1)-mt(cc-2,1))/(mt(ll,cc-1)-mt(cc-2,cc-1));
    end
end

function y=calcthiele(x,xk,dmt)
%!==================================================================!
%! Calculating of the continued fraction for x !
%! x = value for calculating of f(x) !
%! xk = vector of the original abscissa !
%! dmt = rho(x0,x1,...xN) constants !
%!==================================================================!
N=length(dmt); y=1;
for k=N:-1:1, y=dmt(k)+(x-xk(k))/y; end
end

4.4.2 Another decomposition in continuous fractions

Look at the function \( f(x) \) and the series \( \mathcal{S} = \{ x_1, x_2, \ldots, x_N \} \) which we write as \( x_1:N \). We suppose that the \( x_k \) values are all distinct, that \( f(x) \) does not cancel within \( \mathcal{S} \) and that it is derivable over \( \mathcal{S} \). We use the notation \( \rho_0(x) = f(x) \) and write [25]:

\[
f(x) = \frac{\rho_0(x_1)}{1 - (x - x_1)\rho_1(x_1, x)} \Rightarrow \rho_1(x_1, x) = \frac{1}{\rho_0(x)} \frac{\rho_0(x) - \rho_0(x_1)}{x - x_1}
\]
We note that:
\[
\rho_1(x_1, x_1) = \frac{1}{f(x_1)} \left. \frac{df(x)}{dx} \right|_{x=x_1} = \frac{1}{\rho_0(x_1)} \left. \frac{d\rho_0(x)}{dx} \right|_{x=x_1}
\]

We then posit:
\[
\rho_1(x_1, x) = \frac{\rho_1(x_1, x_2)}{1 - (x - x_2)\rho_2(x_1, x, x_2)}
\]
\[
\Rightarrow \rho_2(x_1, x_2) = \frac{\rho_1(x_1, x) - \rho_1(x_1, x_2)}{x - x_2} \frac{1}{\rho_1(x_1, x)}
\]
noting that:
\[
\rho_2(x_1, x_1) = \frac{\rho_1(x_1, x_1) - \rho_1(x_1, x_2)}{x_1 - x_2} \frac{1}{\rho_1(x_1, x)}
\]
\[
\rho_2(x_1, x_2) = \left. \frac{d\rho_1(x_1, x)}{dx} \right|_{x=x_2} \frac{1}{\rho_1(x_1, x)}
\]

More generally speaking:
\[
\rho_{n-1}(x_{1:n-1}, x) = \frac{\rho_{n-1}(x_{1:n-1}, x_{n})}{1 - (x - x_{n})\rho_{n}(x_{1:n-1}, x)}
\]
\[
\Rightarrow \rho_n(x_{1:n}, x) = \frac{\rho_{n-1}(x_{1:n-1}, x) - \rho_{n-1}(x_{1:n-1}, x_{n})}{x - x_{n}} \frac{1}{\rho_{n-1}(x_{1:n-1}, x)}
\]

By way of example, let us consider the expression of \( f(x) \) for \( N \) points:
\[
f(x) = \frac{\rho_0(x_1)}{1 - (x - x_1)} \frac{\rho_1(x_1, x_2)}{1 - (x - x_2)} \frac{\rho_2(x_1, x_2, x_3)}{1 - (x - x_3)} \cdots \frac{\rho_{N-1}(x_{1:N-1}, x_N)}{1 - (x - x_{N-1})\rho_{N-1}(x_{1:N-1}, x_N)}
\]

To obtain \( f(x) \), we replace \( \rho_{N-1}(x_{1:N-1}, x) \) with the quantity \( \rho_{N-1}(x_{1:N-1}, x_{N+1}) \). \( \rho_N \) multiplied by \( (x - x_{N}) \) then gives us the relative error committed regarding \( \rho_{N-1} \) during this approximation.

### 4.4.3 Natural cubic splines

We consider a series of points \( \{P_k\}, k \in [0, N - 1] \), known as control points, with affix \( p_k \in \mathbb{C} \), and the trajectory “segments” \( (\mathcal{S}_k) \) between two successive points \( P_k \) and \( P_{k+1} \) (Figure 4.8). \( p_k(t) \in \mathbb{C} \) is the current point in the segment \( (P_k - P_{k+1}) \), and we use the notation \( \dot{p}_k(t) \) and \( \ddot{p}_k(t) \), respectively, for the first and second derivatives of \( p_k(t) \).
We suppose that \( \ddot{p}_k(t) \) can be written:

\[
\ddot{p}_k(t) = \ddot{p}_k(0)(1 - t) + \ddot{p}_k(1)t
\]

where \( t \) is a parameter \( \in [0, 1] \). From (4.36), we deduce \( \dot{p}_k(t) \) and \( p_k(t) \).

\[
\begin{align*}
\dot{p}_k(t) &= \ddot{p}_k(0) \left( t - \frac{t^2}{2} \right) + \ddot{p}_k(1) \frac{t^2}{2} + \alpha \\
p_k(t) &= \ddot{p}_k(0) \left( \frac{t^2}{2} - \frac{t^3}{6} \right) + \ddot{p}_k(1) \frac{t^3}{6} + \alpha t + \beta
\end{align*}
\]

We first consider the natural cubic splines for which the second derivatives at the endpoints \( P_0 \) and \( P_{N-1} \) are null. We also suppose that the second derivative remains continuous at each of the points \( P_k \). Thus, we can write these conditions in the form:

\[
\begin{align*}
\ddot{p}_0(0) &= 0, \\
\ddot{p}_{N-2}(1) &= 0 \\
\ddot{p}_k(1) &= \ddot{p}_{k+1}(0) \text{ for } k = 0 \ldots N - 3
\end{align*}
\]

Using the fact that \( p_k(0) = p_k \) and \( p_k(1) = p_{k+1} \), we deduce \( \alpha \) and \( \beta \) and then the expression for the trajectory \( p_k(t) \):

\[
p_k(t) = \ddot{p}_k(0) \left( \frac{t^2}{2} - \frac{t^3}{6} - \frac{t}{3} \right) + \ddot{p}_k(1) \left( \frac{t^3}{6} - \frac{t}{6} \right) + (p_{k+1} - p_k)t + p_k
\]

\[
= \left( \ddot{p}_{k+1}(0) - \ddot{p}_k(0) \right) \frac{t^3}{6} + \ddot{p}_k(0) \frac{t^2}{2} + \left( p_{k+1} - p_k - \ddot{p}_k(0) \frac{1}{3} - \ddot{p}_k(1) \frac{1}{6} \right) t + p_k
\]

The continuity conditions (4.38) involve \( N - 2 \) relations for \( 2(N - 2) \) unknowns \( \ddot{p}_k(0) \) and \( \ddot{p}_k(1) \). In order to be able to solve this system, we will express the continuity of the derivative at points \( P_k \).

\[
\dot{p}_k(t) = \ddot{p}_k(0) \left( t - \frac{t^2}{2} \right) + \ddot{p}_k(1) \frac{t^2}{2} + p_{k+1} - p_k - \ddot{p}_k(0) \frac{1}{3} - \ddot{p}_k(1) \frac{1}{6}
\]
which gives us:

\[
\begin{align*}
\dot{p}_k(0) &= p_{k+1} - p_k - \frac{\ddot{p}_k(0)}{3} - \frac{\ddot{p}_k(1)}{6} \\
\dot{p}_k(1) &= p_{k+1} - p_k + \frac{\ddot{p}_k(0)}{6} + \frac{\ddot{p}_k(1)}{3}
\end{align*}
\] (4.41)

The conditions \(\dot{p}_k(1) = \dot{p}_{k+1}(0)\) with the relations (4.38) give:

\[
\frac{\ddot{p}_{k+2}(0)}{6} + 2\frac{\ddot{p}_{k+1}(0)}{3} + \frac{\ddot{p}_k(0)}{6} = p_{k+2} - 2p_{k+1} + p_k
\] (4.42)

All the conditions expressed above can be written in the form (4.43):

\[
\begin{align*}
\ddot{p}_0(0) &= 0 \\
\ddot{p}_{N-1}(0) &= 0 \\
\frac{\ddot{p}_2(0)}{6} + 2\frac{\ddot{p}_1(0)}{3} &= p_2 - 2p_1 + p_0 \\
\frac{\ddot{p}_3(0)}{6} + 2\frac{\ddot{p}_2(0)}{3} + \frac{\ddot{p}_1(0)}{6} &= p_3 - 2p_2 + p_1 \\
&\vdots \\
\frac{2\ddot{p}_{N-2}(0)}{3} + \ddot{p}_{N-3}(0) &= p_{N-1} - 2p_{N-2} + p_{N-3}
\end{align*}
\] (4.43)

Solving this tridiagonal system gives us the second derivatives, thus enabling us to go on to construct each of the \(S_k\) using (4.39).

\[
\begin{bmatrix}
\frac{2}{3} & 1/6 & 0 & 0 & \cdots & 0 \\
1/6 & \frac{2}{3} & 1/6 & 0 & \cdots & 0 \\
0 & \ddots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \ddots & 0 \\
\vdots & 1/6 & 2/3 & 1/6 & \frac{2}{3} & \frac{1}{6} \\
0 & \cdots & 0 & 1/6 & 2/3 & \frac{2}{3} \\
\end{bmatrix}
\begin{bmatrix}
\ddot{p}_1(0) \\
\ddot{p}_2(0) \\
\vdots \\
\ddot{p}_{N-3}(0) \\
\ddot{p}_{N-2}(0)
\end{bmatrix}
= \begin{bmatrix}
p_2 - 2p_1 + p_0 \\
p_3 - 2p_2 + p_1 \\
\vdots \\
p_{N-1} - 2p_{N-2} + p_{N-3}
\end{bmatrix}
\] (4.44)

We are led to the solution of 4.44, a linear system \(Ax = b\) of \(N - 2\) equations with \(N - 2\) unknowns, wherein the matrix \(A\) is tridiagonal symmetrical and positive definite:

\[
A = \begin{bmatrix}
4 & 1 & 1 & \cdots & 1 \\
1 & 4 & 1 & \cdots & 1 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & 4 & 1 & \cdots & 1 \\
1 & 4 & 1 & \cdots & 1
\end{bmatrix} / 6
\] (4.45)
and $b$ is constructed by
\[
\| b = \text{toeplitz}(\text{P}k(3:N), [\text{P}k(3)\ Pk(2)\ Pk(1)]) *[1;-2;1];
\]

Exercise 4.7 (Plot of the natural cubic spline) (see p. 208):
Remark: we consider a series of $N$ pairs $(x_i, y_i)$ and we note $\mathcal{S}_2[a, b]$ all of the twice differentiable functions defined on $[a, b]$. Thus [19], the function $g$ that minimizes on $\mathcal{S}_2[a, b]$ the quantity:
\[
\sum_{i=1}^{K} (y_i - g(x_i))^2 + \alpha \int |g''(u)|^2 du
\] (4.46)
for $\alpha > 0$ is a natural cubic spline. The criterion to be minimized can be seen as the Lagrangian of a quadratic problem with a constraint on the quadratic fluctuations of the second derivative which can be interpreted as a smoothing constraint (see section (6.3)).

Exercise 4.8 (Plot of natural cubic spline) (see p. 208)
We wish to plot a natural cubic spline passing through the points $A(0,0)$, $B(0,2)$, $C(2,0)$, $D(3,2)$ and $E(1,1)$.

1. Write a function giving the second derivatives on the basis of the control points $A$, $B$, $C$, $D$ and $E$.
2. Write a program to plot the associated graph.
4.5 Solving of linear systems

There are many methods in existence for solving the linear system \( Ax = b \). Many of the algorithms available exploit the structure of the matrix \( A \). In signal-processing, the Levinson algorithm, based on the Toeplitz structure of \( A \), is a well-known example of this. Here, we shall look at iterative methods, which involve constructing an array \( x^{(n)} \) that converges toward the solution \( x \). Obviously, these methods are not limited to the case of linear systems.

The method entails writing \( A = M - N \); then, with the initial value \( x^{(0)} \) being given, we execute the iteration:

\[
x^{(n+1)} = M^{-1}Nx^{(n)} + M^{-1}b
\]

(4.47)

We know that if the array \( x^{(n+1)} \) converges toward a limit \( x^{(\infty)} \), then that limit satisfies the recurrent equation (4.47), and from this we deduce that \( Ax^{(\infty)} = b \).

The convergence of this recurrent equation is linked to the eigenvalues \( \lambda_k \) of the matrix \( M^{-1}N \), whose modulus must never exceed 1:

\[
|\lambda_k| < 1
\]

(4.48)

The largest of the eigenvalues is called the spectral radius \( \rho = \max_k |\lambda_k| \).

Let us write the matrix \( A \) in the form:

\[
A = D - L - U
\]

(4.49)

where \( D \) is diagonal, \( L \) is strictly lower triangular and \( U \) strictly upper triangular. Using the notation \( A = [a_{\ell,c}] \) this is written as follows:

\[
D = [a_{\ell,\ell}], \quad L = -[a_{\ell,c}] \quad \text{with} \quad c < \ell \quad \text{and} \quad U = -[a_{\ell,c}] \quad \text{with} \quad c > \ell
\]

4.5.1 Jacobi method

In the Jacobi method, we set \( M = D \) and \( N = L + U \):

\[
x^{(n+1)} = Jx^{(n)} + D^{-1}b
\]

(4.50)

and hence \( J = D^{-1}(L + U) \).

function [x,rho]=jacobimethod(A,b,x0,stcr0,maxit)
%!====================================================!
%! SYNOPSIS: [x,rho]=JACOBIMETHOD(A,b,x0,stcr0,maxit) !
%! x0 = initial state !
%! stcr0 = |x(n+1)-x(n)| limit !
%! maxit = max number of loops !
%! x = solution of Ax=b !
%! rho = spectral radius !
%!====================================================!
n=size(A,1);
U=-triu(A,1); L=-tril(A,-1); N=U+L;
D=diag(A); Dm1=1./D; Mm1=diag(Dm1);
rho=max(abs(eig(Mm1*N)));
if rho>=1, rho, error('the method is not convergent'); end
x=x0; stcr=1; k=0;
while stcr>stcr0 && k<maxit
    xp=(N*x+b).*Dm1; stcr=max(abs(xp-x));
    x=xp; k=k+1;
end

4.5.2 Relaxation method

In the relaxation method, we set:

\[
M = \frac{D}{\omega} - L \quad \text{and} \quad N = \frac{1-\omega}{\omega}D + U
\]

and hence \[
J = \left(\frac{D}{\omega} - L\right)^{-1}\left(\frac{1-\omega}{\omega}D+U\right).
\]

function [x,rho]=relaxmethod(A,b,x0,w,stcr0,maxit)
%!======================================================================!
%! SYNOPSIS [x,rho]=RELAXMETHOD(A,b,x0,w,stcr0,maxit) !
%! x0 = initial xn !
%! stcr0 = |x(n+1)-x(n)| limit !
%! maxit = max number of loops !
%! w = (1 for Gauss-Seidel) !
%! x = solution of Ax=b !
%! rho = spectral radius !
%!======================================================================!

n=size(A,1);
U=-triu(A,1); L=-tril(A,-1); D=diag(A);
N=(1-w)*diag(D)/w+U; M=diag(D)/w-L;
Mm1=inv(M); rho=max(abs(eig(Mm1*N)));
if rho>=1, rho, error('the method is not convergent'); end
x=zeros(n,1); stcr=1; t1=tic; k=0;
while stcr>stcr0 && k<maxit
    xp=Mm1*(N*x+b); stcr=max(abs(xp-x));
    x=xp; k=k+1;
end

Remarks:

– the Gauss–Seidel method corresponds to the case \( \omega = 1 \) in the relaxation
method, i.e. \( M = D - L \) and \( N = U \):

\[
J = (D - L)^{-1}U
\]  (4.51)
the relaxation method entails, at each iteration, calculating a weighted mean between the result yielded by the Gauss–Seidel method \( x^{(n)} \) and the previous result \( x^{(n-1)} \), so that:

\[
x^{(n)} = \omega x^{(n)} + (1 - \omega)x^{(n-1)}
\] (4.52)

In matrix form, this is written as:

\[
x^{(n)} = \omega(D - L)^{-1} \left( U x^{(n-1)} + b \right) + (1 - \omega)x^{(n-1)}
\]

\[
= (D - L)^{-1} \left[ \omega U + (1 - \omega)(D - L) \right] x^{(n-1)} + \omega(D - L)^{-1} b
\]

**Exercise 4.9 (Relaxation method)** (see p. 209)

Consider equation (4.44) with the list of points \( A(0,0), B(0,2), C(2,0), D(3,2) \) and \( E(1,1) \) from exercise 4.8.

1. The symbol \( \rho_{gs} \) denotes the spectral radius in the Gauss–Seidel method, whilst \( \rho_j \) is the spectral radius in the Jacobi method. Show that \( \rho^2_j = \rho_{gs} \).

2. Demonstrate that the convergence of the relaxation method necessitates that \( 0 \leq \omega < 2 \).

3. Perform a simulation using the function `relaxmethod`, viewing the error and the spectral radius as functions of \( \omega \). Accepting that for any positive-definite, tridiagonal symmetrical matrix \( A \), the parameter \( \omega \in [0, 2] \) which minimizes the spectral radius of the matrix \( R_\omega \) is given by:

\[
\omega = \frac{2}{1 + \sqrt{1 - \rho^2_j}}
\] (4.53)

verify this value on the plots obtained.

### 4.5.3 Cholesky factorization

Let \( A \) be a Hermitian matrix. We seek a factorization of the form \( L \times L^H \) where \( L \) is lower triangular. The uniqueness of this factorization is a necessary and sufficient condition to ensure that \( M \) is positive definite.

By way of example, we shall now present a recursive realization of this factorization. It can be represented by the calculation scheme:
\[
L_n \times L_n^H = \begin{bmatrix}
\ell_{11} & 0 & 0 & \ldots & 0 \\
0 & \ell_{22} & 0 & \ldots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots \\
0 & \ldots & \ldots & \ldots & \ell_{nn}
\end{bmatrix} \times \begin{bmatrix}
\ell_{11} & \ell_{21} & \ldots & \ell_{n1} \\
0 & \ell_{22} & \ldots & \ell_{n2} \\
\vdots & \ddots & \ddots & \ddots \\
0 & \ldots & \ldots & \ell_{nn}
\end{bmatrix}^* = \begin{bmatrix}
\ell_{11} & 0 \\
0 & \ell_{22} \\
\vdots & \ddots \\
0 & \ldots & \ldots & \ell_{nn}
\end{bmatrix} \times \begin{bmatrix}
\ell_{11} & 0 \\
0 & \ell_{22} \\
\vdots & \ddots \\
0 & \ldots & \ldots & \ell_{nn}
\end{bmatrix}
\]

From this, we deduce that:

\[
\begin{cases}
\ell_{11} = \sqrt{a_{11}} \\
\bar{v} = \frac{A_{(2:n,1)}}{\ell_{11}} \\
L_{n-1}L_{n-1}^H = A_{(2:n,2:n)} - \bar{v}\bar{v}^H
\end{cases}
\]

It should be noted that, in terms of performances, MATLAB®, like all interpreters, is ill-suited for recursive programming. Nevertheless, it can be used to test the algorithm before writing a program to execute it in a more suitable language, such as C.

**Exercise 4.10 (Cholesky factorization)** (see p. 214)

Write a function to perform the Cholesky factorization using a recursive algorithm.
Chapter 5

Speech Processing

Speech is an important field for the applications of digital signal processing. This first paragraph deals with how transporting and storing a signal can be facilitated by analyzing and compressing it. The arguments outlined below make use of certain elementary concepts regarding random processes – particularly covariances and autoregressive modeling.

5.1 A speech signal model

5.1.1 Overview

The first issue is the choice of the sampling frequency. When it comes to telephone communications, there usually are two constraints: the message must be comprehensible, and it must be possible to identify the person speaking. These constraints mean that the frequency band can be restrained to the [300-3,400] Hz interval, which implies a Nyquist frequency of 8 kHz. Figure 5.1 shows 2,000 values, or 0.25 s of a speech signal sampled at that frequency.

![Figure 5.1 – Speech signal sampled at 8,000 Hz. The x-coordinates correspond to the number of samples](image)

This 300-3,400 Hz band is called the telephone-band. Of course, the larger
the band, the better the quality. However, as the width of the band increases, so does the amount of informations sent per unit of time!

In order to enhance the quality, a larger band is considered that goes from 150 Hz up to 7 kHz; this band is devoted to the wideband digital handsets telephones. It works well for speech signals but is still insufficient for musical signals. In the case of music, usually two qualities are considered: the FM band (short for Frequency Modulation) quality used for frequency modulated radio broadcasts, which goes up to 15 kHz, and the HIFI band quality, used for example for compact discs, which goes up to 22 kHz. In any case, the speech signal must be sampled, according to the sampling theorem, at a frequency at least twice that of the band used.

<table>
<thead>
<tr>
<th>Quality</th>
<th>Maximum frequency</th>
<th>Sampling frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Telephone-band</td>
<td>300-3,400 Hz</td>
<td>8,000 samples/s</td>
</tr>
<tr>
<td>Wideband</td>
<td>150-7,000 Hz</td>
<td>16,000 samples/s</td>
</tr>
<tr>
<td>FM band</td>
<td>50 Hz-15,000 Hz</td>
<td>32,000 samples/s</td>
</tr>
<tr>
<td>HIFI band</td>
<td>&lt;22,050 Hz</td>
<td>44,110 samples/s</td>
</tr>
</tbody>
</table>

5.1.2 A typology of vocal sounds

Consider the part of the signal shown in Figure 5.1. As you can see, there are two clearly distinct sections corresponding to two types of sounds:

- the sounds that have the aspect of a harmonic vibration and that are said to be voiced. Vowels are a perfect illustration of this type of sound. An example is shown in Figure 5.1 in the window with the indices from 0 to 1,200;

- the sounds we interpret more as noise, and that are said to be unvoiced. An example is shown in Figure 5.1 in the window with the indices from 1,200 to 1,800.

Vowels generally last longer than consonants. They can easily be recognized by their harmonic aspect. Consonants are divided in the following categories:

- the nasal consonants/m/, /n/… for which the “oral cavity + pharynx” system forms a closed resonant cavity, with the air going the nostrils;

- the unvoiced fricatives /f/, /s/, /ch/… produced by turbulence of a continuous air flow in the oral cavity. The cavity is divided in two sub-cavities, the one in the back causing the “zeros” in the transfer function;

- the voiced fricatives /v/, /z/… which can be described in the same way as the unvoiced fricatives but with vibrations of the vocal folds;

- the voice plosives /b/, /d/… which are transitions caused by the sudden opening of the oral cavity following a rise in pressure. They strongly depend on the vowels they are pronounced with;
Speech Processing 107

the voiced plosives /p/, /t/, etc.

Figure 5.2 illustrates the case of the sounds “sh” and “ee”.

Figure 5.2 – Temporal shapes of a speech signal sampled at 8,000 Hz: top graph: an unvoiced sound; bottom graph: voiced sound

5.1.3 The AR model of speech production

The production of sounds is a very complex phenomenon that cannot be easily described by a model. It is bound to the anatomy of the vocal tract, represented in Figure 5.3.

A functional represents [15] is shown in Figure 5.4. The vocal tract is simplified as a series of cavities, the shapes of which change with time as air, coming from the lungs, flows through them.

Studies conducted on the vocal tract show that the two type of sounds, voiced and unvoiced, can be described using as a model the output of a all pole linear filter of the type $1/A(z)$, the order of which is between 10 and 50, and with, as the input:

- a white noise for unvoiced sounds;
- and an impulse sequence for voiced sounds.

The impulse sequence associated with voiced sounds corresponds to the derivative of the volume of the air flowing through the glottis (the space between the vocal cords) which opens and closes periodically, with opening phases
that last longer than the closing phases. This leads to a sudden decrease, during the closing phase, of the air flow which causes by derivation a very brief negative impulse (see Figure 5.5). Although it is sometimes sufficient to crudely approximate the sequence produced by the glottis by a simple sequence of ideal impulses, there are more sophisticated glottal excitation models [21], such as those of Rosenberg or Liljencrants-Fant (see Figure 5.5).

The fundamental frequency of the periodic sequence of glottal impulses is called the *pitch*. This frequency goes from about 70 Hz for a very low voice to 450 Hz for very high one. For a man, it goes basically from 70 to 200 Hz, for a woman from 140 to 350 Hz, and for a child from 180 to 450 Hz. In any case, for a given person, the pitch varies in the course of a conversation. For voiced sounds, the sequence of periodic impulses produced by the vocal folds acts as a frequency analyzer and causes resonant frequencies to appear in the vocal
These frequencies are called formants. As you can see in Figure 5.6, four formants are found by following the spectrum’s envelope shown on the right.

Let $1/A(z)$ be the transfer function of the filter used as a model for the vocal tract with $A(z) = 1 + a_1 z^{-1} + \cdots + a_P z^{-P}$. In the case of an unvoiced sound, the input can be seen as a white noise, and therefore the speech signal \{s(n)\} is an autoregressive process:

$$s(n) = -a_1 s(n-1) - \cdots - a_P s(n-P) + w(n)$$

We can then estimate the coefficients of $A(z)$ from an unvoiced sound window, using the results obtained for AR processes with the Yule-Walker equa-
digital signal and image processing using MATLAB®

\[
\begin{bmatrix}
R(0) & R^*(1) & \ldots & R^*(P) \\
R(1) & R(0) & \ldots & R^*(P-1) \\
\vdots & \vdots & \ddots & \vdots \\
R(P) & R(P-1) & \ldots & R(0)
\end{bmatrix}
\begin{bmatrix}
1 \\
a_1 \\
\vdots \\
a_P
\end{bmatrix}
= \begin{bmatrix}
\sigma^2 \\
0 \\
\vdots \\
0
\end{bmatrix}
\]  

(5.1)

where \( R(k) \) is the covariance function and \( \sigma^2 \) the power of \( w(n) \).

Then, when the unvoiced signal is applied to the FIR filter with the transfer function \( A(z) \), we get an estimate of the white noise input.

\[
\{ s(n) \} \rightarrow A(z) \rightarrow \{ w(n) \}
\]

In the case of a voiced sound, the glottis signal is assumed to be a sequence of periodic impulses with the pitch frequency \( F_p \). If the vocal tract is described as an all pole filter \( 1/A(z) \) and if we assume that:

\[
g(n) = \sum_k i(n - kM) \approx \sum_k A \delta(n - kM)
\]  

(5.2)

provides a good approximation of the glottis signal with \( MT_s \approx T_p = 1/F_p \), the voiced signal \( s(n) \) is a periodic signal containing the same frequencies. If \( A(z) \) is a \( P \) degree polynomial, \( s(n) \) obeys the filtering equation:

\[
s(n) + a_1 s(n - 1) + \cdots + a_P s(n - P) = g(n)
\]  

(5.3)

Based on \( N \) observations, we get:

\[
\begin{bmatrix}
s(P+1) \\
s(P) \\
\vdots \\
s(N)
\end{bmatrix}
\begin{bmatrix}
s(1) \\
\vdots \\
s(N-P)
\end{bmatrix}
\begin{bmatrix}
1 \\
a_1 \\
\vdots \\
a_P
\end{bmatrix}
= \begin{bmatrix}
g(P+1) \\
\vdots \\
g(N)
\end{bmatrix}
\]

which is written in matrix form:

\[
[ s \quad S ] \begin{bmatrix}
1 \\
a
\end{bmatrix} = g
\]  

(5.4)

\( S \) is a Toeplitz matrix. The vector \( g \) is a periodic vector containing an \( A \) followed by \((M-1)\) zeros (see approximation (5.2)). Typically, for a pitch frequency of \( F_p = 190 \) Hz, the pitch period is \( T_p \approx 5 \) ms. If \( T_p \) is assumed to be much greater than the glottal impulse duration, the vector \( g \) contains almost nothing but zeros. Equation (5.4) can then be written:

\[
[ s \quad S ] \begin{bmatrix}
1 \\
a
\end{bmatrix} \approx 0
\]
and \(a\) can be estimated using a least squares approach by minimizing this vector. The advantage of this method is its simplicity\(^{(1)}\) because it requires neither the estimate of the period \(M\) nor the estimate of the phase corresponding to the exact moment when the glottis close. The minimization leads to:

\[
\hat{a} = -(S^T S)^{-1} S^T s
\]

an expression similar to the expression \(a = -R^{-1} r\) found with a Yule-Walker equation which relates the parameters of an AR process with the covariance coefficients. Once \(\hat{a}\) has been estimated, we can then calculate \(g(n)\) with the use of expression (5.3), that is simply by feeding the speech signal \(\{s(n)\}\) into the input of the FIR filter with the transfer function \(A(z)\).

\[
\{s(n)\} \rightarrow A(z) \rightarrow \{g(n)\}
\]

As a conclusion, whether the sound is voiced or not, the estimate of the residual signal, glottis impulses in the first case and white noise in the second, is obtained by filtering the signal by the filter with the transfer function \(A(z)\), the coefficients of which are estimated as the parameters of an autoregressive process. The following example allows you to experimentally check the shape of the residual signal for a speech signal.

**Example 5.1 (Observation of a speech signal’s residual)** First record a voiced sound /ee/ and an unvoiced sound /sh/ at 8 kHz. Then create a program:

- that estimates the coefficients \(\{a_k\}\) of a 20th order all pole model for an analysis window with a duration of 30 ms, or 240 samples;
- that performs the filtering with the transfer function \(A(z)\) of the same speech signal block.

Apply this process to both the voiced and the unvoiced sound.

**HINTS:** type the following program:

```
%===== residuAR.m
clear
load voye;  \% voyel
load conch; \% consonant
Fs=8000; tbloc=.05; \% block size (ms)
modorder=20; nb=tbloc*Fs; mtime=(0:nb-1)/Fs;
sigi=ltre(1+1600:nb+1600); sigch=ltrch(1:nb);
```

\(^{(1)}\)When the glottal impulses have to be reconstructed with a higher accuracy, for example when it is used to perform a medical diagnosis, the least squares method used here can give insufficient results [15].
The results are shown in Figures 5.7 and 5.8. The graphs give the shape of the residual signal for a voiced sound and for an unvoiced sound. Very short impulses are clearly visible for the residual signal of a voiced signal, with a period of about 10 ms, that is about 100 Hz, corresponding to the closing frequency of the glottis.

Remember that the Toeplitz nature of the estimate of a covariance matrix ensures that the filter with the transfer function $1/A(z)$ is causal and stable. However, in the analysis problem, which consists of constructing, as we have just done, the input signal based on the observed signal, this property is not crucial since the filter $A(z)$ is a FIR filter, and is therefore stable. On the other hand, in the synthesis problem, which consists of reconstructing the speech signal based on the input signal, this property is of course essential.
5.1.4 Compressing a speech signal

We are going to present in this paragraph an example of speech signal compression. The word compression is used for the digitization of a signal as a bitstream with a rate as low as possible for a given level of distortion. Compression is too difficult a problem to be discussed here thoroughly. The solution we present is merely an introductory example, inspired from a encoder chosen a very wide choice of existing encoders [26, 15].

The simplest form of signal digitization is the b-bit uniform quantization of the values sampled at the frequency \( F_s \). Typically, for \( F_s = 8 \) kHz and \( b = 8 \) bits, the result is a 64 kbits/s stream called PCM (Pulse Code Modulation).

To achieve compression, in other words to use less bits, while maintaining a low level of distortion, a first idea would be to quantize the difference \( \delta(n) = s(n) - s(n-1) \) in place of the samples \( s(n) \). The result should be smaller than \( s(n) \) and hence the same accuracy would be achieved with less bits. This approach, which uses the correlations contained in the consecutive samples can easily be generalized by quantizing the residual signal \( \epsilon(n) = s(n) - (\alpha_1 s(n-1) + \cdots + \alpha_P s(n-P)) \), where the sequence \( \alpha_p \) is the one that minimizes \( \mathbb{E}\{|\epsilon(n)|^2\} \). As you may have noticed, we are once again faced with the linear prediction problem. This approach, called DPCM, for differential PCM has the advantage of being applicable to any signal for what we call waveform coding.

A second approach consists of considering the speech creating system and describing it using a model comprising a small number of parameters we are going to estimate. Then, all we have to do is reconstruct the signal from these parameters to obtain a signal that “sounds” like the original one. This type of encoder, called a vocoder, does not follow the shape of the original signal. This is why it cannot be used to compress a signal originating for example from a modem operating in the phone frequency band. However, the bitstream is usually lower, for the same quality, than the ones obtained in the first approach.
The example we chose belongs to the second category. The compression principle we decided on is based on representing the speech production system as an all pole filter, the input of which is either a sequence of periodic impulses, in the case of voiced sounds, or a white noise in the case of unvoiced sounds. The result is the synthesis diagram shown in Figure 5.9. With this approach, we assume that the signal is stationary, which is almost true for speech signals with a duration of about 10 to 20 milliseconds.

To have a better idea of this model, consider a signal sampled at 8,000 Hz with 8 bits, which corresponds to a rate of 64,000 bits/s, and assume that for each block of \( N = 180 \) samples, corresponding to a duration of 22.5 ms, \( P = 22 \) parameters are extracted. If we use 8 bits to represent each of the 22 parameters, \( 180 \times 8 = 1,440 \) bits are replaced with \( 22 \times 8 = 176 \) bits, which means we have a compression factor of about 8. In terms of rate, everything happens as if only 1 bit of each sample was kept. This would be the rate obtained if, for example, we only kept the sign bit of the samples \( s(n) \). You can run the following program:

```matlab
signs = sign(s);
soundsc(signs,8000);
```

where \( s \) represents the sample sequence of a speech signal sampled at 8,000 Hz.

![Figure 5.9 – Creating voiced and unvoiced sounds with all pole filtering](image)

**Exercise 5.1 (Compression of a speech signal)** (see p. 215)
The task you are supposed to do is divided in three parts:

1. Detecting whether a sound is voiced or unvoiced, and pitch measurement:
   when a signal is periodic, its autocorrelation function, defined by the normalized covariance:
   
   \[
   \rho_{XX}(n_1,n_2) = \frac{\mathbb{E}\{X_c(n_1)X_c^*(n_2)\}}{\sqrt{\mathbb{E}\{|X_c(n_1)|^2\} \mathbb{E}\{|X_c(n_2)|^2\}}}
   \]
   shows maxima distant from each other by one period of the fundamental. Hence the idea to detect whether or not the sound is voiced and to
measure the pitch based on the computation of the estimate \( J(k) \) of the autocorrelation function:

\[
J(k) = \frac{\sum x(n)x(n-k)}{\left(\sum x^2(n)\right)^{1/2}\left(\sum x^2(n-k)\right)^{1/2}}
\]  

(5.5)

The use of this function, incidentally, will be justified by a least squares approach when we deal with the similar problem of cardiac rhythm estimation (see page 136).

As we have seen, the pitch belongs, in practice, to an interval \((F_{\text{min}}, F_{\text{max}})\). You can therefore restrain the computation of the function \( J(k) \) to the values of \( k \) greater than \( k_{\text{min}} = F_s/F_{\text{max}} \) and smaller than \( k_{\text{max}} = F_s/F_{\text{min}} \), where \( F_s \) refers to the sampling frequency. If the maximum of \( J(k) \) goes beyond a certain threshold, typically 0.6, the sound is considered voiced, and the pitch period is then given by the value \( k_0 \) corresponding to the maximum of \( J(k) \).

Write a function that detects sound activity, determines whether the sound is voiced or unvoiced and measures, in the case of a voiced sound, the pitch period. Use this function to divide the signal voiced and unvoiced areas, using windows of 240 points, that is 30 ms of signal, with an overlap rate of 25%.

2. All pole filter parameter extraction based on the previous partitioning:

- use the \texttt{xtoa} function:

```matlab
function [a,sigma2]=xtoa(x,P)
%!==================================================!
%! SYNOPSIS: [a,sigma2]=XTOA(x,P)!
%! x = signal Coefficients array  
%! P = model order  
%! a = [1 a_1 a_2 ... a_P]   
%! sigma2 = power of the input white noise !
%!==================================================!
N=length(x); x=x(:); x=x-mean(x);
for k=1:P+1
    %==== biased estimate of R^*(k)
    rconj(k)=x(k:N)' * x(1:N-k+1) / N;
end
Rc=toeplitz(rconj); vaux=Rc \ eye(P+1,1);
a=vaux / vaux(1); sigma2=1 / vaux(1);
```

to extract the \( P \) coefficients of the all pole filter associated with each analysis window. You can choose \( P = 20 \) as the model order for a voiced sound window and \( P = 10 \) for an unvoiced sound window;

- create a coefficient file containing for each block the coefficients of the all pole model, as well as the value of the pitch when the signal is voiced. Set the pitch value to zero to indicate an unvoiced sound.
3. Synthesis: write a program that achieves the synthesis. To tone down the possible sudden variations from the coefficients of one window to those of the next, set the overlap to 25% for the consecutive outputs calculated over time intervals of 30 ms. Use a white Gaussian noise or a simple sequence of impulse with identical amplitudes as the filter’s input, depending on whether the sound is unvoiced or voiced.

In Exercise 5.1, we did not quite perform the encoding operation we said we would. What we should have done is use 176 bits to encode all 22 parameters for every time of analysis, but that is not as simple as it seems, because we cannot just simply associate 8 bits to each of the 22 parameters, the bit distribution has to be optimal. A classic approach is to study, using a speech database, the parameter distribution and to create a codebook of representative elements by a vector quantization.

A voice activity detector usually precedes the whole encoding system. Its role is to determine the segments containing “silence”, for which no parameter estimation is done. When reconstructing the signal, a faint noise is added to these segments of silence for purposes of listening comfort.

5.2 Dynamic Time Warping

In signal processing, and particularly in the field of speech, different series of measurement conducted in seemingly identical conditions can provide recordings that actually show significant differences in terms of amplitude, duration, utterance speed, etc. The algorithm, which will be detailed later, performs a time-alignment of two observation sequences independently from possible differences in amplitude, duration, or utterance speed. It is commonly called the DTW algorithm, short for Dynamic Time Warping.

In 1975, Itakura [12] suggested using the DTW for speech recognition. In the particular case of the recognition of isolated words, a dictionary is used, containing the sound recordings of the words to be recognized, and during the recognition operation, it has to be decided which word has been pronounced based on the observed sound signal. If the sound signal corresponding to a given word were perfectly reproduced, we would simply have to subtract, sample by sample, the sample signal from the reference signal: a word would then be recognized if the difference is equal to zero. Unfortunately, this is never the case. So, in order to perform the comparison, we are going to associate as best we can the consecutive phases of the signal to be recognized with those of the different dictionary signals and find a value for the discrepancy. The DTW performs both those operations.
5.2.1 The DTW algorithm

Consider two sequences of length $d$ vectors $\{x_1, \ldots, x_I\}$ and $\{y_1, \ldots, y_J\}$. Let:

$$d(i, j) = \sqrt{\sum_{\ell=1}^{d} (x_{i,\ell} - y_{j,\ell})^2}$$

with $1 \leq i \leq I$ and $1 \leq j \leq J$, be the distance between the two vectors $x_i$ and $y_j$.

A solution to the time alignment problem consists of taking one by one the indices of the sequences $\{x_i\}$ and $\{y_j\}$ using a pair of functions $\phi = (\phi_x, \phi_y)$ defined on $\{1, \ldots, T\}$ and within the range $\{1, \ldots, I\} \times \{1, \ldots, J\}$, and to calculate the cumulated sum of the distances associated with $\phi$:

$$d_{\phi}(I, J) = M_\phi^{-1} \sum_{k=1}^{T} m_k d(\phi_x(k), \phi_y(k))$$

where the $m_k$ are positive weighting coefficients and $M_\phi$ is a normalization constant given by:

$$M_\phi = \sum_{k=1}^{T} m_k$$

It seems $M_\phi$ should depend on the choice of $\phi$. In practice, the $m_k$ are chosen such that $M_\phi$ is independent of $\phi$. This can be achieved for example by choosing:

$$m_k = (\phi_x(k) - \phi_x(k-1)) + (\phi_y(k) - \phi_y(k-1)) \quad \text{(5.6)}$$

The function $\phi$ satisfies different types of constraints, such as:

- initial and final values: $\phi_x(1) = 1$, $\phi_x(T) = I$, $\phi_y(1) = 1$, $\phi_y(T) = J$;

- trajectory monotony and continuity: $0 \leq \phi_x(k) - \phi_x(k-1) \leq 1$ and $0 \leq \phi_y(k) - \phi_y(k-1) \leq 1$;

- local continuity: finally, the pair $(\phi_x(k), \phi_y(k))$ satisfies certain pathfinding rules.

The comparison is performed on all the elements in the dictionary. Figure 5.10 illustrates the comparison of $\{x_1, \ldots, x_I\}$ with $\{y_1^{(1)}, \ldots, y_{J_1}^{(1)}\}$, $\{y_1^{(2)}, \ldots, y_{J_2}^{(2)}\}$, $\ldots$, $\{y_1^{(N)}, \ldots, y_{J_N}^{(N)}\}$. 
5.2.2 Examples of pathfinding rules

To ensure local continuity, a sequence of possible paths is defined by a graph such as the ones shown in Figures 5.11 and 5.12: the arrows in these figures show the only possible paths that can be taken to reach the final point.

The goal of the DTW algorithm is to determine:

\[
D(I, J) = \min_{\phi \in \Phi} d_{\phi}(I, J)
\]
where $\Phi$ is the set of functions that satisfy the constraint.

Dynamic programming is a recursive approach introduced by R. Bellmann which allows the previous criterion to be minimized using the following property: consider two sequences $\{x_1, \ldots, x_I\}$ and $\{y_1, \ldots, y_J\}$ and let $C(I, J)$ be the minimal length DTW path associated with these two sequences. Then the sub-path of the path $C(I, J)$ which reaches the point with coordinates $(x_i, y_j)$ is optimal for the two sub-sequences $\{x_1, \ldots, x_i\}$ and $\{y_1, \ldots, y_j\}$, because among all the paths that lead to the point with coordinates $(i, j)$, minimization means we only keep the shortest one and therefore:

$$D(i, j) = \min_{\phi \in \Phi} d_{\phi}(i, j)$$

Thus, for the constraint graph shown in Figure 5.11, we infer that:

$$D(i, j) = \min \begin{cases} D(i - 1, j) + d(i, j) \\ D(i - 1, j - 1) + 2d(i, j) \\ D(i, j - 1) + d(i, j) \end{cases} \quad (5.7)$$

Likewise, for the constraint graph shown in Figure 5.12, we have:

$$D(i, j) = \min \begin{cases} D(i - 2, j - 1) + 0.5d(i - 1, j) + 0.5d(i, j) \\ D(i - 1, j - 1) + d(i, j) \\ D(i - 1, j - 2) + 0.5d(i, j - 1) + 0.5d(i, j) \end{cases} \quad (5.8)$$

**Exercise 5.2 (DTW)** (see p. 219) Write a function that implements the DTW algorithm by taking the pathfinding constraints found in the graph in Figure 5.11. Use the two sequences of vectors as the input and as the output the array of the cumulated sums $D(i, j)$, as well as the value $D(I, J)/(I + J)$ as the output.

Exercise 5.3 uses the function obtained in Exercise 5.2 to recognize a word in a given list of words. We will first briefly describe the cepstral analysis used to provide the characteristic sound features the DTW alignment is based on.

### 5.2.3 Cepstral coefficients

We know that the phonetic content of a speech signal can be characterized in a satisfactory way by a short-term spectral representation. This is why in speech recognition the first signal is usually performed, consisting of such an analysis over windows of about 20 milliseconds with overlap. Most of the time, the extracted characteristics are the first values of the short-term **cepstrum**,
defined as the inverse Fourier transform of the logarithm of the modulus of the signal’s DFT. If \( x(n) \), with \( n \in \{0, \ldots, N - 1\} \), refers to a portion of a signal with a duration of \( N \), possibly weighted by a window, then the cepstral coefficients have the expression:

\[
c(k) = \frac{1}{L} \sum_{\ell=0}^{L-1} S(\ell) e^{2j\pi n \ell / L} \text{ with } S(\ell) = \log \left| \sum_{n=0}^{N-1} x(n) e^{-2j\pi n \ell / L} \right|
\]

It can be proven that the first cepstral coefficient represents the energy of the signal segment and that the following \( d \) coefficients, where \( d \) corresponds to a duration of a few milliseconds, characterize the shape of the vocal tract. In speech recognition, the relevant information is essentially characterized by the shape of the vocal tract. This is why most recognition methods are based on the use of the first cepstral coefficients. Most of the time, the cepstral coefficients are calculated on a logarithmic frequency scale; this is called \( MFCC \), for \( Mel \) Frequency Cepstral Coefficients. Exercise 5.3 only uses a linear frequency scale, leading to a sequence of length \( d \) vectors that will undergo the recognition processes.

**Exercise 5.3 (DTW word recognition) (see p. 221)**

1. Write a function that extracts the short-term cepstrum from a speech signal sampled at 8,000 Hz using the operations:
   - multiplication of the time window by a Hamming window: choose a window duration corresponding to 20 ms with a temporal overlap of 50%;
   - computation of the logarithm of the modulus of the Fourier transform over \( L = 256 \) points;
   - computation of the inverse Fourier transform;
   - extraction of the 12 values useful to recognition.

2. Write a program that uses the DTW to compute, based on the sequences of cepstral coefficients, the “distance” between two recordings of the same pronounced word, and between two recordings of two different pronounced words.

### 5.3 Modifying the duration of an audio signal

Modifying the temporal scale of a sound has applications in many fields, such as solutions for the hearing impaired, speech design and recognition, movies, TV and radio advertisement, etc. A simple way of performing this modification is to reconstruct the signal with a sampling frequency different from the one used
for the acquisition. Unfortunately, this method causes frequency distortion, because, as we know, if $X(f)$ represents the Fourier transform of $x(t)$, then the Fourier transform of $x(\gamma t)$, with $\gamma > 0$, is given by $\gamma^{-1}X(f/\gamma)$. Therefore, the frequency axis is dilated or contracted, depending on whether $\gamma$ is greater or smaller than 1. You can observe the effects on a speech signal with the use of the MATLAB® function `sound` and by trying different reconstruction frequencies: when the frequency is greater than the acquisition frequency, the pitch seems higher. To prevent this type of distortion, several techniques have been suggested. The most popular one is probably the technique called PSOLA, for Pitch Synchronous Overlapping Addition, which works in the time domain [27]. Another one is referred to as the “phase vocoder”, which works in the frequency domain [16]. One of the drawbacks of PSOLA is the addition of unwanted noises. As for the phase vocoder, it causes a reverberation effect.

### 5.3.1 PSOLA

To reduce the total duration of the signal, while preserving the frequency scale, we can simply eliminate small segments of the signal throughout the recording. Conversely, to increase the total duration of the signal, we can duplicate some of its segments. However, these segments have to be long enough to prevent spectrum aliasing, but short enough compared with the duration of the basic acoustic units: with PSOLA [27], the durations of the segments have to be chosen equal to the “instantaneous” pitch period. PSOLA is essentially comprised of two steps:

- **analysis**: the signal $s(t)$ is time-“marked” according to a sequence of analysis times $t_a(i)$ such that $t_a(i) = t_a(i - 1) + P_a$, where $P_a$ refers to the “instantaneous” fundamental period (the pitch) estimated with a window long enough, starting at the time $t_a(i - 1)$;

- **synthesis**: the modified signal $\tilde{s}(t)$ is constructed by an overlap-add operation on the basic segments $s_i(t)$ of the original signal relocated at synthesis times $t_s(j)$ according to the expression:

$$\tilde{s}(t) = \sum_n \tilde{s}_i(t - t_s(n))$$

where $\tilde{s}_i(t - t_s(n)) = h(t - t_a(i(n)))s(t)$ is a signal segment centered in $t_a(i(n))$. The synthesis times are such that $t_s(n) = t_s(n - 1) + P_a(i(n))$. If $\gamma$ refers to the speed modification rate, the index $i(n)$ is the closest integer to the value $n\gamma$. 
Exercise 5.4 (PSOLA) (see p. 222)

1. **Analysis**: use the `f0cor.m` function (page 139), which estimates a signal’s pitch, to construct the sequence of analysis times $t_a(n)$ in the following way:

   - the initial values are set as $t_a(1)=1$ and $P_a = L_{10}$ where $L_{10}$ is the number of samples corresponding to 10 ms;
   - the window starting at the time $t_a(n)$ and lasting at least two pitch periods $P_a$ is extracted. In practice, the duration has to be chosen equal to twice the smallest period expected in the signal;
   - pitch detection is performed on the window. If the signal is voiced, we get the pitch period $P_a$ and the analysis time $t_a(n+1) = t_a(n) + P_a$. If the signal is unvoiced, the value $t_a(n+1) = t_a(n) + L_{10}$ is chosen.

   At the end of this first process, we have a sequence of $N_a$ analysis times (positions in the file) $t_a(n)$ that are synchronous with the pitch period;

2. **Synthesis**: we wish to modify the prosodic speed by a factor $\gamma$. This is done by generating the sequence of synthesis times $t_s$ as follows:

   - initially, $t_s=1$;
   - the following synthesis time is calculated by executing:

     ```matlab
     te = te + gamma;
     ie = ceil(te);
     ```

     The index is used to point out the analysis time after which the segment of the signal used for creating the desired signal is extracted. Hence, if $\gamma < 1$, which corresponds to a slower utterance, the value of $i_e$ after $n$ iterations will be smaller than $n$. Hence, some segments of the signal will be repeated and the signal created will last longer;

   - the sequence of synthesis times is generated by executing:

     ```matlab
     ts = ts + (ta(ie+1) - ta(ie));
     ```

     This is equivalent to generating a window with a length equal to the pitch (hence the phrase Pitch Synchronous), because, using obvious notations, we can write:

     \[
     t_s(n + 1) = t_s(n) + \left( \frac{t_a(i_e(n) + 1) - t_a(i_e(n))}{P_a} \right)
     \]

     Table 5.1 shows some of the values for the sequences that were found for $\gamma = 0.8$. As you can see, the portion centered in 7,668 in the original signal is in position 9,610 in the synthesis signal. Notice also that, because
the signal is slowed down, as we wanted it to be, we have to duplicate the portion centered on 7,629, once in position 9,532, and once more in the following position 9,571.

The signal synthesis is performed as follows:

- the segment centered in $t_a(i_e)$ and with a length of $2(t_a(i_e + 1) - t_a(i_e))$ is extracted from the original signal;
- the extracted segment is multiplied by a Hann window then added to the previous portion with a 50% overlap.

Write a function that consecutively performs the two steps of the process.

### 5.3.2 Phase vocoder

The basic idea behind the phase vocoder [16] is to perform a short-term Fourier analysis. If the spectrum is comprised of narrow band spectral components, which means that the sound is closer to a voiced sound than it is to an unvoiced sound, and that the analysis window is much longer than the pitch period, then the values of every pitch harmonic will be clearly identified. In this case, if the spectral characteristics are maintained, for a duration slightly lower or slightly greater than the original one, then the value of the pitch is preserved.

The only difficulty is that the phases associated with each frequency component in the modified signal’s spectrum have to be calculated in such a way as to ensure the proper alignment of the consecutive phases during the overlap-add operation.

The following exercise shows that the sum of overlapping Hann windows remains constant. This property is used in exercise 5.6.

**Exercise 5.5 (Hann window)** (see p. 223)

Consider the sequence (called a Hann window):

$$h(n) = \begin{cases} 0.5(\cos(2\pi n/L) + 1) = \sin^2(\pi n/L) & \text{for } n \in \{0, \ldots, L-1\} \\ 0 & \text{otherwise} \end{cases}$$

Let $\alpha \in (0, 1)$, and $n_0 = \lfloor \alpha L \rfloor$. Write a program that plots against $n$ the

<table>
<thead>
<tr>
<th>$n$</th>
<th>1 ...</th>
<th>199</th>
<th>200</th>
<th>201</th>
<th>202 ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_a(n)$</td>
<td>1 ...</td>
<td>159.4</td>
<td>160.2</td>
<td>161.0</td>
<td>161.8 ...</td>
</tr>
<tr>
<td>$i_a(n)$</td>
<td>1 ...</td>
<td>160</td>
<td>161</td>
<td>161</td>
<td>162 ...</td>
</tr>
<tr>
<td>$t_s(n)$</td>
<td>1 ...</td>
<td>9,493</td>
<td>9,532</td>
<td>9,571</td>
<td>9,610 ...</td>
</tr>
<tr>
<td>$t_a(i_e(n))$</td>
<td>1 ...</td>
<td>7,590</td>
<td>7,629</td>
<td>7,629</td>
<td>7,668 ...</td>
</tr>
</tbody>
</table>

Table 5.1 – Sequences of synthesis and analysis times corresponding to $\gamma = 0.8$, that is a slower utterance
sequence:

\[ x(n) = \sum_{k} h^2(n - kn_0) \quad (5.9) \]

Notice that the sequence \( h(n) \) has a finite length. Therefore, to construct \( x(n) \), all you need to do is to calculate the sum of a finite number of segments distant from each other by \( h^2(n) \).

What happens when \( L \) varies, when \( \alpha \) varies? Try other powers of \( h(n) \) in expression (5.9).

**Exercise 5.6 (Phase vocoder) (see p. 224)**

Write a program that performs the following operation:

- calculation of the DFTs of the signal segments, each segment weighted by a length \( L \) Hann window \( h(n) \). The consecutive windows are distant from each other by a number of points \( n_0 = \lfloor \alpha L \rfloor \) where \( \alpha \in (0, 1) \). With such an overlap, we have:

\[ \sum_{k} g(n - kn_0) = C \]

where \( g(n) = h^2(n) \) and where \( C \) is constant depending on \( \alpha \) that you will determine. According to what we saw in Exercise 5.5, if \( L \) is a power of 2, the distance between the windows has to be in the form \( n_0 = L/2^m \);

- generating the sequence of synthesis times with a factor \( \gamma \) compared with the original sequence according to the method suggested in Exercise 5.4;

- calculation of the modified DFTs by performing an amplitude interpolation, proportional to the original DFTs on the interval containing the synthesis time. For the phase calculations, sum the phase increases of the original DFTs;

- calculation of the inverse DFTs of the modified DFTs, each DFT being once more weighted by a length \( L \) Hann window. For \( \gamma = 1 \), everything happens as if the window \( g(n) = h^2(n) \) were applied and, therefore, the correct amplitudes can theoretically be found by dividing the obtained signal by the constant \( C \).

**5.4 Eliminating the impulse noise**

We are now going to look into the restoration of recordings containing errors with a relatively large amplitude and with a very brief duration (less than a millisecond), referred to as clicks, and assumed to be in small numbers. Thus,
the signal represented in Figure 5.13, originating from record, shows a scratch spread over a few samples. Cracks are not always as “visible” as this one, and the recording often has to be listened to in order to detect them.

![Figure 5.13 – Signal originating from a record and containing a click](image)

The considered restoration method is comprised of two separate steps:

1. the detection of clicks in the signal;
2. the restoration of corrupted samples.

### 5.4.1 The signal model

When observing the signal \( s(n) \) shown in Figure 5.13, what makes us decide that there is a click in position \( n_0 \) is that the value in \( n_0 \) is noticeably different from what would seem predictable to us based on the past signal. Therefore, if we know how to calculate a prediction \( \hat{s}(n) \) based on the last \( K \) values \( s(n-1), \ldots, s(n-K) \), the procedure can be made automatic, by deciding the presence of a click in position \( n \) if the difference between \( \hat{s}(n) \) and \( s(n) \) is above a threshold that can be determined experimentally.

For an WSS process, the best \( K \) order linear predictor:

\[
\hat{s}(n) = \alpha_1 s(n-1) + \cdots + \alpha_K s(n-K) \quad (5.10)
\]

in the least squares sense is obtained by choosing the solutions to the Yule-Walker equation (5.1) as the coefficients \( \alpha_i \). The expression of the prediction error \( \sigma^2 \) is given by:

\[
\varepsilon^2 = R(0) - \sum_{k=1}^{P} \alpha_k R(-k) \quad (5.11)
\]
If \( s(n) \) is a \( K \) order AR process, the prediction process \( s(n) - \hat{s}(n) \), also called the residual signal, is white. Because we are going to use this property in click detection, we will assume that this is the case: in the absence of clicks, the signal \( s(n) \) is a \( K \) order AR process. From now on, \( K \) is assumed to be known. In practice, its value is set by examining the results.

If the number of clicks is small, we can assume that the effects caused by their presence in the estimation window are negligible. We can then estimate the parameters \( a_1, \ldots, a_K \) and \( \sigma^2 \) of the \( K \) order AR process directly on the signal’s window. Once these values have been estimated, the FIR filter with the transfer function \( A(z) = 1 + a_1z^{-1} + \cdots + a_Kz^{-K} \) can be applied to the signal \( x(n) \). The signal \( y(n) \) is obtained. Because \( s(n) \) is a \( K \) order AR, then in the absence of clicks, this signal \( y(n) \) is a white noise with the variance \( \sigma^2 \). If, in addition to that, \( s(n) \) is Gaussian, then \( y(n) \) is itself Gaussian. Let us now see how to detect the presence of clicks.

### 5.4.2 Click detection

We now assume that a click is described as an impulse \( i(n) = A_0\delta(n-n_0) \) added to the useful sound signal \( s(n) \). The observed signal is then \( x(n) = s(n) + i(n) \). We are going to try to detect the possible presence of \( i(n) \) using a linear filter, then by comparing the filter’s output with a threshold. The following exercise shows how the methods works.

**Exercise 5.7 (Detecting impulse clicks)** (see p. 226)

Let \( d(n) \) be a signal with a known shape, corrupted by a noise \( b(n) \). “Detecting” the signal \( d(n) \) means we have to choose a decision rule to be able to say if the signal \( d(n) \) is or not in the observed signal \( y(n) \). Hence we have two hypotheses that can be summed up as follows:

- in the absence of the signal, what we see is \( y(n) = b(n) \);
- in the presence of the signal \( d(n) \), what we see is \( y(n) = d(n) + b(n) \).

We will assume that \( b(n) \) is a white noise, with the variance \( \sigma^2 \). To conduct the processing, we are going to impose that the detector is comprised of a linear filter followed by a threshold comparator, and we will find the optimal settings for the filter and the threshold value.
Let \( z_d(n) \) and \( z_b(n) \) be the outputs of the filter \( g(n) \) we are trying to determine, the inputs of which are the deterministic signal \( d(n) \) and the noise \( b(n) \).

1. Show that the signal-to-noise ratio (SNR) defined by
\[
\rho = \frac{|z_d(n)|^2}{\mathbb{E}\{|z_b(n)|^2\}}
\]
has the expression:
\[
\rho = \frac{1}{\sigma^2} \left| \sum_{u=-\infty}^{+\infty} g(u) d(n-u) \right|^2 \sum_{u=-\infty}^{+\infty} |g(u)|^2
\]

2. Using the Schwartz inequality, find the expression of the impulse response of the filter \( g(n) \) that maximizes the value of \( \rho \). In the literature, this filter is called a matched filter (the phrase implies that the filter is matched with the signal \( d(n) \)).

3. Consider now the problem of detecting, with a linear filter \( h(n) \), an impulse \( \delta(n) \) in a \( K \) order \( AR \) signal defined by \( s(n) + a_1 s(n-1) + \cdots + a_K s(n-K) = w(n) \) where \( w(n) \) is a white noise with the variance \( \sigma^2 \).

To achieve such a detection, we can decompose \( h(n) \) in a cascade of two linear filters \( h_1(n) \) and \( h_2(n) \), the first one a FIR filter with the impulse response \( h_1(0) = 1, h_1(1) = a_1, \ldots, h_1(K) = a_K \) (Figure 5.15). Let \( y(n) \) be this filter’s output.

![Figure 5.15 – Signal whitening before detection](image)

By applying the result of the previous question, determine the filter \( h_2(n) \) that maximizes the detection signal-to-noise ratio. Let \( z(n) \) be the output of the filter \( h_2(n) \).

4. In the absence of clicks, determine as a function of \( a_1, \ldots, a_K \) and \( \sigma^2 \) the expression of the variance \( P_z \) at the output of the filter \( h_2(n) \).

5. A threshold \( s \) is set, and the following decision rule is adopted: if \( |z(n)| > s \), the presence of a click is decided at the time \( n \). Under the hypothesis that \( s \) is Gaussian, use the decision rule to show that the expression of the threshold that ensures a probability equal to \( \alpha \) of deciding the presence of a click where there isn’t one, is \( s = \lambda(\alpha) \sqrt{P_z} \). For \( \alpha = 0.01 \), we have \( \lambda = 3 \).
6. Simulation: write a program that generates 500 samples of a 10 order AR process, simulating the useful signal \( s(n) \). Use, as the parameters of the AR-10 process, the values \( \sigma^2 = 1 \) and \( a_1, \ldots, a_{10} \) given by:

```matlab
%===== AA.M
a = [1 -1.6507 0.6711 -0.1807 0.6130 -0.6085 0.3977 ...
     -0.6122 0.5412 0.1321 -0.2393];
```

Let \( s_e \) be the root mean square value of \( s(n) \), an estimate of which is given under MATLAB\textsuperscript{®} by \( \text{seff} = \sqrt{s \cdot s' / N} \). Add five clicks with amplitudes equal to \( \pm 1.5s_e \) at arbitrary times. Let us assume that the order of the filter is known, but that, despite the presence of a few clicks, the model’s parameters can be estimated with the `xtoa` function. Successively estimate the model’s parameters, the whitening, the matched filtering, then the comparison with the previously determined threshold.

The results obtained with the previous program show that a given click can lead to several close positions detected around the real value. In practice, it is preferable to “group” these positions together. Describe a processing algorithm corresponding to several positions detected for the same click.

**In Summary:** the previous results lead us to summarize click detection in the following operations:

- estimation of the \( K \) order AR model’s \( K + 1 \) parameters \( a_1, \ldots, a_K \) and \( \sigma^2 \) based on the length \( N \) window \( x(n) \);

- filtering of \( x(n) \) by the filter with the impulse response \( \{1, a_1, \ldots, a_K\} \);

- filtering of the previous signal by the matched filter with impulse response \( \{a_K, a_{K-1}, \ldots, a_1, 1\} \). The output signal is denoted by \( z(n) \);

- comparison of \( |z(n)| \) with the threshold:

\[
s = \lambda \sqrt{\sigma^2 (1 + a_1^2 + \cdots + a_K^2)}
\]

where the parameter \( \lambda \approx 3 \) is set experimentally.

### 5.4.3 Restoration

Once a click is detected in position \( n_0 \), you would think that only one value has to be reconstructed: that of the altered sample. But in fact, the error is rarely found on only one point of the signal. This is why it is better to consider several samples on both sides of the detected position to be errors. This means
we have a corrupt area of \( m \) consecutive values. Typically, \( m \) is chosen between 9 and 15 (see Figure 5.16).

The idea, which consists of performing a simple linear interpolation based on the values placed on either side of the corrupt area, does not take into account the correlations among the points of the signal. The solution we are going to use consists of using the prediction:

\[
\hat{x}(n) = \alpha_1 x(n-1) + \cdots + \alpha_K x(n-K)
\]  

(5.12)
of a \( K \) order AR process.

**Exercise 5.8 (Restoring “missing values”)** (see p. 229)

Let us assume the parameters of the AR model have been estimated, and that, in the considered block, the corrupt zone goes from position \( \ell \) to position \( \ell + m - 1 \). We are going to try, by minimizing the square deviation, to search for the best values of \( x(\ell), \ldots, x(\ell + m - 1) \) (Figure 5.16).

\[
\begin{align*}
\ell & \quad \ell + m - 1 \\
x(n) & \quad \times \quad \times \quad \times \quad \times \quad \times \quad \times \quad \times
\end{align*}
\]

*Figure 5.16 – Several values are restored around the detected position*

1. Show that estimating the \( m \) unknown values can be seen as a linear problem depending on the non-corrupt values and the model’s coefficients.

2. Use this result to estimate, using the least squares method, \( \hat{y} \) of the corrupt zone \( y = [x(\ell), \ldots, x(\ell + m - 1)]^T \).

Write the expression of \( \hat{y} \) in the form:

\[
\hat{y} = -(B^TB)^{-1}B^T(A_0 x_0 + A_1 x_1)
\]

(5.13)

where \( A_0 \), \( A_1 \) and \( B \) are matrices build from the model’s coefficients \( (a_1, \ldots, a_K) \) and \( x_0 \) and \( x_1 \) are vectors constructed from the non-corrupt observations \( x_1, \ldots, x_{\ell-1}, x_{\ell+m}, \ldots, x_N \).

The plots in Figure 5.17 were obtained using the following program. They show the signal containing the clicks and the signal after restoration:
%===== RESTAU.M
% Signal reconstruction
% Run the detection program, then
% select a position to restore in the list of
% positions detected by detect.m
pos=input('Click position: ');
lsig=length(sig); tps=[0:lsig-1]; sig=reshape(sig,lsig,1);
m=15; ell=pos-7;
X0=sig(ell-K:ell-1); X1=sig(ell+m:ell+m+K-1);
colT=[aest(K);zeros(m+K-1,1)];
ligT=[aest(K:-1:1)' zeros(1,m+K)];
T=toeplitz(colT,ligT);
A0=T(:,1:K); B=T(:,K+1:K+m);
A1=T(:,K+m+1:2*K+m); X=A0*X0+A1*X1;
%===== solving the system
Y=-B \ X; sigr=sig; sigr(ell:ell+m-1)=Y;
plot(tps,sig,'-r', tps,s,'b', tps, sigr,':y'); grid;

Figure 5.17 – Clicks (dashed line), restored signal (full line)

The method presented here is successfully used to clean up recordings, such as the ones stored on old records. It requires a few adjustments to determine experimentally the window size, the AR order and the threshold value that provide the best acoustic results.
Chapter 6

Selected Topics

6.1 Tracking the cardiac rhythm of the fetus

6.1.1 Objectives

The human heart’s activity produces electrical currents that spread through the tissue and can be measured with the use of electrodes attached to the skin. These signals are called an electrocardiogram (ECG, or EKG). In obstetrics, these signals can be used to keep watch over the cardiac condition of the fetus, in particular during the delivery, which allows doctors to detect possible anomalies very early on, and hence to treat them more rapidly.

There are essentially two methods that are used to track the cardiac rhythm of the fetus. The first one, which can only be carried out during the delivery, consists of measuring the electrocardiogram (ECG or EKG) directly off the fetus by placing an electrode on its scalp. The second one, the main advantage of which is to be non-invasive, consists of placing electrodes on the mother to pick up signals from which the fetal cardiac signal will be extracted.

In the second method, the use of a single sensor placed on the mother’s abdomen is not sufficient because the amplitude of the fetal EKG can be several times less than the noise produced by different sources of interference such as the mother’s EKG, but also the signals caused by her muscle activity (electromyogram) or the ones related to her breathing. Therefore, the observation is difficult. However, the use of several sensors makes it possible to separate the signals (Figure 6.1).

The data we are going to process in this work come from measurements done on EKG signals samples at $F_s = 300$ Hz over a duration of a few seconds. Many websites make such signals available to the public. From now on, the signal originating from the sensor on the chest will be denoted by $x_p$ and the one originating from the sensor on the abdomen by $x_v$ (Figure 6.2).
A reading of such signals is shown in Figure 6.3.

Two processings have to be conducted. First, we have to extract the fetus’s EKG from the signals $x_p(n)$ and $x_v(n)$ and, second, estimate the cardiac rhythm of the fetus based on the obtained signal.

### 6.1.2 Separating the EKG signals

Theoretically, the signal observed with the chest sensor represents the mother’s cardiac signal noised by the fetus by the signal originating from the heart of the fetus. However, because this sensor is located far away from the heart of the fetus, i.e. because its heartbeat is faint, we will assume that the signal $x_p(n)$ represents the mother’s heart filtered by traveling through the thoracic tissue. If the mother’s cardiac signal is denoted by $c_M(n)$, and if traveling through the tissue acts as a linear filter with a length $K_p$ finite impulse response, we have:

$$x_p(n) = g_1 c_M(n - n_p) + \cdots + g_{K_p} c_M(n - n_p - K_p + 1)$$  \hspace{1cm} (6.1)$$

where anyone $n_p$ accounts for the overall propagation delay through the chest. In practice, this value turns out to be small.

As for the signal $x_v(n)$, it is the sum of the cardiac signal of the fetus, filtered by traveling through the abdomen tissue and a signal originating from the
Figure 6.3 – Mother-fetus EKG signals. Graph on the top: signal measured off the chest, and which can be considered as the signal originating from the mother’s heart. Graph on the bottom: signal measured off the abdomen containing both cardiac signals.

mother’s heart. However, this last signal is not quite the signal $x_p(n)$ picked up by the chest sensor, because the signal produced by the mother’s heart reaches the abdomen sensor after traveling through the abdomen. Therefore, it has undergone some transformations. By assuming once more that the abdominal transfer behaves like a linear filter with a length $K_a$ finite impulse response filter, the signal originating from the mother’s heart can be written (in fact the FIR hypothesis should be related to the comments made on the approximation of an IIR by a FIR):

$$v_M(n) = f_1c_M(n - n_a) + \cdots + f_{K_a}c_M(n - n_a - K_a + 1)$$

where $n_a$ accounts for the overall propagation delay through the abdomen. By replacing this expression in (6.1) we end up with a relation between the disruptive signal from the abdomen sensor and the signal from the chest sensor, which is written:

$$v_M(n) = h_{-M}x_p(n + M) + \cdots + h_{-1}x_p(n + 1) + h_0x_p(n) + \cdots + h_{L-1}x_p(n - (L - 1))$$

This expression calls for a few comments. The signal $x_p(n)$ is not exactly the one produced by the mother’s heart, because the latter is subjected, according to (6.1), to a delay $n_p$ due to the propagation through the thoracic tissue.
Because the propagation times are unknown, \( n_p \) may very well be greater than \( n_a \). In that case, \( x_p(n) \) is delayed with respect to \( v_M(n) \). This is why we planned for an “anticausal” part represented by the terms \( h(-M), \ldots, h(-1) \). However, if \( n_p < n_a \), which certainly is the case in this experiment, the signal \( v_M(n) \) is delayed with respect to \( x_p(n) \) and we have to make sure that the coefficients \( h(-M), \ldots, h(-1) \) are also almost null.

In the end, the two signals \( x_p(n) \) and \( x_v(n) \), picked up off the mother’s chest and abdomen, are such that:

\[
x_v(n) = h_{-M}x_p(n + M) + \cdots + h_{-1}x_p(n + 1) + h_0x_p(n) + \cdots + h_{L-1}x_p(n - L + 1) + c_F(n)
\]

where \( c_F(n) \) represents the signal we are trying to determine, originating from the heart of the fetus. We are going to estimate the coefficients \( h_i \) in such a way as to extract \( c_F(n) \) from the signals \( x_p(n) \) and \( x_v(n) \).

**Example 6.1 (Extracting the signal \( c_F(n) \))**

The first part of our work consists of identifying the abdominal transfer:

1. Establish that estimating the linear sequence \( \mathbf{h} = [h_{-M} \ldots h_0 \ldots h_{L-1}] \)
   is equivalent to a linear problem of the type:
   \[
   \mathbf{x}_v = \mathbf{X}_p \mathbf{h} + \mathbf{c}_F
   \]
   where \( \mathbf{x}_v \) and \( \mathbf{X}_p \) are respectively a vector and a matrix with the adequate sizes, constructed from the observations. \( \mathbf{c}_F \) is a vector that represents the signal originating from the heart of the fetus.

2. Using the ordinary least squares method, find the expression of an estimate of \( \mathbf{h} \). Use the result to find the estimate of the signal \( c_F(n) \).

3. Based on the previous result, write a function \( \text{extract}(\mathbf{X}_p, \mathbf{X}_v, M, L) \) that estimates \( h_{-M}, \ldots, h_0, \ldots, h_{L-1} \) and infers an estimate of the signal \( c_F(n) \) originating from the heart of the fetus.

**Comment:** The method considered for the extraction of the signal \( c_F(n) \) assumes that the useful information concerning the signal \( c_F(n) \) (such as its frequency, but also other characteristics useful to the practitioner) is not contained in the space generated by the columns of the matrix \( \mathbf{X}_p \). The likelihood of this hypothesis can really be evaluated only by whether or not the results are relevant. If, for whatever reason, a part of the signal originating from the heart of the fetus actually does belong to the space generated by the columns of \( \mathbf{X}_p \), then this signal is impossible to extract using the suggested method. If this signal happens to be useful to the practitioner, then another separation method will have to be considered. In this case, for the estimation of the cardiac frequency, the results obtained are quite satisfactory.
HINTS:

1. The observed sequences are indexed from 1 to $N$. If we stack the expressions:

$$x_v(n) = h_M x_p(n + M) + \cdots + h_1 x_p(n + 1) + h_0 x_p(n) + \cdots + h_{L-1} x_p(n - L + 1) + c_F(n)$$

for $n \in \{L, L+1, \ldots, N-M\}$ and if we use a vector notation, we get:

$$x_v = X_p h + c_F$$

where $x_v = [x_v(L) \cdots x_v(N-M)]^T$ and where:

$$X_p = \begin{bmatrix} x_p(L + M) & \cdots & x_p(1) \\ \vdots & \ddots & \vdots \\ x_p(N) & \cdots & x_p(N - L - M + 1) \end{bmatrix}$$

is a Toeplitz matrix constructed from the observations $x_p(n)$.

2. We infer that $h = X_p^# x_v$. $X_p^#$ refers to the pseudo-inverse of $X_p$ which can be obtained, either by using the \texttt{pinv(Xp)} function, or by typing $h = Xp \backslash x_v$. Once $h$ has been estimated, we can find an estimation of $c_F$ using the expression $c_F = x_v - X_p X_p^# x_v$.

The processing is performed by the program:

```
%===== separecg.m
% The file FOETUS.DAT contains:
% xp(2500 * 1) (chest sensor),
% almost equal to the mother's EKG,
% xv(2500 * 1) (abdominal sensor)
% equal to the fetus's EKG plus
% the filtered mother's EKG: Xv = h * Xp + cf
% Sampling frequency = 300 Hz
load fetus.dat
xp=fetus(:,1)-mean(fetus(:,1));
xv=fetus(:,2)-mean(fetus(:,2));
N=length(xv);
%===== estimation of h
L=20;  \% L causal
M=3;  \% M anticausal
Xv=xv(L:N-M);
col= xp(L+M:N); lig= xp(M+L:-1:1);
Xp=toeplitz(col,lig);
h=Xp \ XV; \% resolution
cf=Xv-Xp*h; \% fetal heart beats
%===== displaying the results
```
Nmax=1000; indx=[1:Nmax];
subplot(311); plot(xp(indx)); grid
subplot(312); plot(xv(indx)); grid
subplot(313); plot([zeros(L-1,1);cf(L:Nmax)]); grid

We represented in Figure 6.4, in the bottom graph, the signal extracted after processing. As you can see, the signal originating from the mother has been correctly extracted from the signal picked up by the abdomen sensor (middle graph). However, you can make out in some places the presence of a very faint residue of the mother’s heart beats in the bottom graph.

Figure 6.4 – EKG signals: at the bottom, the EKG signal of the fetus, extracted from the abdominal signal (middle graph) using the mother’s EKG (top graph)

6.1.3 Estimating cardiac rhythms

We now have to estimate the fundamental frequency of the EKG signal assumed to be periodic. We have already encountered this problem with pitch detection in speech, and we used a correlation measurement to perform this estimation. Here, we are going to give a theoretical justification for a least squares approach.

Theoretically, if a centered signal \( s(n) \) is periodic with period \( P \), the func-
tion defined by:
\[
g(k) = \lim_{K \to +\infty} \frac{\sum_{n=-K}^{+K} s(n+k)s(n)}{\sqrt{\sum_{n=-K}^{+K} s^2(n)}\sqrt{\sum_{n=-K}^{+K} s^2(n+k)}}
\]  

(6.2)

is itself periodic with period \(P\). It reaches its maximum in \(k = 0\), and reaches it again for the various multiples of \(P\).

**Example 6.2 (Measuring the fundamental frequency)**  
Consider a signal \(s(n)\), with \(n = 1 \ldots N\), periodic with period \(P\), and let us construct the two length \(L\) vectors \(v_0 = [s(1) \ldots s(L)]^T\) and \(v_k = [s(k+1) \ldots s(L+k)]^T\). If \(k\) is a multiple of the period \(P\), we will have:
\[
\frac{v_k}{\|v_k\|} = r(k) \frac{v_0}{\|v_0\|} + \varepsilon
\]

where the coefficient \(r(k)\), which is equal to \(+1\) in the ideal case, is used in the sequence to account for the slight variability of the signal’s amplitude. Let \(w_0 = v_0/\|v_0\|\) and \(w_k = v_k/\|v_k\|\). The previous equation used as a model for the periodicity of the signal \(s(n)\) can be written:
\[
w_k = r(k)w_0 + \varepsilon
\]  

(6.3)

Notice that \(\|w_0\| = \|w_k\| = 1\).

1. Using the least squares method, determine the estimator \(\hat{r}(k)\) of \(r(k)\) that minimizes the norm of the error \(\varepsilon\).

2. Show that \(|r(k)| < 1\). What does the limit case where \(r(k) \lesssim +1\) signify? Use this interpretation to determine a method for estimating the period \(P\) of the signal \(s(n)\). How must the length \(L\) of the vector \(v_0\) be chosen?

3. If the rhythm that has to be estimated is between 80 and 200 beats per minute, and if the sampling period is equal to \(1/300\) s, determine the range of values that must be chosen for \(P\).

4. The previous estimate leads to a value of the period, expressed in seconds, which is a multiple of the sampling period. How can this accuracy be improved?

5. Write a *MATLAB®* function that estimates, using the previous method, a signal’s fundamental frequency.
Hints:

1. Directly applying the least squares formula to equation (6.3) leads to the following estimation of $r(k)$:

$$
\hat{r}(k) = \frac{1}{w_0^T w_0} w_0^T w_k = \frac{v_0^T v_k}{\|v_0\|\|v_k\|}
$$

$$
= \frac{\sum_{n=1}^L s(n+k)s(n)}{\sqrt{\sum_{n=1}^L s^2(n)}\sqrt{\sum_{n=1}^L s^2(n+k)}}
$$

This expression should be compared with the expression (6.2) defining $g(k)$. We can infer a method for estimating $P$:

(a) Calculate the sequence $\hat{r}(k)$ by varying $k$ over a range of values between $P_m$ and $P_M$.

(b) The maximum of the resulting sequence is determined. If this maximum is close enough to 1, the signal is said to be periodic, the maximum’s argument is chosen as the period.

The choice of $L$ is essentially related with the practical duration $N$ for which it is reasonable to assume that the signal is periodic in a stationary sense. Obviously, if we wish to detect the possible fluctuations of the rhythm, the windows have to be chosen smaller. Once $N$ is chosen, the square deviation of the estimation decreases as $L$ increases. Hence, if $P_M$ refers to the maximum value for the period analysis, we can choose $L = N - P_M$.

2. Using the Schwarz inequality, we have to check that $r(k)$ has a modulus smaller than 1. When $r(k) \lesssim +1$, $v_0 \approx +v_k$. We can then consider that the signal is periodic and that the period is a sub-multiple of $k$. In the opposite case, the signal is likely not to be periodic. This result leads to a procedure for detecting and estimating the pitch. $r(k)$ is calculated with $k$ belonging to an exploration range $(P_m, P_M)$ of possible values. If the maximum value of $r(k)$ is greater than a threshold $\rho$ chosen beforehand, the signal is considered periodic. In this case, the first maximum of $r(k)$, greater then $\rho$, leads to an estimation of the period $P$. The value of $\rho$ is chosen experimentally based on a large number of observations of the signals to be processed.

3. For a rhythm of $B$ beats per minute, that is $b = B/60$ beats per second, the period $P$ expressed as a number of points is given by the integer part of $bF_s$, where $F_s$ refers to the sampling frequency. As such, to explore the pulse range $(B_m, B_M)$, we have to explore the range of values of $k$ defined by $(\text{fix}(B_m*F_s/60):\text{fix}(B_M*F_s/60))$. 

4. Notice that the values that can be obtained for the period, expressed in seconds, are multiples of $1/F_s$. Therefore, the determination accuracy of the correlation function’s maximum depends on the choice of the sampling rate, but because the signal is assumed to have been properly sampled, it can be interpolated with a rate $R$ and thus increase the evaluation accuracy of the maximum. Of course, in the same manner as with the frequency resolution, this increases the accuracy with which the $x$-coordinate of the maximum is obtained, but *in no way does it enhance*, in the presence of noise, the mean square error, which is related to the random position of the maximum.

5. The `f0cor.m` function detects and estimates the fundamental frequency:

```matlab
function [F0,corr]=f0cor(sn,Fs,R,thr,Fmin,Fmax)
%!==================================================!
%! SYNOPSIS: [F0,corr]=F0COR(sn,Fs,R,thr,Fmin,Fmax)!
%! sn = signal from which the freq. is extracted!
%! Fs = sampling frequency (Hz)!
%! R = oversampling factor!
%! thr = threshold!
%! Fmin = min. frequeny (Hz)!
%! otherwise Fmin=2*Fs/longueur(sn);!
%! Fmax = max. frequency (Hz)!
%! otherwise Fmax=Fs/2-Fmin;!
%! F0 = fundamental frequency (Hz)!
%! corr = correlation sequence!
%!==================================================!

sn=interp(sn,R); Fs=R*Fs;
N=length(sn); sn=sn(:); sn=sn-mean(sn);
lagmin=fix(Fs/Fmax); lagmax=fix(Fs/Fmin);
corr=zeros(1,lagmax-lagmin+1);
%===== the effects of the window's size can be tested
% by taking wlg<wlgmax=N-lagmax
wlg=N-lagmax; v0=sn(1:wlg);
for ii=lagmin:lagmax
    vP=sn(ii:ii+wlg-1);
    corr(ii-lagmin+1)=(v0'*vP)/sqrt((v0'*v0)*(vP'*vP));
end
[niv1, indmax]=max(corr);
if niv1<thr
    pf0=0; F0=NaN;
    return
else
    for ii=lagmin+1:lagmax
        if corr(ii-lagmin+1)>niv1*0.9
            while corr(ii-lagmin+1)>corr(ii-lagmin)
```
ii=ii+1;
end
pf0=ii-2; F0=Fs/pf0;
return
else
F0=Fs/(indmax+lagmin-1);
end
end

In the method considered for measuring the pitch in this example, the correlation function shows maxima in the multiples of the period we wish to determine. Therefore, when scanning the range of possible values, several of these maxima can be encountered. Because of measurement uncertainty, it is possible that the highest of these maxima does not correspond to the period. Therefore, we must search for the possible maxima at sub-multiples of the one corresponding the highest of these maxima. The f0cor.m function we have given performs such an operation by searching for other possible maxima greater than 0.9 times the highest maximum. Other more efficient processes can be designed to solve this problem (see further on in this paragraph).

Figure 6.5 shows the levels of the correlation functions, in their respective exploration ranges, for the mother’s EKG (top graph), and for the EKG of the fetus (bottom graph). These graphs were obtained using the following program:

```matlab
%===== extrythm.m
% Rhythm estimation
% This program uses signals from separreg.m
% with the file fetus.mat
% Uses: f0cor.m
pulsemin=50;  %==== beats per mn
pulsemax=300; %==== beats per mn
%===== oversampling ratio
R=2;
maxcor_apriori=0.25;
[F_mother corr_mother]=...
f0cor(xp,Fe,R,maxcor_apriori,pulsemin/60,pulsemax/60);
[F_fetus corr_fetus]=...
f0cor(cf,Fe,R,maxcor_apriori,pulsemin/60,pulsemax/60);
%===== displaying the results
disp('******************************************')
disp(sprintf('* Pulses (mother): %5.2f',60*F_mother));
disp(sprintf('* Pulses (fetus) : %5.2f',60*F_fetus));
disp('******************************************')
%===== displaying the results
```
Figure 6.5 – Levels of the correlation function for the mother’s EKG (top graph), and for the EKG of the fetus (bottom graph)

```
lagminM=fix(60*Fe*R/pulsemax);
lagminF=fix(60*Fe*R/pulsemax);
MinCM=min(corr_mother);
LCM=length(corr_mother); MaxCM=max(corr_mother);
subplot(211);
plot((lagminM:LCM+lagminM-1)/R,corr_mother)
grid; hold on;
plot(Fe/F_mother*[1 1]+1,[min(corr_mother) MaxCM],':');
hold off
%=====
MinCF=min(corr_fetus); MaxCF=max(corr_fetus);
subplot(212);
plot((lagminF:length(corr_fetus)+lagminF-1)/R,...
corr_fetus)
grid; hold on;
plot(Fe/F_fetus*[1 1]+1,[MinCF MaxCF],':');
hold off
```

Running the program returns:

```
****************************
* Pulse (mother): 97.56
* Pulse (fetus) : 160.71
****************************
```

We saw previously that the presence of several maxima in the correlation we
are measuring can pose a problem. One way of eliminating this error consists of calculating the spectral product defined by:

\[ P(f) = \prod_{k=1}^{K} |X(e^{2j\pi kf})|^2 \text{ where } f \leq 1/2K \]  

(6.4)

where \( X(f) \) is the DTFT of a block of the signal. If the latter is periodic with the fundamental frequency \( f_0 = F_0/F_s \), its spectrum will show high amplitude peaks in frequencies that are multiples of the frequency \( f_0 \). Hence the product \( P(f_0/2) \) of the spectrum’s values calculated in multiples of \( f_0/2 \) will have a small value. This is due to the almost null values associated with the frequencies that are odd multiples of \( f_0/2 \). This is not the case of the product \( P(f_0) \) calculated in \( f_0 \), which accumulates the spectrum’s amplitudes in \( f_0, 2f_0, \) etc. Therefore, the function \( P(f) \) allows us to do away with the ambiguity in \( f_0/2 \): if the frequency estimate \( \hat{f} \) is close to \( f_0/2 \), the value of \( P(\hat{f}) \) will be much smaller than \( P(2\hat{f}) \). Comparing these two values then allows us to choose \( 2\hat{f} \) as the fundamental frequency. However, in order for this method to work well, it requires precise calculation of the spectrum, especially if the spectrum has very sharp peaks, which implies long calculation times.

**COMMENT:** when the signal we wish to estimate the pitch of contains much higher frequencies than the pitch, we can also, as it is done with some speech encoders, perform a low-pass filtering before estimating the pitch. For a narrow band signal around a central frequency \( F_c \), that is a signal in the form of a brief oscillating impulse, we can also replace the signal \( x \) with its *envelope*.

### 6.2 Extracting the contour of a coin

We are going to try to determine the ellipse representing the contour of a coin (Figure 6.6). The process can be achieved in two very different ways: the first one performs an approximate extraction of the contour, then applies the least squares method. The second performs an approximate extraction of the elliptical disk representing the coin, then extracts the contour using a correlation method.

**Example 6.3 (Extracting the elliptical disk)** Based on the image of a coin that you will save in “levels of gray”, write a program that approximately extracts the elliptical disk corresponding to the inside of the coin, as it is shown in Figure 6.6.

**HINTS:** type:

```matlab
% preprocesscoin.m
clear all; close all;
```
load mcoin2; % or imread...
figure(1); set(gcf,'color',[1 1 1])
subplot(221); imagesc(pixc); Spix=size(pixc);
colormap('gray'); axis('image'); title('Coin')
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])

%===== threshold
minpix=100; ipx0=find(pixc < minpix); yim0=ones(Spix)*255;
yim0(ipx0)=zeros(size(ipx0));
subplot(222); imagesc(yim0); colormap('gray');
axis('image'); title('Threshold')
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])

%===== gaussian filtering
hg=moygauss(5); yim0g=filter2(hg,yim0);
subplot(223); imagesc(yim0g); title('Gaussian filter')
colormap('gray'); axis('image')
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])

%===== threshold
minpix2=170; ipx02=find(yim0g < minpix2);
yim02=ones(Spix)*255; yim02(ipx02)=zeros(size(ipx02));
subplot(224); imagesc(yim02); colormap('gray');
axis('image'); title('Threshold')
set(gca,'Xcolor',[0 0 0],'Ycolor',[0 0 0])
save('yim02','yim02')

with:

function hg=moygauss(sigma)

%!=================================================================================
%! LOW-Pass Gaussian filter !
%! SYNOPTIC: hg=M5YGAUSS(sigma) !
%! sigma = standard deviation !
%! hg = coefficients !
%!=================================================================================

xx=-sigma*5:sigma*5;
g=exp(-xx.*xx / (2*sigma*sigma)); % gaussian
hg=g(g>max(g)*.005); hg=hg/sum(hg(:));

return

Exercise 6.1 (Ellipse contour: the least squares method)  (see p. 231)
Remember that an ellipse can be described by the equation:

$$ax_1^2 + bx_2^2 + cx_1x_2 + dx_1 + ex_2 - 1 = 0$$

with $$ab - c^2/4 > 0$$  \hfill (6.5)
where \((x_1, x_2)\) represents the point in the plane with the \(x\)-coordinate \(x_1\) and the \(y\)-coordinate \(x_2\). A first process leads to the least squares estimation over \(N\) points of the ellipse, to the coefficients \(a, b, c, d\) and \(e\) of equation (6.5).

1. Write a program, based on the program in example 6.3, which extracts an approximate contour of the coin. You can use the “numerical differentiation” function \((\text{diff})\) or a Gaussian differentiation function \((\text{dergauss})\).

```matlab
function hd=dergauss(sigma)
%!==================================================!
%! Gaussian derivative filter !
%! SYNOPSIS: hd = DERGAUSS(sigma) !
%! sigma = Standard deviation !
%! hd = filter with (N*N)-PSF !
%!==================================================!
rho=[-sigma*3:sigma*3]; N=length(rho);
rp=1.4; s2=2*sigma^2; s22=s2*rp*rp;
idx= ([1:N]-(N+1)/2)' * ones(1,N); idy=idx';
idxa=[1:N]' * ones(1,N); idya=idxa';
%%%inds(1,:)=reshape(idx,1,N*N); inda(1,:)=reshape(idxa,1,N*N);
inds(2,:)=reshape(idy,1,N*N); inda(2,:)=reshape(idya,1,N*N);
rho2=sum(inds .* inds); rho=sqrt(rho2);
for k=1:N*N
    g1=(1/sigma)*exp(-rho2(k) / s2);
    g2=(1/sigma/rp)*exp(-rho2(k) / s22);
    hd(inda(2,k),inda(1,k))=g1-g2;
```

**Figure 6.6 – Result of pre-processing**
2. Using the “least squares method”, determine the coefficients \(a, b, c, d,\) and \(e\) of the ellipse closest to the contour that we found. Write a function that plots the ellipse defined by equation (6.5).

Do so by rewriting the equation of the ellipse in the form \((x - x_0)^T E (x - x_0) = \gamma\), where \(x = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^T\), then determine the expressions of \(E, x_0\) and \(\gamma\) as functions of \(a, b, c, d,\) and \(e.\)

Exercise 6.2 (Ellipse contour: the covariance method) (see p. 232)
Consider a sequence of \(N\) points on the plane, described by \(N\) random variables \(x_1, \ldots, x_N\) assumed to be independent and uniformly distributed on the elliptical disk defined by its contour:

\[(x - \mu)^T M^{-1} (x - \mu) = 1\]

where \(M\) refers to a positive matrix and \(\mu\) is the center of the ellipse. Let:

\[y_n = M^{-1/2} (x_n - \mu)\]  \hspace{1cm} (6.6)

You can easily check that the sequence \(y_n\) constitutes a sequence of independent random variables uniformly distributed on the circular disk with a unit radius.

1. Determine the expression of the vector \(\mathbb{E}\{y_1\}\) and of the matrix \(\mathbb{E}\{y_1 y_1^T\}.\)

2. According to the law of large numbers, if \(y_n\) refers to a sequence of independent random vectors with the mean \(\nu\) and the covariance \(C\) then, when \(N\) tends to infinity:

\[\frac{1}{N} \sum_{n=1}^{N} (y_n - \hat{\nu}_N) (y_n - \hat{\nu}_N)^T \xrightarrow{a.s.} C\]

where \(\hat{\nu}_N = N^{-1} \sum_{n=1}^{N} y_n.\) By applying the law of large numbers to the sequence \(y_1, \ldots, y_N,\) infer that:

\[M^{-1} \approx \frac{4}{N} \sum_{n=1}^{N} (x_n - \hat{\mu}_N) (x_n - \hat{\mu}_N)^T\]  \hspace{1cm} where \(\hat{\mu}_N = \frac{1}{N} \sum_{n=1}^{N} x_n\)

3. Apply these results to the problem of determining the ellipse closest to the contour of the coin.
6.3 Constrained optimization and Lagrange multipliers

This section constitutes an introduction to the problem of constrained optimization and Lagrange multipliers. To illustrate the issue, we have chosen to present the Modern Portfolio Theory put forward by H. Markowitz.

In this section, the notation $h(x) \leq 0_p$ is used for:

\[
\begin{aligned}
&h_1(x_1, \ldots, x_n) \leq 0 \\
&\vdots \\
&h_p(x_1, \ldots, x_n) \leq 0 \\
\end{aligned}
\]  

(6.7)

6.3.1 Equality-constrained optimization

Consider the following problem:

\[
\begin{aligned}
\min & J(x) \\
\text{s.t.} & \quad h(x) = 0_p
\end{aligned}
\]  

(6.8)

where $J$ is a function of $\mathbb{R}^n$ in $\mathbb{R}$, and $h(x)$ a function of $\mathbb{R}^n$ in $\mathbb{R}^p$. We write:

\[
\mathcal{L}(x, \lambda) = J(x) + \lambda^T h(x)
\]  

(6.9)

where $\lambda \in \mathbb{R}^p$. The function $\mathcal{L}$ is called the Lagrangian of the problem and $\lambda$ the Lagrangian multiplier vector. Under very general regularity conditions, e.g. when $J$ is convex, we can show [35] that if $x_o$ is a local minimum – it satisfies:

\[
\begin{aligned}
\exists \lambda_o \in \mathbb{R}^p, x_o \in \mathbb{R}^n & \quad \text{s.t.} \\
\frac{\partial \mathcal{L}}{\partial x}(x_o) = \frac{\partial J}{\partial x}(x_o) + \lambda_o \frac{\partial h}{\partial x}(x_o) &= 0 \\
h(x_o) &= 0
\end{aligned}
\]  

(6.10)

It is worth noting that, according to the first equation in (6.10), the normals to the functions $J$ and $h$ coincide at the point $x_o$. Note that the second equation can be seen as the cancellation of $\mathcal{L}$ with respect to $\lambda$.

For simplicity’s sake, the optimum will be denoted either by an asterisk (*), or by the letter o.
Example 6.4 (Production of goods)
Let $G$ be the gain (or profit) obtained by the sale of two goods with respective unit prices $p_1$ and $p_2$. Thus, we have $G = p_1 x_1 + p_2 x_2$. We seek to maximize a function of the form $x_1^\alpha x_2^\beta$ (Cobb–Douglas function [10]), and therefore minimize $J(x_1, x_2) = -x_1^\alpha x_2^\beta$, under the constraint $G - p_1 x_1 - p_2 x_2 = 0$. It is shown that the function $J$ is convex if $\alpha \geq 0$, $\beta \geq 0$ and $\alpha + \beta \leq 1$. The Lagrangian is written:

$$L(x_1, x_2, \lambda) = -x_1^\alpha x_2^\beta + \lambda(G - p_1 x_1 - p_2 x_2)$$

At the minimum, we have:

$$-\alpha x_1^\alpha - 1 x_2^\beta - p_1 \lambda^* = 0 \Rightarrow \lambda^* = -\frac{\alpha x_1^\alpha x_2^\beta}{p_1}$$

$$-\beta x_1^\alpha x_2^\beta - p_2 \lambda^* = 0 \Rightarrow \lambda^* = -\frac{\beta x_1^\alpha x_2^\beta}{p_2}$$

$$G - p_1 x_1^* - p_2 x_2^* = 0$$

From the first two expressions, we can deduce:

$$\frac{\alpha x_1^\alpha - 1 x_2^\beta}{p_1} = \frac{\beta x_1^\alpha x_2^\beta}{p_2} \Rightarrow x_2^* = x_1^* \frac{\beta p_1}{\alpha p_2}$$

With the constraint, we obtain (Figure 6.7):

$$x_1^* = \frac{\alpha G}{(\alpha + \beta)p_1} \text{ and } x_2^* = \frac{\beta G}{(\alpha + \beta)p_2}$$

In Figure 6.7, we have plotted the curves defined by $J(x_1, x_2) = \text{constant}$ and also the normal to the constraint line the equation of which is $p_1 x_1^* + p_2 x_2^* = G$. This normal is normal only to one of the curves, which is defined by $J_o = 40.4886$ at the coordinate $x_1 = 7.2$ and $x_2 = 72$.

Example 6.5 (Minimization with quadratic constraints)
Consider the following problem:

$$\begin{cases} 
\text{min } J(x) = \frac{1}{2} x^T Q x \\
\text{with } x^T x = 1 
\end{cases}$$

where $Q$ is a positive, invertible matrix, of dimension $n$ and $x \in \mathbb{R}^n$. Determine the solution.

Hints: the Lagrangian of the problem is written as:

$$L(x) = \frac{1}{2} x^T Q x + \lambda(1 - x^T x)$$
Figure 6.7 – Optimization of utility function; $\alpha = 0.25, \beta = 0.75, p_1 = 10, p_2 = 3, G = 288$

and therefore:

$$\frac{\partial}{\partial x} L(x) = Qx - \lambda x = 0$$

Consequently, $Qx = \lambda x$, so $x$ is one of the eigenvectors of $Q$. We deduce from this that the minimum of $J(x)$ must be equal to $\frac{1}{2}\lambda$ and therefore $2J(x)$ is necessarily equal to an eigenvalue. As $Q$ is positive, it is diagonalizable and its eigenvalues are positive. It follows from this that the minimum we seek is given by the lowest eigenvalue of $Q$, and $x$ is the eigenvector associated therewith. We can verify that these expressions are still valid when $x \in \mathbb{C}^n$.

**Example 6.6 (Maximum entropy in a system under mean constraints)**

Consider a sequence $p_k \geq 0$ where $k \in \mathbb{N}$ and such that $\sum_k p_k = 1$. The series $p = \{p_k\}$ is interpreted as a probability distribution. We use the term “entropy of the series” for the concave function:

$$h(p) = - \sum_{k=0}^{+\infty} p_k \log p_k \quad (6.11)$$

We note that $0 \leq p_k \leq 1$ and therefore $h \geq 0$. We wish to solve the following problem:

$$\begin{cases}
\max h(p) \\
\sum_{k=0}^{+\infty} p_k = 1 \\
\sum_{k=0}^{+\infty} kp_k = m
\end{cases}$$
Note that \( m \geq 0 \). Maximizing \( h(p) \) is tantamount to minimizing the convex function \( J(p) = -h(p) \).

HINTS: the Lagrangian is written as:

\[
\mathcal{L}(p, \lambda) = \sum_k p_k \log p_k + \lambda_1 \left( 1 - \sum_k p_k \right) + \lambda_2 \left( m - \sum_k k p_k \right)
\]

The cancellation of the \( k \)-th component of the gradient with respect to \( p \) is written as:

\[
\frac{\partial p_k \mathcal{L}(p, \lambda)}{\partial p_k} = 1 + \log p_k - \lambda_1 - \lambda_2 k = 0
\]

hence:

\[
p_k = e^{-1 + \lambda_1 + \lambda_2 k} = \mu \gamma^k
\]

\( \mu \) and \( \gamma \) are determined by satisfying the constraints:

\[
\sum_k p_k = \frac{\mu}{1 - \gamma} = 1 \quad \text{and} \quad \sum_k k p_k = \frac{\mu \gamma}{(1 - \gamma)^2} = m
\]

Therefore, \( \gamma = \frac{m}{m + 1} \), \( \mu = (1 - \gamma) \) and \( p_k = (1 - \gamma) \gamma^k \).

### 6.3.2 Quadratic problem with linear inequality constraints

Consider the following quadratic problem:

\[
\begin{align*}
\text{min} & \quad \frac{1}{2} x^T Q x + w^T x \\
\text{subject to} & \quad A x + b = 0, \\
& \quad C x + d \leq 0
\end{align*}
\]

where \( Q \) is a strictly positive matrix of dimensions \( n \times n \), \( w \) is a vector of length \( n \), \( A \) a positive matrix of dimensions \( p \times n \), \( b \) a vector of length \( p \), \( C \) a matrix of dimensions \( q \times n \) and \( d \) a vector of length \( q \).

The Lagrangian is:

\[
\mathcal{L}(x, \lambda, \mu) = \frac{1}{2} x^T Q x + w^T x + \lambda^T (A x + b) + \mu^T (C x + d)
\]

where \( \lambda \in \mathbb{R}^p \) and \( \mu \in \mathbb{R}^{q+} \).

We can show that \( x_o \) is a solution to the problem if and only if:

\[
\exists \lambda_o \in \mathbb{R}^p, \mu_o \in \mathbb{R}^{q+}, x_o \in \mathbb{R}^n ? = 0 \quad \text{for all} \quad i = 1, \ldots, q
\]

where \( C_i \) denotes the row \( i \) in the matrix \( C \).
The set of equations (6.13) are called the Karush–Kuhn–Tucker (KKT) conditions. The condition $\mu_{i,o} \geq 0$ is called the duality constraint. The last condition, known as complementarity, implies that if $\mu_{o,i}$ is strictly positive, then the constraint of inequality becomes one of equality. The constraint is said to be saturated or active.

Example 6.7 Determine the solution to the problem:

\[
\begin{align*}
\min J(x) &= \frac{1}{2} x^T Q x \\
\quad x^T u &= 1, \\
\quad x^T v &\leq a
\end{align*}
\]

where $a \in \mathbb{R}$, where $x \in \mathbb{R}^3$, where $u = [1 \quad 1 \quad 1]^T$, where $v \in \mathbb{R}^3$, and where $Q$ is a strictly-positive matrix of dimensions $3 \times 3$, with real elements. Note that we can replace $x_3$ with $(1 - x_1 - x_2)$ in $J$, numerically calculate $J$ in the plane $(x_1, x_2)$ and seek the minimum which is below the line defined by $v_1 x_1 + v_2 x_2 + v_3 (1 - x_1 - x_2) = a$.

Hints: The Lagrangian is written as:

\[
\mathcal{L}(x, \lambda, \mu) = \frac{1}{2} x^T Q x + \lambda (x^T u - 1) + \mu (x^T v - a)
\]

Using the KKT conditions, we obtain:

\[
\begin{align*}
\left\{ \begin{array}{l}
Q x + \lambda u + \mu v = 0 \\
\lambda (x^T u - 1) = 0 \\
\mu (x^T v - a) = 0
\end{array} \right. \quad (6.15)
\]

and therefore:

\[
x = - (\lambda Q^{-1} u + \mu Q^{-1} v)
\]

We determine $\mu$ by applying the condition of complementarity. Thus, we must distinguish two cases:

- the constraint $x^T v - a \leq 0$ is not active, and therefore $\mu = 0$ and $x = -\lambda Q^{-1} u$. $\lambda$ is determined by writing that $u^T x = 1$, which gives us $x = (u^T Q^{-1} u)^{-1} Q^{-1} u$, and $2 J_1 = (u^T Q^{-1} u)^{-1} > 0$.
- the constraint $x^T v - a \leq 0$ is made active, so $x^T v - a = 0$ and:

\[
x = - (\lambda Q^{-1} u + \mu Q^{-1} v)
\]
and to determine $\lambda$ and $\mu$ we apply the equality constraint and active inequality constraint, which gives:

\[
\begin{align*}
\lambda u^T Q^{-1} u + \mu u^T Q^{-1} v &= -1 \\
\lambda v^T Q^{-1} u + \mu v^T Q^{-1} v &= -a
\end{align*}
\]

where we have set:

\[
A = u^T Q^{-1} u, \quad B = v^T Q^{-1} u \quad \text{and} \quad C = v^T Q^{-1} v
\]

Note that $A \geq 0$, $C \geq 0$ and that, according to Schwarz’s inequality, $AC - B^2 > 0$. By substituting the value into $J$, the vector $Qx$ given by the first expression of (6.14), we obtain:

\[
2J_2 = x^T Qx
= (\lambda Q^{-1} u + \mu Q^{-1} v)^T (\lambda u + \mu v)
= (\lambda^2 A + \mu^2 C + 2\lambda \mu B)
= -(\lambda + \mu a)
= [1 \ a] \begin{bmatrix} A & B \\ B & C \end{bmatrix}^{-1} [1] > 0
\]

To conclude, we need to take the lower of the two values $2J_1 = 1/A$ and $2J_2$.

In Figure 6.8, we have illustrated the function $J$ in relation to the plane $x^T u = 1$ and calculated on a value grid.

**Figure 6.8** – Function $J$ from example 6.7 in relation to the plane $(x_1, x_2)$ where $x_3 = 1 - x_1 - x_2$. The circle indicates the minimum obtained analytically by the Lagrangian and the cross denotes the minimum obtained on a grid in the plane $x^T u = 1$
We can see in example 6.7 that, when we have inequality constraints, we must review all possible cases of activation of the constraints. If \( n \) is large, this may lead to an impossible exhaustive analysis. This is why numerical methods have had to be developed. However, the study of these algorithms goes beyond the bounds of this book.
6.3.3 Portfolio optimization

The placement of investments on the financial markets raises the following two questions:

1. Which stocks should we choose?

2. What proportion of our portfolio should we assign to each stock holding?

The purpose of the portfolio theory is to answer these two questions. It was a long time before financial theory ventured into the notion of uncertainty. The indisputable pioneer in this area was Harry Markowitz: his first article on the subject was published in 1952 [23] and his first book in 1959 [24]. For his work, he received the Nobel Memorial Prize in Economic Sciences in 1990.

Markovitz’s work was opposed to the idea which was widely accepted at the time, i.e. that the investor chooses the portfolio distribution which will generate the highest revenue. Indeed, if this were the case, then investors would only ever choose one investment: the one with the highest rate of return. This runs counter, though, to the phenomenon observed in real life, which is the diversification of portfolios. To explain this mechanism of diversification, Markovitz introduced a probabilistic model which introduces the notions of return and risk. On the mathematical level, he constructed a fairly straightforward problem of quadratic optimization. The originality of the work essentially lies in the application of this engineering model to the world of finance.

Generally speaking, all investors are, by nature, “risk-averse”. However, they are prepared to take more risks in exchange for a better return. Markowitz’s idea was to link the risk to the variability of the assets.

The model is static, in the sense that we consider only two periods:

- the initial period, 0, when the investor is constructing his or her portfolio;

- and the final period, $T$, when the yields from the assets are recorded and the final value of the portfolio is determined.

We use the following notation:

- $n$ for the number of assets (shares) making up the financial market;

- $B_{0,i}$ and $B_{T,i}$ for the respective prices of the asset $i$, at moments which are a length of time $T$ apart;

- the return $\rho_i$ on the asset $i$ for the period $T$ is then defined by:

$$B_{T,i} = B_{0,i}(1 + \rho_i)^T \Rightarrow \rho_i = -1 + \left( \frac{B_{T,i}}{B_{0,i}} \right)^{1/T}$$

(6.16)
Sometimes we add to $B_{T,i}$ any revenue perceived in the form of dividends. In any case, $\rho_i$ is a number which may be positive or negative. Evidently, its modulus may be greater than 1. However, over short lengths of time, this number is generally small. It is typically to multiply it by 100 so as to be able to express it as a percentage.

Markowitz’s model states that the returns $\rho_i$ of assets on the market are random variables. We note the mean vector:

$$m = \mathbb{E}\{\rho\} = \left[ \mathbb{E}\{\rho_1\} \ldots \mathbb{E}\{\rho_n\} \right]^T \in \mathbb{R}^n \quad (6.17)$$

and covariance matrix:

$$Q = \begin{bmatrix}
cov(\rho_1, \rho_1) & cov(\rho_1, \rho_2) & \cdots & cov(\rho_1, \rho_n) \\
cov(\rho_2, \rho_1) & cov(\rho_2, \rho_2) & \cdots & cov(\rho_2, \rho_n) \\
\vdots & \vdots & \ddots & \vdots \\
cov(\rho_n, \rho_1) & cov(\rho_n, \rho_2) & \cdots & cov(\rho_n, \rho_n)
\end{bmatrix}$$

Remember that a covariance matrix is positive – particularly its determinant and its eigenvalues are positive. If $Q$ is strictly positive, its determinant is strictly positive and its inverse exists, which we will assume.

Hereafter, we will consider that the vector $m$ and the matrix $Q$ are known. In practice, these quantities are estimated on the basis of observations of the market over long periods.

Let $W$ be the investor’s original wealth and $B_i \geq 0$ the sum invested in the asset $i$. The proportion of the asset $B_i$ in the portfolio is defined by:

$$x_i = \frac{B_i}{W} \in (0,1) \quad (6.18)$$

with the budgetary constraint:

$$\sum_i x_i = 1, \text{ or } x^T u = 1 \quad (6.19)$$

where $x = [x_1 \ldots x_n]^T$ and where $u$ is a column vector whose $n$ components have the value of 1.

The problem of the construction of a portfolio therefore lies in determining the proportions $x_i$. The rate of return of a portfolio, comprising the proportions $x_i$ of the shares on the market, is given by the random variable:

$$\Theta = \sum_{i=1}^{n} x_i \rho_i \quad (6.20)$$

whose average is:

$$\mathbb{E}\{\Theta\} = x^T m \quad (6.21)$$
and whose variance or volatility has the expression:

\[ \text{var}(\Theta) = x^T Q x \]  

(6.22)

H. Markowitz stipulates that the variability \( \sigma = \sqrt{x^T Q x} \) is the measurement of the risk. The greater the value of \( \sigma \), the greater the risk, but also the higher the potential return. Consequently, the greater the risk that investors are willing to accept, the greater will be the expected gain. In practice, a tailored survey is used to estimate the level of risk that an individual is willing to take.\(^{(1)}\) For people who have no taste for risk, the market generally includes an investment with a fixed return, risk-free, i.e. with zero volatility. Of course, the return on such an investment is low.

The optimal portfolio is defined as the solution to the problem which minimizes the risk with a level of expected return which is higher than or equal to a given level \( \rho_e \). The optimal portfolio therefore needs to be found among all the portfolios which are acceptable solutions to the problem – known as efficient portfolios:

\[
\begin{align*}
\min J(x) &= \frac{1}{2} x^T Q x \\
x^T u &= 1 \\
x^T m &\geq \rho_e \\
\forall i, \ x_i &\geq 0
\end{align*}
\]  

(6.23)

It is important to note that \( Q \) is positive, because it is a covariance matrix. Hereafter, we will suppose that it is strictly positive, and therefore invertible. Given the positive nature of \( Q \), the problem of minimization is convex and, therefore, has a global minimum which satisfies the KKT conditions (6.13).

Optimal portfolios are calculated on the basis of the market data, i.e. the vector \( m \) of the average returns and the covariance matrix \( Q \). For each value of the return \( \rho_e \) in which we are interested, we can associate with it the distribution \( x_e \) and the risk \( \sigma_e \) of the portfolio. A pair \( (\sigma_e, \rho_e) \) constitutes an efficient portfolio and the set of efficient portfolios forms a curve called the market frontier. Each point on the curve is indexed by its distribution \( x_e \).

On the basis of the market frontier, the investor determines the construction of his/her portfolio of assets depending on his/her taste for risk.

**Example 6.8** We use the notation \( J_m \) for the solution to the problem (6.23) and \( K_m \) for that of the following problem:

\[
\begin{align*}
\min \frac{1}{2} x^T Q x \\
u^T x &= 1
\end{align*}
\]  

(1) On the Internet, there are many sites where users can evaluate their own risk-acceptance.
Show that:

\[ J_m \geq \frac{1}{u^T Q^{-1} u} \]

Conclude.

Hints: Given that the problem (6.23) is subject to more stringent constraints than the problem at hand here, we have \( J_m \geq K_m \). The Lagrangian of the current problem is:

\[ L(x, \lambda) = \frac{1}{2} x^T Q x + \lambda(x^T u - 1) \]

By deriving in relation to \( x \) we obtain:

\[ Q x_o = -\lambda u \Rightarrow x_o = -\lambda Q^{-1} u \]

By applying the stress \( u^T x = 1 \), we find \( \lambda = -1/(u^T Q^{-1} u) \) and then \( x_o = Q^{-1} u / (u^T Q^{-1} u) \) and therefore

\[ K_m = \frac{1}{u^T Q^{-1} u} \Rightarrow J_m \geq \frac{1}{u^T Q^{-1} u} \]

From this, we conclude that there is a risk which depends only on the market (by way of the matrix \( Q \)), below which it is not possible to go.

Thus, we can expect a decrease of the risk:

- when the number of assets \( n \) in the portfolio increases;
- when we choose a pair of assets which vary in an inverse direction – i.e. which have a negative correlation. This is what we can see in the exercise 6.4. For a long time, investors have been using such pairs where the gain on one of the assets compensates for the losses on another. For example, we can cite the negative correlation of the share/obligation couple. Negative correlations are the cornerstone of diversification.

Exercise 6.3 (Optimization of a portfolio with two assets) (see p. 233)

Consider a portfolio containing two assets. \( Q \) is the risk matrix defined by:

\[
Q = \begin{bmatrix}
q_1 & \frac{\rho \sqrt{q_1 q_2}}{\sqrt{q_1} q_2} \\
\frac{\rho \sqrt{q_1} q_2}{\sqrt{q_1} q_2} & q_2
\end{bmatrix}
\]

where \( q_1 \) and \( q_2 \) are positive and \(-1 < \rho < 1\). We take \( m = [m_1 \quad m_2]^T \). We set:

\[
\begin{align*}
A &= u^T Q^{-1} u \\
B &= m^T Q^{-1} u \\
C &= m^T Q^{-1} m
\end{align*}
\]

Note that \( A \geq 0, C \geq 0 \) and that, in view of Schwarz’s inequality, \( AC - B^2 > 0 \). These quantities can be pre-calculated.
1. Solve the problem (6.23) of portfolio optimization.

2. Write a program to determine the market frontier.

**Exercise 6.4 (Negative correlation)** (see p. 236)

Look again at the statement of exercise 6.3 and write a program which demonstrates the decrease in risk when $\rho$ becomes negative.

**Market containing a risk-free investment**

Consider the case where the market includes a risk-free investment – i.e. with zero variance and return $\rho_f$. We set $x = [x_f \; x_a^T]$ where $x_f$ represents the proportion of the portfolio invested in the share with a fixed return rate of $\rho_f$ and $x_a = [x_{a,1} \; \ldots \; x_{a,n}]^T$ the vector of the $n$ components of the portions invested in the risky shares. The symbol $m_a$ represents the average yield vector for the risky shares. The problem of optimization can be rewritten as:

$$
\begin{align*}
\min J(x) &= \frac{1}{2} \begin{bmatrix} x_f & x_a^T \end{bmatrix} \begin{bmatrix} 0 & 0^T_n \\ 0_n & Q \end{bmatrix} \begin{bmatrix} x_f \\ x_a \end{bmatrix} \\
\begin{bmatrix} x_f & x_a^T \end{bmatrix} \begin{bmatrix} 1 \\ u \end{bmatrix} &= 1 \\
\begin{bmatrix} x_f & x_a^T \end{bmatrix} \begin{bmatrix} \rho_f \\ m_a \end{bmatrix} &\geq \rho_e \\
\forall i = 1 \cdots n, \; x_{a,i} &\geq 0 \\
x_f &\geq 0
\end{align*}
$$

$$
\begin{align*}
\min J(x) &= \frac{1}{2} x_a^T Q x_a \\
x_f + x_a^T u &= 1 \\
x_a^T m_a + x_f \rho_f &\geq \rho_e \\
\forall i = 1 \cdots n, \; x_{a,i} &\geq 0 \\
x_f &\geq 0
\end{align*}
$$

(6.24)

It should be noted that the constraint $x_f + x_a^T u = 1$, and the constraints $x_{a,i} \geq 0$ and $x_f \geq 0$, imply that $x_{a,i} \leq 1$ and $x_f \leq 1$.

If the client is particularly risk-averse then the optimal portfolio comprises only the risk-free investment and, in this case, $x_f = 1$ and the return achieved is $\rho_f$. The frontier is reduced to the single point $(0, \rho_f)$. Hereafter we will therefore suppose that the expected return $\rho_e \geq \rho_f$.

$M_a = m_a(j_M)$ is the maximum value of the returns on the risky investments. We suppose that $M_a > \rho_f$, or else the optimal solution is $x_a = 0$, $x_f = 1$ and the return is $\rho_f$ and the risk is nonexistent. Obviously, the return which is achieved satisfies:

$$
x_f \rho_f + x_a^T m_a \leq x_f \rho_f + x_a^T u M_a \leq x_f M_a + x_a^T u M_a = M_a
$$

In conclusion, the desired return $\rho_e$ satisfies $\rho_f \leq \rho_e \leq M_a$. When the desired return is $\rho_f$ then $x_f = 1$, when the desired return is $M_a$, $x_f = 0$ and
\( x_a \) has a single non-null component equal to 1, in position \( j_M \) (which is an admissible point in the sense that it satisfies the constraints and activates the constraint of return).

If \( x_f > 0 \), the optimal portfolio shifts along a line running from the point \((0, x_f)\) to a curve which depends only on the assets on the market. Let \((x_{a,o}, x_{f,o})\) be the optimal solution to the problem (6.24). The associated risk is written as:

\[
\sigma_o = \sqrt{x_{a,o}^T Q x_{a,o}}
\]

We set:

\[
z_{a,o} = \frac{x_{a,o}}{x_a^T u} = \frac{x_{a,o}}{1 - x_{f,o}}, \quad \Rightarrow z_{a,o}^T u = 1
\]  

(6.25)

\( z_{a,o} \) has a clear meaning: it represents the distribution of the part \((1 - x_{f,o})\) of the portfolio given over to risky investments. With these notations, the expression of the risk is:

\[
\sigma_o = (1 - x_{f,o}) \sqrt{z_{a,o}^T Q z_{a,o}}
\]

and the associated return is:

\[
\rho_o = x_{f,o} \rho_f + x_{a,o}^T m_a = x_{f,o} \rho_f + (1 - x_{f,o}) z_{a,o}^T m_{a,o}
\]

\[
= \rho_f + \frac{(z_{a,o}^T m_a - \rho_f)}{\sigma_{a,o}}
\]

(6.26)

where we have set \( \sigma_{a,o}^2 = z_{a,o}^T Q z_{a,o} \). Thus, we can see that the optimal return as a function of the risk is on a line with the slope:

\[
\rho = \frac{z_{a,o}^T m_a - \rho_f}{\sigma_{a,o}}
\]

called Sharpe’s ratio. The straight line defined by equation (6.26) is called the Capital Allocation Line (CAL). Using the fact that \( z_{a,o}^T u = 1 \), we can rewrite Sharpe’s ratio thus:

\[
\rho = \frac{z_{a,o}^T (m_a - \rho_f u)}{\sigma_{a,o}}
\]

(6.27)

Note that \( \rho_o \geq \rho_f \). Indeed, the portfolio \((x_f = 1, x_a = 0_n)\) is acceptable, its risk is null (0 is an absolute minimum) and its return is \( \rho_f \). The positive difference \( \rho_o - \rho_f \) is called the risk premium. From this, it follows that, in light of equation (6.26), \( z_{a,o} m_a \geq \rho_f \). Finally, if \( x_{f,o} = 0 \), \( \rho_o = z_{a,o}^T m_a \). We will now maximize Sharpe’s ratio.
Maximization of Sharpe’s ratio

The problem of maximization of Sharpe’s ratio is written as:

\[
P_1: \begin{cases} 
\max J(z_a) = \frac{z_a^T(m_a - \rho_f u)}{\sqrt{z_a^T Q z_a}} \\
z_a \in D
\end{cases}
\]  

(6.28)

where:

\[
D = \{z_a \in \mathbb{R}^n \text{ s.t. } z_a^T u = 1, \; z_a^T m_a \geq 0, \; z_a \geq 0\}
\]  

(6.29)

Note that \(J(z_a/s) = J(z_a)\) for all values of \(s \neq 0\). Thus, let us consider the domain:

\[
\tilde{D} = \{(z_a, s) \in \mathbb{R}^n \times \mathbb{R} \text{ s.t. } \frac{z_a}{s} \in D\} \cup \{0_n, 0\}
\]  

(6.30)

We have the following property:

\[
P_1: \begin{cases} 
\max J(z_a) \\
z_a \in D
\end{cases} \iff P_2: \begin{cases} 
\max J(z_a) \\
z_a \in \tilde{D}
\end{cases}
\]

Indeed, if we use the notation \(z_{a,o}\) for an optimal solution to \(P_1\) and \((z_{a,#}, s#)\) for an optimal solution to \(P_2\), we have:

- \((sz_{a,o}, s) \in \tilde{D}\), therefore \(J(z_{a,o}s/s) = J(z_{a,o}) \leq J(z_{a,#})\);
- \(z_{a,#}/s# \in D\), so \(J(z_{a,#}s#/s#) = J(z_{a,#}) \leq J(z_{a,o})\).

and hence \(J(z_{a,o}) = J(z_{a,#})\).

In the problem \(P_2\), we can now choose the parameter \(s\) in such a way that \((m_a - \rho_f u)^T z_a\) will be equal to 1. Indeed, by dividing \(z_a\) by \(s = (m_a - \rho_f u)^T z_a\), we obtain \(\tilde{z}_a = z_a/s\) which satisfies \((m_a - \rho_f u)^T \tilde{z}_a = 1\) and such that \(J(z_a) = J(\tilde{z}_a)\).

Consequently, we can replace the problem \(P_2\), and therefore problem \(P_1\) as well, with the following problem, which is easier to solve:

\[
\begin{cases} 
\min J(x) = \frac{1}{2} z_a^T Q z_a (m_a - \rho_f u)^T z_a = 1 \\
m_a^T z_a \geq 0 \\
z_a \geq 0
\end{cases}
\]  

(6.31)

These results are shown in Figure 6.9. The CAL is the tangent to the frontier of the risky investment market, passing through the point \((0, \rho_f)\). For a degree of risk less than \(\sigma_a^*\) (see Figure 6.9) the optimal portfolio lies on the...
Figure 6.9 – Optimal portfolios: the point of intersection between the Oy axis and the line has the coordinates \((0, \rho_f)\), to which \(x_f = 1\) corresponds. The point of contact \((\sigma_a, \rho_a)\) between the line and the curve also corresponds to \(x_f = 1\). Any given point on the line is associated with \(0 < x_f < 1\) CAL in accordance with a distribution given by equation (6.26). For a degree of risk greater than \(\sigma_a^*\), \(x_f = 0\) and the optimal portfolio contains solely risky investments and follows the frontier which is a solution to the problem (6.24).

The fundamental property is that the solution to the problem (6.31) depends solely on the market parameters, i.e. \(\mathbf{m}_a\), \(\rho_f\) and \(\mathbf{Q}\), and thus is independent of the expected return \(\rho_e\). In that sense, this solution constitutes a market index. Consequently, if certain financial organizations reproduce this indicator, it will be sufficient for the investor to refer to those organizations (this is known as passive management) and, on the basis of his/her taste for risk, decide upon the proportion \(x_f\) and \((1 - x_f)\) of his/her portfolio to be attached, respectively, to the risk-free investment and the risky investments.

We show, under general conditions [6], that in the presence of two uncorrelated investments funds, the optimal portfolio is a linear combination of two optimal portfolios associated, respectively, with each fund. This constitutes what is know as the two-fund separation theorem. Equation (6.26) represents a special case, where one of the two funds is risk-free.

Example 6.9 Consider a market with nine shares, whose frontier over the space of a month is illustrated in Figure 6.10 and whose distribution \(z_{a,o}\) are given in Table 6.1. These elements are market data.

Suppose that the risk-free investment has an annual return of 0.01 and the investor’s risk-acceptance is \(\sigma_o = 0.03\). Determine the optimal portfolio over a month.
Hints:

- the monthly return of the risk-free investment is:

\[ \rho_f = (1 + 0.01)^{1/12} - 1 = 8.3 \times 10^{-4} \]

We plot the tangent from the point \((0, \rho_f)\) which touches the curve at the point \((\sigma_a = 0.066, \rho_a = 0.0036)\).

- from the value \(\sigma_o = 0.03\) we deduce the optimal portfolio which is on the CAL and a return rate of \(\rho_o = 0.0021\);

- from the table we deduce the value \(\rho_a = 0.0036\) that the distribution \(z_{a,o}\) is given by the 19th row in the table;

- the non-risk/risk distribution is associated with \(x_f\) by the relation:

\[ x_f = 1 - \frac{\sigma_o}{\sigma_a} = 0.55 \]

and therefore \(x_a = 0.45z_{a,o}\). In summary, the portfolio repartition is:

\[ x_f = 0.55, \ x_{a,1} = 0.45 \times 0.0666, \ldots, x_{a,9} = 0.45 \times 0.0225 \]

which presents the return of \(21 \times 10^{-4}\) instead of the \(8.3 \times 10^{-4}\) return of the non risky asset.

As previously stated, the higher the number of assets, the more complicated it becomes to take account of inequalities. For this reason, in exercise 6.5 we use the function \texttt{fmincon} from the toolbox \texttt{optim} in MATLAB®.
Table 6.1 – Risk ($\sigma_a$) and return ($\rho_a$) of the optimal portfolio on nine assets. The other four columns represent the proportion of the portfolio invested in the four assets. The other five are given in Table 6.1

Exercise 6.5 (Optimization of a portfolio) (see p. 237)
Consider a market including one risk-free investment with a return of 0.01 and 6 investments with respective returns of 0.03, 0.04, 0.08, −0.02, 0.06 and 0.08. Its volatility matrix is:

$$Q = 0.01 \begin{bmatrix} 4 & 2 & -1 & -3 & 0 & 2 \\ 2 & 7 & 0 & 1 & 4 & -1 \\ -1 & 0 & 20 & 7 & -1 & 13 \\ -3 & 1 & 7 & 10 & -2 & 4 \\ 0 & 4 & -1 & -2 & 11 & -7 \\ 2 & -1 & 13 & 4 & -7 & 18 \end{bmatrix}$$

Using the function `fmincon` in MATLAB®, plot the market frontier.
It is widely used in a number of applications. This method has the advantage of being linear, and makes no hypothesis concerning the data distribution. 

<table>
<thead>
<tr>
<th>$\rho_a$</th>
<th>$\sigma_a$</th>
<th>$z_{a,5}$</th>
<th>$z_{a,6}$</th>
<th>$z_{a,7}$</th>
<th>$z_{a,8}$</th>
<th>$z_{a,9}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0000</td>
<td>0.0354</td>
<td>0.1556</td>
<td>0.0000</td>
<td>0.1895</td>
<td>0.0546</td>
<td>0.2174</td>
</tr>
<tr>
<td>0.0002</td>
<td>0.0354</td>
<td>0.1556</td>
<td>0.0000</td>
<td>0.1895</td>
<td>0.0546</td>
<td>0.2174</td>
</tr>
<tr>
<td>0.0004</td>
<td>0.0354</td>
<td>0.1556</td>
<td>0.0000</td>
<td>0.1895</td>
<td>0.0546</td>
<td>0.2174</td>
</tr>
<tr>
<td>0.0006</td>
<td>0.0354</td>
<td>0.1556</td>
<td>0.0000</td>
<td>0.1895</td>
<td>0.0546</td>
<td>0.2174</td>
</tr>
<tr>
<td>0.0008</td>
<td>0.0354</td>
<td>0.1556</td>
<td>0.0000</td>
<td>0.1895</td>
<td>0.0546</td>
<td>0.2174</td>
</tr>
<tr>
<td>0.0010</td>
<td>0.0354</td>
<td>0.1556</td>
<td>0.0000</td>
<td>0.1895</td>
<td>0.0546</td>
<td>0.2174</td>
</tr>
<tr>
<td>0.0012</td>
<td>0.0354</td>
<td>0.1556</td>
<td>0.0000</td>
<td>0.1895</td>
<td>0.0546</td>
<td>0.2174</td>
</tr>
<tr>
<td>0.0014</td>
<td>0.0354</td>
<td>0.1556</td>
<td>0.0000</td>
<td>0.1895</td>
<td>0.0546</td>
<td>0.2174</td>
</tr>
<tr>
<td>0.0016</td>
<td>0.0354</td>
<td>0.1556</td>
<td>0.0000</td>
<td>0.1895</td>
<td>0.0546</td>
<td>0.2174</td>
</tr>
<tr>
<td>0.0018</td>
<td>0.0359</td>
<td>0.1271</td>
<td>0.0000</td>
<td>0.1685</td>
<td>0.0525</td>
<td>0.2038</td>
</tr>
<tr>
<td>0.0020</td>
<td>0.0376</td>
<td>0.0956</td>
<td>0.0000</td>
<td>0.1607</td>
<td>0.0530</td>
<td>0.1950</td>
</tr>
<tr>
<td>0.0022</td>
<td>0.0398</td>
<td>0.0641</td>
<td>0.0000</td>
<td>0.1530</td>
<td>0.0536</td>
<td>0.1862</td>
</tr>
<tr>
<td>0.0024</td>
<td>0.0425</td>
<td>0.0327</td>
<td>0.0000</td>
<td>0.1453</td>
<td>0.0541</td>
<td>0.1774</td>
</tr>
<tr>
<td>0.0026</td>
<td>0.0456</td>
<td>0.0012</td>
<td>0.0000</td>
<td>0.1375</td>
<td>0.0547</td>
<td>0.1686</td>
</tr>
<tr>
<td>0.0028</td>
<td>0.0490</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.1212</td>
<td>0.0514</td>
<td>0.1400</td>
</tr>
<tr>
<td>0.0030</td>
<td>0.0530</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.1046</td>
<td>0.0479</td>
<td>0.1106</td>
</tr>
<tr>
<td>0.0032</td>
<td>0.0573</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0880</td>
<td>0.0444</td>
<td>0.0813</td>
</tr>
<tr>
<td>0.0034</td>
<td>0.0618</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0713</td>
<td>0.0410</td>
<td>0.0519</td>
</tr>
<tr>
<td>0.0036</td>
<td>0.0666</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0547</td>
<td>0.0375</td>
<td>0.0225</td>
</tr>
<tr>
<td>0.0038</td>
<td>0.0716</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0374</td>
<td>0.0336</td>
<td>0.0000</td>
</tr>
<tr>
<td>0.0040</td>
<td>0.0768</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0179</td>
<td>0.0282</td>
<td>0.0000</td>
</tr>
<tr>
<td>0.0042</td>
<td>0.0821</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0227</td>
<td>0.0000</td>
</tr>
<tr>
<td>0.0044</td>
<td>0.0876</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0169</td>
<td>0.0000</td>
</tr>
<tr>
<td>0.0046</td>
<td>0.0932</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0110</td>
<td>0.0000</td>
</tr>
<tr>
<td>0.0048</td>
<td>0.0990</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>0.0050</td>
<td>0.1077</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

Table 6.1 – (continued)

### 6.4 Principal Component Analysis (PCA)

The principal component analysis (or PCA) provides a simple way of reducing a complex set of data by projecting it onto a space with a small dimension while preserving as much of the variability as possible. This method has the advantage of being linear, and makes no hypothesis concerning the data distribution. It is widely used in a number of applications.
6.4.1 Determining the principal components

Consider $N$ length $d$ vectors $\mathbf{x}_1, \ldots, \mathbf{x}_N$ each one of them associated with a set of $d$ measurements, and $\mathbf{X}$ the matrix:

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & \ldots & x_{1,N} \\ \vdots & \ddots & \vdots \\ x_{d,1} & x_{d,2} & \ldots & x_{d,N} \end{bmatrix}$$

$\mathbf{X}$ can be understood from two points of view:

- either as a set of $N$ columns $\mathbf{x}_n$, each one of them representing $d$ factors associated with a same individual. This leads to $N$ points in the space $\mathbb{R}^d$;

- or as $d$ lines, each one of them representing the same factor for $N$ individuals. This leads to $d$ points in the space $\mathbb{R}^N$.

We wish to reduce the number of factors to keep only the $k < d$ most significant ones.

First consider the case where $k = 1$. We have to search in $\mathbb{R}^d$ the direction of the unit vector $\mathbf{v}$ such that the projection of the set of the $\mathbf{x}_n$ onto this direction leads to the scatter of $N$ points with the highest dispersion (Figure (6.11) for $d = 2$).

![Figure 6.11 – Projecting the samples for the directions $\mathbf{v}_1$ and $\mathbf{v}_2$: the dispersion of the projected points is more favorable to an analysis for the vector $\mathbf{v}_1$ than it is for $\mathbf{v}_2$](image)

This can be interpreted as follows: if we can only keep one component, it might as well be the one that best separates all of the points. A classic criterion to evaluate this dispersion is to consider the sum of the distances between all of the projected points. Remember that the projection of $\mathbf{x}_n$ is given by $\mathbf{vv}^T \mathbf{x}_n$.
and that the distance between any two points is written $\|vv^T x_i - vv^T x_j\|^2 = v^T (x_i - x_j)(x_i - x_j)^T v$. The criterion to be minimized is then written:

$$J(v) = \frac{1}{2N^2} v^T \left( \sum_{i=1}^N \sum_{j=1}^N (x_i - x_j)(x_i - x_j)^T \right) v$$

Let $\bar{x} = N^{-1} \sum_{j=1}^N x_j$. Therefore:

$$J(v) = \frac{1}{2N^2} v^T \left( \sum_{i=1}^N \sum_{j=1}^N [(x_i - \bar{x}) - (x_j - \bar{x})][(x_i - \bar{x}) - (x_j - \bar{x})]^T \right) v$$

$$= v^T R_N v$$

where we have:

$$R_N = \frac{1}{N} \sum_{n=1}^N (x_n - \bar{x})(x_n - \bar{x})^T \quad (6.32)$$

Notice that the $(d \times d)$ matrix $R_N$ can be interpreted as a covariance matrix. Hence the problem can be laid out as follows:

$$\begin{cases}
\max_v J(v) \\
\text{with} \quad v^T v - 1 = 0
\end{cases}$$

The Lagrange multipliers lead us to the following equivalent problem:

$$\begin{cases}
\max_v (v^T R_N v - \lambda(v^T v - 1)) \\
v^T v - 1 = 0
\end{cases}$$

which leads, by setting to zero the gradient with respect to $v$:

$$\begin{cases}
R_N v - \lambda v = 0 \\
v^T v - 1 = 0
\end{cases}$$

The first equation means that $v$ is an eigenvector of the matrix $R_N$. If the eigenvalue associated with $v$ is denoted by $\lambda$, then $J(v) = \lambda \geq 0$. Hence the maximum is reached when $v$ is chosen as the eigenvector of $R_N$ associated with the highest eigenvalue: hence the name of this method, Principal Component Analysis (or PCA).

By calculating the product $v^T X$, we get a line vector of $\mathbb{R}^N$. This line vector can be interpreted as the “mean” factor that best characterizes, by itself, the $N$ individuals (it results in a good separation). However, this “factor” does
not really exist, it is merely a linear combination of factors that are actually observed.

The previous result can easily be generalized: the \( k \) principal directions are the \( k \) eigendirections of the highest eigenvalues.

Implementing the previous calculations can pose a problem in the case of image processing. Let us assume for example that we are dealing with images comprised of \( 100 \times 100 \) pixels represented in the form of length \( 10^4 \) vectors. The matrix \( R_N \) associated with these vectors is a \((10^4 \times 10^4)\) matrix! The following property can therefore be useful when \( d \gg N \).

**Property 6.1** Let \( A \) be a \((d \times N)\) matrix. Let \( \{\lambda_1, \ldots, \lambda_N\} \) be the \( N \) eigenvalues of the \((N \times N)\) matrix \( A^H A \). Then the \( d \) eigenvalues of the \((d \times d)\) matrix \( AA^H \) are:

\[
\{\lambda_1, \ldots, \lambda_N, 0, \ldots, 0\}, \quad d-N
\]

HINTS: Let \( \lambda \) be an eigenvalue of \( A^H A \) associated with the eigenvector \( v \), which is written \( A^H A v = \lambda v \). If we multiply both sides of this equality on the left by \( A \), we get:

\[
AA^H A v = \lambda A v
\]

Letting \( w = A v \), we get \( AA^H w = \lambda w \). Therefore \( \lambda \) is the eigenvalue of \( AA^H \) associated with the eigenvector \( w = Av \).

**Example 6.10 (Eigenfaces)** Construct a database containing front views of the faces of different people\(^{(2)}\). We can use the ORL database, available to anyone on AT&T’s web site. This database contains photographs showing the faces of 40 people. Each one of them was photographed 10 times. These photos are stored as images in levels of grey with \( 112 \times 92 \) pixels. In our example, we constructed a catalog called orlfaces, comprised of the catalogs named \( s_1, s_2, \ldots, s_{40} \), each one of them containing the 10 photographs we are going to process.

Write a MATLAB\textsuperscript{®} program:

- that changes each \((d_1 = 112) \times (d_2 = 92)\) photograph into a vector;

- that constructs, using a photograph of each of the \( N \) people, a subspace \( \mathcal{H} \) the dimension of which is less than or equal to \( N \), and such as to have the maximum dispersion of the \( N \) projections (think of using the property (6.1));

\(^{(2)}\)When using PCA for face recognition, it is important for the photos corresponding to the same person to be taken in approximately the same position and the same lighting. Otherwise, an alignment and a calibration are often unavoidable to achieve satisfactory results.
that checks the identity corresponding to a photograph by determining its projection onto $H$ then by comparing the distances of this projection with respect to the $N$ projections obtained with the $N$ previous photographs;

which constructs the confusion, for which the element $(i,j)$ represents the number of times the person $i$ was chosen as being the person $j$.

**Hints:** Type:

```matlab
%===== exeigenfaces.m
clear all
d1=112; d2=92; d=d1*d2; figure(1); colormap('gray')
imagesNb=10; peopleNb=10; images=cell(peopleNb,imagesNb);
matX=zeros(d,peopleNb,imagesNb);
eigenfaces1=zeros(d,peopleNb);
for ni=1:peopleNb
    for kimg=1:imagesNb
        filename=sprintf('orlfaces/s%i/%i.png',ni,kimg);
        images{ni,kimg}=imread(filename);
        aux=images{ni,kimg};
        matX(:,ni,kimg)=reshape(aux,d,1);
    end
end
%===== Training of the eigenfaces using the first image
% of each person
moymatX=matX(:,:,1)*ones(peopleNb,1)/peopleNb;
nbeig=6; eigenfaces=zeros(d,nbeig); matXc=zeros(d,peopleNb);
for ni=1:peopleNb, matXc(:,ni)=matX(:,ni,1)-moymatX; end
RR=matXc'*matXc; [UU, DD, VV]=svd(RR); DD=diag(DD);
UUpbT=matXc(:,:,1)*VV(:,1:nbeig);
UUpbT=UUpbT*diag(1./sqrt(DD(1:nbeig)));
for ni=1:peopleNb,
    cni(ni,:)=matXc(:,ni,1)'*UUpbT;
    vi=reshape(UUpbT*cni(ni,:),d1,d2);
    subplot(211);imagesc(reshape(matX(:,ni),d1,d2));
    axis('image'); subplot(212);
    imagesc(vi); axis('image'); pause
end
%===== Testing the pictures of the person
matriceconf=zeros(peopleNb,peopleNb);
for testedNb=1:peopleNb
    for ii=2:imagesNb
        aux1=(matX(:,testedNb,ii)-moymatX)'*UUpbT;
        for ni=1:peopleNb
            aux2(ni)=norm(aux1-cni(ni,:));
        end
        [aa zz]=min(aux2);
        matriceconf(zz,testedNb)=matriceconf(zz,testedNb)+1;
    end
end
```

```
6.4.2 2-Dimension PCA

Consider $N$ images of the same size $d_1 \times d_2$. Let $A_\ell$ be the matrix representing the image $\ell$ and

$$A_{c,\ell} = A_\ell - N^{-1} \sum_{j=1}^{N} A_j$$

be the centered image. We need to determine $k_1$ length $d_1$ unit vectors $v$ such that the vectors $y_\ell = v^T A_{c,\ell}$ are maximally dispersed. This is done by maximizing the quantity:

$$\sum_{\ell=1}^{N} v^T A_{c,\ell} A_{c,\ell}^T v$$

A calculation similar to the one done in the previous paragraph, for length 1 vectors, proves that the vectors we are trying to determine are the $k_1$ eigenvectors of the $(d_1 \times d_1)$ square matrix $R_1$:

$$R_1 = N^{-1} \sum_{\ell=1}^{N} A_{c,\ell} A_{c,\ell}^T$$

Let $V$ be the $(d_1 \times k_1)$ matrix obtained by compiling these $k_1$ vectors according to the expression:

$$V = \begin{bmatrix} v_1 & \ldots & v_k \end{bmatrix}$$

(6.33)

Likewise, if we wish to determine the $k_2$ length $d_2$ unit vectors such that the vectors $y_\ell = A_{c,\ell} w$ are maximally dispersed, we end up finding that these vectors are the $k_2$ eigenvectors of the $(d_2 \times d_2)$ square matrix $R_2$:

$$R_2 = N^{-1} \sum_{\ell=1}^{N} A_{c,\ell}^T A_{c,\ell}$$

Let $W$ be the $(d_2 \times k_2)$ matrix obtained by compiling these $k_2$ vectors according to the expression:

$$W = \begin{bmatrix} w_1 & \ldots & w_k \end{bmatrix}$$

(6.34)

In practice, $k_1$ and $k_2$ are chosen such that $k_1 < d_1$ and $k_2 < d_2$, leading to a sequence of $N$ “reduced” images of the size $k_1 \times k_2$, according to the expression:

$$B_\ell = V^T A_\ell W$$

(6.35)
Example 6.11 (2D-PCA) Write a program that extracts from a set of $N$ $(d_1 \times d_2)$ images the matrices $V$ and $W$ that allow us, according to expression (6.35), to obtain $N$ $(k_1 \times k_2)$ images. Have the program display the $N$ images in the form of an array of cells, each cell representing a $(d_1 \times d_2)$ image, and return the matrices $V$ and $W$.

Hints: type the following function:

```matlab
function [V,W] = PCA2D(matXcell,k1,k2)
%!=========================================!
%! SYNOPSIS: [V,W] = PCA2D(matXcell,k1,k2) !
%! matXcell : dimension N cell array !
%! a cell is a dimension (d1 x d2) array !
%! k1 = reduced number of rows !
%! k2 = reduced number of columns !
%! Output: !
%! V = dimension (k1 x d1) array !
%! W = dimension (k2 x d2) array !
%!=========================================!
N=length(matXcell); gXcellc=cell(1,N);
[d1,d2]=size(matXcell{1});
moy_image=zeros(d1,d2); GG=zeros(d1,d2);
for ii=1:N,
    moy_image=moy_image+double(matXcell{ii});
end
moy_image=moy_image/N;
%===== centered images
for ii=1:N,
gXcellc{ii}=double(matXcell{ii})-moy_image;
end
gR=zeros(d1,d1);
for ii=1:N, gR=gR+gXcellc{ii}*gXcellc{ii}'; end
%===== a covariance
gR=gR/N; [gU, lambda, gV]=svd(gR);
ap=lambda(1:k1,1:k1); V=gU(:,1:k1);
%===== the other covariance
gR=zeros(d2,d2);
for ii=1:N, gR=gR+gXcellc{ii}'*gXcellc{ii}; end
gR=gR/N; [gU, lambda, gV]=svd(gR);
W=gU(:,1:k2);
```

This function is used in exercise 6.6.

When trying to recognize someone among $g$ individuals, the PCA approach makes you determine separately, during the training phase, the principal directions for each group of individuals. The approach we are now going to see makes it possible to simultaneously optimize the choice of the principal directions and the separation into groups.
6.4.3 Linear Discriminant Analysis

We now consider \( g \) groups of individuals, each group comprised of \( N_\ell \) individuals for which \( d \) factors were measured. The data can then be represented in the form of \( g \) matrices of the type:

\[
X_\ell = \begin{bmatrix}
x_{\ell,1} & \cdots & x_{\ell,N_\ell}
\end{bmatrix}
\]

with \( \ell \) from 1 to \( g \).

In the space \( \mathbb{R}^d \) we obtain \( g \) scatters containing respectively \( N_1, \ldots, N_g \) points. We let \( N = \sum_{\ell=1}^{g} N_\ell \).

The goal of the Linear Discriminant Analysis (or LDA) is to find the best separation for these \( g \) scatters of points. To achieve this, we must first introduce the following definitions:

- the mean, or barycenter, of a group:

\[
m_\ell = \frac{1}{N_\ell} \sum_{j=1}^{N_\ell} x_{\ell,j}
\]

- the overall mean of the \( g \) groups:

\[
m = \frac{1}{N} \sum_{\ell=1}^{g} \sum_{j=1}^{N_\ell} x_{\ell,j} = \frac{\sum_{\ell=1}^{g} N_\ell m_\ell}{\sum_{\ell=1}^{g} N_\ell} \tag{6.36}
\]

- the intraclass covariance (internal to the considered class) defined by:

\[
R_I = \frac{\sum_{\ell=1}^{g} N_\ell R_\ell}{\sum_{\ell=1}^{g} N_\ell} \quad \text{with} \quad R_\ell = \frac{1}{N_\ell} \sum_{j=1}^{N_\ell} (x_{\ell,j} - m_\ell)(x_{\ell,j} - m_\ell)^T
\]

which leads us to:

\[
R_I = \frac{1}{N} \sum_{\ell=1}^{g} \sum_{j=1}^{N_\ell} x_{\ell,j} x_{\ell,j}^T - \frac{1}{N} \sum_{\ell=1}^{g} N_\ell m_\ell m_\ell^T
\]

which can be interpreted as the mean of the dispersions inside of each group;
- the extraclass covariance defined by:

\[ R_E = \frac{1}{N} \sum_{\ell=1}^{g} N_\ell (m_\ell - m)(m_\ell - m)^T \]

which can be interpreted as the dispersion of the barycenters of each class with respect to the overall mean;

- the total covariance defined by:

\[ R = \frac{1}{N} \sum_{\ell=1}^{g} \sum_{j=1}^{N_\ell} (x_{\ell,j} - m)(x_{\ell,j} - m)^T = \frac{1}{N} \sum_{\ell=1}^{g} \sum_{j=1}^{N_\ell} x_{\ell,j}x_{\ell,j}^T - mm^T \]

These covariance matrices are \((d \times d)\) matrices. It can easily be proved that:

\[ R = R_I + R_E \]

**Property 6.2** \(R_E\) is of rank \(r = \min(d, g - 1)\).

**Hints:** Indeed

\[ NR_E = \sum_{\ell=1}^{g} N_\ell (m_\ell - m)(m_\ell - m)^T \]

where \(m\) is given by expression (6.36). We let

\[ M = [\sqrt{N_1}(m_1 - m) \ldots \sqrt{N_g}(m_g - m)] \]

The \(d \times g\) matrix \(M\) verifies \(MM^T = \sum_{\ell=1}^{g} N_\ell (m_\ell - m)(m_\ell - m)^T = NR_E\). On the other hand

\[ M \begin{bmatrix} \sqrt{N_1} \\ \vdots \\ \sqrt{N_g} \end{bmatrix} = \sum_{j=1}^{g} N_j m_j - mN = 0 \]

Therefore the rank of \(M\) is at most equal to \(\min(d, g - 1)\). That means that the space spanned by the columns of \(M\) is at most of dimension \(\min(d, g - 1)\). That is also true for \(NR_E = MM^T\).
From property 6.2, it follows that, if \( g - 1 < d \), \( R_E \) is not invertible.

We are now going to find the direction, parallel to the vector \( v \), such that the intraclass dispersion is minimal and the interclass dispersion is maximal: graphically speaking, the scatters are farther away from each other, and more compact. To achieve this objective, one possible criterion is to minimize the evaluation function defined by:

\[
J(v) = \frac{v^T R_I v}{v^T R_E v}
\]

This amounts to searching \( v \) such that:

\[
\min_v v^T R_I v \quad \text{with} \quad 1 - v^T R_E v = 0
\]

Using the Lagrange multiplier method (section (6.3)), we end up with the following equivalent problem:

\[
\begin{align*}
\min_v & \quad (v^T R_I v + \lambda (1 - v^T R_E v)) \\
\text{with} & \quad 1 - v^T R_E v = 0
\end{align*}
\]

By setting the gradient with respect to \( v \) to zero, we find that \( v \) is given by:

\[
R_I v = \lambda R_E v
\]

Notice that if \( v \) is a solution of equation (6.38), then \( J(v) = \lambda \) with \( \lambda \geq 0 \). Now we have to solve the equation (6.38) with respect to \( v \) and choose \( v \) associated to the minimal value of \( \lambda \). The resolution of (6.38) is a well known problem which is called the generalized eigenvalue problem. In our case \( R_E \) and \( R_I \) are positive, therefore it exists a basis of generalized eigenvectors which are solution of (6.38).

**Remark**: if \( R_I \) is invertible, then the equation (6.38) can be written in the form:

\[
R_I^{-1} R_E v = \frac{1}{\lambda} v
\]

which is a standard eigenvalue problem. However, in most situations it is preferable to avoid the inversion operation. Fortunately, the generalized eigenvalue problem is solved in MATLAB® using the function `eig.m`.

To increase the capability of classification, we can choose more than one eigenvector verifying equation (6.38). Let \( v_1, \ldots, v_k \) be the \( k \) eigenvectors associated with the \( k \) lowest generalized eigenvalues of equation (6.38). By compiling these \( k \) vectors, we get the \((d \times k)\) matrix:

\[
V = [v_1 \ldots v_k]
\]
It is worth noting that, because $R_t^{-1}R_E$ is not in general positive, the eigenvectors $v_1, \ldots, v_k$ are not orthogonal. This means that for each of the $g$ families, the vectors:

$$y_{\ell,j} = (V^T V)^{-1} V^T x_{\ell,j} \quad (6.41)$$

give the representative points in $\mathbb{R}^k$ of each image. Theoretically, each of the $g$ scatter has a minimal dispersion, and all of the scatters are as far away from each other as possible.

**Example 6.12 (LDA)** Write a function that performs the linear discriminant analysis of $g$ groups, each group containing $N_\ell$ vectors of the same length $d$. We will be using $g$ cells as the data format with MATLAB®, where each cell is a $(d \times N_\ell)$ matrix. The program returns the matrix $V$ defined by equation (6.40) which is used for changing the $g$ scatters in $\mathbb{R}^d$ into $g$ “well” separated scatters in $\mathbb{R}^k$.

**Hints:** Type the following function:

```matlab
function VlowestEIG = LDA(gX,kk)

% SYNOPSIS: VlowestEIG=LDA(gX,kk)
% gX = array of cells. Each cell is a (d x Nell) matrix associated to the Nell vectors of a class
% kk = reduced dimension (kk<d)
% VlowestEIG = (kk x d) matrix used to reduce X with Xreduced=V'X

d = size(gX{1},1); G = length(gX); moyell = cell(1,G); Nell = cell(1,G); gXc = cell(1,G); moyT = zeros(d,1); NT = 0; RI = zeros(d,d);
for ell=1:G
  gXell = gX{ell}; Nell{ell} = size(gXell,2); NT = NT+Nell{ell}; moyell{ell} = mean(gXell,2); moyT = moyT+Nell{ell}*moyell{ell}; moyT = moyT/NT; moyT = moyT*Nell{ell}; gXellc = gXell-double(moyell{ell}); gXc{ell} = gXc{ell}; RI = RI+gXellc*gXellc';
end
RI = RI/NT; moyT = moyT/NT; RE = zeros(d,d);
```
Exercise 6.6 (Face recognition) (see p. 238) Consider a database comprised of \( g \) groups of photographs in levels of grey of the same person’s face, taken from the front. We can once again use the ORL database, available on AT&T’s website. In this exercise, the database of ten photos is divided in two: three photographs are used for training, and the seven others are used for recognition tests.

**Training:** Write a program:

- that determines, using the function in example 6.11, the matrices \( V \) and \( W \) associated with each person (try for instance \( k_1 = 7 \) and \( k_2 = 7 \));
- that determines the \( k = k_1 k_2 \) vector characterizing each person and each photograph;
- that learns from the \( g \) previous groups of vectors the matrices of the linear discriminant analysis. Use the function from example (6.12). By choosing the dimension \( d = 2 \), you can display representative barycenters in the plane, but in order to obtain good recognition results, you will need a higher value for \( d \), for example \( d = 20 \), requiring that the maximal value of \( d \) is \( k \). When \( d \) becomes too large, for example \( d = 40 \) with \( k_1 = k_2 = 7 \), we observe that performance worsens. The reason is that, when you increase the number of parameters, the learning step memorizes too much unnecessary detail on the training examples. This phenomena is called overfitting;
- that finds the barycenters characterizing each individual.

**Recognition:** Write a program that successively determines from expressions (6.35) and (6.41) the position of the test images in the space \( \mathbb{R}^d \) \((d = 2)\). Note that, usually, the person’s identity is not known. Therefore the obtained position has to be compared with the scatters characterizing each person. As a test function, you can use, for example, the distance between obtained position and the barycenter of the scatter, and construct the confusion matrix.
The programs developed in exercise 6.6 can also be used for character recognition. The program `geneBDDgene.m` allows you to generate a database of the 10 digits in printing characters, each digit being recorded in 12 different copies. After having created the folders `s0, ..., s9` in the folder `mdigits`, type:

```matlab
%===== digitbdddgenem.m
clear all; close all
name={'times','courrier','verdana'}; sizech=[120 100 60];
angle={'normal','italic'}; bolfCh={'normal','bold'};
for ii=0:9
    for jj=1:3
        figure(1); tt=sprintf('%i',ii);
        set(gcf,'color', [1 1 1], 'position', [60 500 180 180])
        plot(0,0); set(gca,'unit','pixel')
        set(gca,'xtick', []), set(gca,'ytick', [])
        set(gca,'box','off','xcolor', [1 1 1], 'ycolor', [1 1 1])
        ff=text(-1,0,tt);
        set(ff,'fontsize',sizech(jj),'fontname',name{jj})
        for kk=1:2
            set(ff,'fontangle',angle{kk})
            for mm=1:2
                num=2*2*(jj-1)+2*(kk-1)+mm;
                cde=sprintf(...
                'print -dtiffnocompression mdigits/s%i/%i.tif',...
                ii,num);
                cde=sprintf(...
                'print -dpng mdigits/s%i/%i.png',ii,num);
                % Windows only:
                cde=sprintf(...
                'print -dbitmap mdigits/s%i/%i.bmp',ii,num);
                eval(cde)
        end
    end
end
```

Use the program `LDAPCAtest.m` while adapting the values of the parameters. Try it in particular with `nbimages_A=4; dim_barycenter=30;` and `k1=6; k2=6;` (set in `LDAPCAtraining.m`). The `-dbitmap` parameter is only supported by the MS-Windows operating system. The called functions will have to be modified accordingly. It is usually preferable to use the options `dtiffnocompression` or `dpng`.

### 6.5 GPS positioning

GPS is the US-designed satellite positioning system (Global Navigation Satellite System, GNSS). It comprises a network of 30 satellites in quasi-circular
orbit at an altitude of around 20,000 km, which takes a little less than half an Earth day to complete. In practice, it is hoped that at least 4 of these satellites will be reachable at any given time from any point on the Earth’s surface.

In an arbitrary frame of reference, \(x\) is the receiver’s true position, \(\xi_i\) the position of the satellite \(i\) and \((a_i, e_i)\) the azimuth/elevation coordinates which characterize the satellite’s position \(i\) in relation to the receiver.

Assume that the receiver can see \(I\) satellites and that the position of those satellites is perfectly known at all times (Figure 6.12).

The term “pseudo-distance” is used for the time taken, in seconds, for the signal from the satellite to reach the receiver. By multiplying this figure by the celerity of light \(c\), it is possible to transform these time values into distances. Thus, the pseudo-distance between the receiver and the satellite \(i\) is written as:

\[
p_i = c^{-1}d(x, \xi_i) + \tau + e_i
\]

where \(d(x, \xi_i) = \sqrt{(x_1 - \xi_{1,i})^2 + (x_2 - \xi_{2,i})^2 + (x_3 - \xi_{3,i})^2}\) is the distance between the receiver and the satellite \(i\); \(\tau\) an unknown time difference between the receiver’s clock and those of the satellites, which are supposed to be synchronized; and \(e_i\) an error due to noise which is supposed to be centered.

The pseudo-distances are measured by observing the time taken for signals to travel from the satellites to the receiver. Thus, if two signals are received at exactly the same moment, this means that the receiver is at an equal distance from the two satellites.

The estimations of the position \(x\) and the delay \(\tau\) on the basis of the measurements \(p_i\) are obtained by a least-squares method, i.e. as a solution to the problem:

\[
[\hat{x}_1 \ \hat{x}_2 \ \hat{x}_3 \ \hat{\tau}]^T = \arg \min_{x, \tau} \sum_{i=1}^{I} (p_i - c^{-1}d(x, \xi_i) - \tau)^2
\] (6.42)
By adopting matrix notation, we can also write:

\[
\begin{bmatrix}
\hat{x}_1 & \hat{x}_2 & \hat{x}_3 & \hat{\tau}
\end{bmatrix}^T = \arg \min_{x_1, x_2, x_3, t} \| p - f(x_1, x_2, x_3, t) \|^2
\]

where \( p = [p_1 \ldots p_I]^T \) is the observations vector and \( f : [x_1 \ x_2 \ x_3 \ t]^T \mapsto [f_1 \ldots f_I]^T \) is a function of \( \mathbb{R}^4 \) in \( \mathbb{R}^I \), whose \( i \)-th component has the expression:

\[
f_i(x_1, x_2, x_3) = c^{-1}\sqrt{(x_1 - \xi_{i,1})^2 + (x_2 - \xi_{i,2})^2 + (x_3 - \xi_{i,3})^2 + t}
\]

where \( d_i = f_i(x_1, x_2, x_3, 0) \) represents the distance between the satellite \( i \) and the receiver. The solution to the problem (6.42) satisfies:

\[
- \nabla f^T(\hat{x}_1, \hat{x}_2, \hat{x}_3, \hat{\tau}) (p - f(\hat{x}_1, \hat{x}_2, \hat{x}_3, \hat{\tau})) = 0
\]  
(6.43)

where:

\[
\nabla f(x_1, x_2, x_3, t) = \begin{bmatrix}
\frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\
\vdots & \vdots & \vdots \\
\frac{\partial f_I}{\partial x_1} & \frac{\partial f_I}{\partial x_2} & \frac{\partial f_I}{\partial x_3}
\end{bmatrix}
\]

Note that \( \nabla f(x_1, x_2, x_3, t) \) depends neither on \( t \) nor on the distances between the receiver and the satellites, but only from the angles from which the receiver sees the satellites. Unfortunately, the system of equations (6.43) to solve is highly nonlinear. In practice, we solve equation (6.43) with a Newton–Raphson-type iterative algorithm (see expression (4.26)). Remember that the Newton–Raphson method is used to seek the zeros in equation \( g(\theta) = 0 \) where \( g : \theta \in \mathbb{R}^p \mapsto \mathbb{R}^q \). It is written as:

\[
\theta^{(p+1)} = \theta^{(p)} - (\nabla_\theta g(\theta^{(p)}))^{-1} g(\theta^{(p)})
\]

where:

\[
\nabla_\theta g(\theta^{(p)}) = \begin{bmatrix}
\frac{\partial g_1}{\partial \theta_1} & \frac{\partial g_1}{\partial \theta_2} & \ldots & \frac{\partial g_1}{\partial \theta_p} \\
\vdots & \vdots & \ldots & \vdots \\
\frac{\partial g_q}{\partial \theta_1} & \frac{\partial g_q}{\partial \theta_2} & \ldots & \frac{\partial g_q}{\partial \theta_p}
\end{bmatrix}
\]

It is associated with problems of minimization when the function to be canceled is the gradient of a function \( J(\theta) \), \( \theta \in \mathbb{R}^p \mapsto \mathbb{R} \).
Exercise 6.7 (GPS location) (see p. 240)
Write a program which uses the Newton–Raphson method to determine the receiver’s position based on the data provided by the following program:

```matlab
%===== data4exeGPS.m
%===== light velocity in m/s
c_mps = 299792458;
%===== true location in meter
% for comparison with the estimated location
true_location_m = [248645.5722;-4828261.0758;4146460.5047];
%===== location in meter of the 6 satelites
sat_location_m = 1.0e+07 * ...;
% pseudo-ranges in second
pseudorange_s = [ ...;
    0.075522044080650; ...;
    0.074908679917403; ...;
    0.079304017358102; ...;
    0.076286737020851; ...;
    0.080875081503129; ...;
    0.069734492372016];
```

6.6 The Viterbi algorithm

Digital Signal Processor (DSP) architectures were initially designed for filtering. Afterwards, abilities for calculations involved in digital signal processing were added: power, spectrum, FFT, etc. Subsequently, with the development of mobile phones, the processing of frames received by such devices was also added. Below, we use an example to explain the processing of “hard decision” in the Viterbi algorithm and its implementation in common digital signal processors. MATLAB® programs are given to illustrate the coding and decoding processes.

A simplified communication channel is illustrated by Figure 6.13. Coding of the binary symbols $b_k$ with values in the range $\{0, 1\}$ provides binary vectors denoted by $\tilde{s}_k$ with values in the range $\{0, 1\}$ (see Figure 6.14). The baseband vectors thus emitted are denoted by $\tilde{s}_k$, with each component being 1 or −1 given by the equation $\tilde{s}_k = 2s_k - 1$. 

The noise is denoted by \( \{n_k\} \) and the vectors \( \tilde{x}_k \) are coded over 1 bit (hard decision) or \( n > 1 \) bits (soft decision).

### 6.6.1 Convolutional non-recursive encoder

Among the numerous methods which exist for the coding of data frames, convolutional codes are widely used [11]. They exhibit performances which deteriorate progressively with the error rate, and decoding can be done over data blocks of arbitrary length. Most recent DSPs provide processing units which simplify the implementation of the Viterbi algorithm.

Let us consider as an example the convolutional non-recursive encoder [20] shown in Figure 6.14. This encoder is described as a “1/3 rate convolutional non-recursive encoder”. For each binary symbol \( b_n \), three bits – \( s_0(n) \), \( s_1(n) \) and \( s_2(n) \) – are generated, and the output stream is \( \{... s_2(n-1), s_1(n-1), s_0(n-1), s_2(n), s_1(n), s_0(n), s_2(n+1), s_1(n+1), s_0(n+1) ...\} \).

The symbols \( \oplus \) denote modulo-2 sums. They will be replaced by “+” when there is no ambiguity.

The parameters used to describe such an encoder are:

1. The Constraint Length \( K \), which is also the sequential automaton state space dimension +1, here \( K = 3 \).

2. The Code Rate \( R \), which is the ratio between the number \( n \) of information bits to be coded and the code length \( k \), here \( R = n/k = 1/3 \).
3. The *generator polynomials* which define the code – here $G_0 = 100$, $G_1 = 101$ and $G_2 = 111$ where $G_k = \{a_0, a_1, a_2\}$ corresponds to the sum $a_0b_n \oplus a_1b_{n-1} \oplus a_2b_{n-2}$.

**Example:** in some of the DSPs of the TMS320C6000™ family, the TMS320C64x DSP Viterbi-Decoder Coprocessor unit (VCP) [20] available parameters are $K = 5$ to 9 and $R = 1/2$, 1/3 or 1/4.

**Example:** in the GSM (Global System for Mobile communications), voice coding uses the parameters: $R = 1/2$, $K = 5$, a frame length of $N = 189$ bits and 50 frames per second. Parameters for data transmission are slightly different and *puncturing* is used, facilitating a code rate equal to $R = 57/61$ (puncturing deletes bits of the emitted frame and decoding necessitates a soft decision algorithm).

The state transition graph corresponding to Figure 6.14 can be represented by Figure 6.15.

**Figure 6.15** – *The state transition graph (transition: $b(n)/(s_2s_1s_0)(n)$) of the encoder shown in Figure 6.14*

The encoding can also be represented by Figure 6.16 corresponding to a step in the encoding process.

**Figure 6.16** – *A step in the encoding process*
The coding of states by \("b_nb_{n-1}\)" is chosen in order to make the programming easier. With such a code, the decoding is straightforward when the sequence of states is known. Hereafter, we shall denote the emitted vector corresponding to the binary symbol \(b_n\) as \(\underline{s}(n) = [s_2(n) \ s_1(n) \ s_0(n)]\).

**Coding**

Let \(\underline{S}_n\) be the state at time \(n\). \(x_n\) and \(y_n\) define the two-bit code for \(\underline{S}_n\):

\[
\underline{S}_n = [x_n \ y_n]^T
\]  
(6.44)

One has a state representation (the modulo-2 sum can be replaced by a sum):

\[
\underline{S}_{n+1} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \underline{S}_n \oplus \begin{bmatrix} 0 \\ 1 \end{bmatrix} b_n = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \underline{S}_n + \begin{bmatrix} 0 \\ 1 \end{bmatrix} b_n
\]  
(6.45)

In the program `coderexple.m` the coded sequence can be generated more efficiently (avoiding a matrix-vector multiplication) as follows:

```matlab
%===== coderexple =====
clear all, N=1000; n=(0:N-1);
prcodes=[[0;0;0],[1;1;0],[1;0;0],[0;1;0],...
         [1;1;1],[0;0;1],[0;1;1],[1;0;1]];
%===== Mersenne twister
s=RandStream.create('mt19937ar','seed',134);
RandStream.setDefaultStream(s);
bn=randi([0 1],1,N);
Rn=zeros(3,N); % coded sequence
Sn=zeros(1,N); % state
for k=1:N-1
    kn=4*bn(k)+Sn(k);
    Rn(:,k)=prcodes(:,kn+1);
    Sn(k+1)=floor(kn/2); % transition
end
plot(n,Sn,'o',n,Sn,'-'); grid
Rnt=reshape(Rn,1,3*N);
save codedseq bn Rnt -V6
```

### 6.6.2 Decoding and hard decision

We suppose in our simulations below that the received sequence is coded with binary values 0 and 1.

**Decoding with a majority function**

Upon receipt of the coded sequence, that sequence includes 3-bit words from which one has to reconstruct \(b(n)\). Let \(b(n|k)\) the reconstructed bit at time \(n\)
knowing \( r_0(k) \), \( r_1(k) \) and \( r_2(k) \). Decoding provides:

\[
\begin{cases}
  b_{n|n} = r_0(n) \\
  b_{n-1|n} = r_1(n) + r_2(n) \\
  b_{n-2|n} = r_0(n) + r_1(n)
\end{cases}
\]

With the blocks number \( n + 1 \) and \( n + 2 \), there are two other solutions for reconstructing of \( b_n \):

\[
\begin{cases}
  b_{n+1|n+1} = r_0(n + 1) \\
  b_{n|n+1} = r_1(n + 1) + r_2(n + 1) \\
  b_{n-1|n+1} = r_0(n + 1) + r_1(n + 1)
\end{cases}
\]

\[
\begin{cases}
  b_{n+2|n+2} = r_0(n + 2) \\
  b_{n+1|n+2} = r_1(n + 2) + r_2(n + 2) \\
  b_{n|n+2} = r_0(n + 2) + r_1(n + 2)
\end{cases}
\]

**Figure 6.17** – Decoding the sequence. We use the notation \( b_{n|k} \) for the bit reconstructed using \( r_0(k), r_1(k) \)...

We use a majority function to get the estimated value \( \tilde{b}_n \):

\[
\tilde{b}_n = b_{n|n}b_{n|n+1} + b_{n|n+1}b_{n|n+2} + b_{n|n+2}b_{n|n}
\]  
(6.46)

The program `decoderexple.m` gives an example of decoding of the sequence provided by `coderexple.m`.

```matlab
%===== decoderexple.m =====
clear all, load codedseq
P=3; Nc=length(Rnt); N=Nc/P;
nberr=10; % error number
idxerr=randi([1,Nc],1,nberr);
serr=zeros(1,Nc); serr(idxerr)=1;
Rnte=mod(Rnt+serr,2); % received seq.
%===== for k=1:N-P+1
  idxd=P*(k-1)+1; idxf=idxd+P*P-1;
  bl=reshape(Rnte(idxd:idxf),P,P)';
%===== b(n|n)
  b0=bl(1,3);
%===== b(n|n+1)
  b1=mod(bl(2,1)+bl(2,2),2);
```
\[ b(n|n+2) = \text{mod}(b(3,2) + b(3,3), 2); \]
\[ b(k) = \text{mod}(b0 \cdot b1 + b0 \cdot b2 + b1 \cdot b2, 2); \]

**Viterbi Algorithm**

The principle of the Viterbi algorithm [41] is based on the minimization of a distance between the estimated and received sequences. Consider the trellis (Figure 6.18) corresponding to the encoder shown in Figure 6.14.

![Trellis Diagram](image)

**Figure 6.18** – The trellis of the encoder; the initial state is chosen to be 00

We execute the following steps:

1. **Encoding of the sequence**: starting from the state 00 with a sequence 0 1 1 0 1, we obtain (Figure 6.19) the emitted sequence 000 111 011 010 001.

![Encoded Sequence Diagram](image)

**Figure 6.19** – Encoding the sequence 0 1 1 0

2. **Addition of noise to the sequence**: the emitted sequence suffers from errors. Let us suppose the following sequence is received: 000 110 111
010 000 (corresponding to the error vector 000 001 100 000 001). From the initial state 00, we mark each state with its minimal Hamming distance from the initial state and label it with the previous state number on the shortest path to it (Figure 6.20).

3. Marking the path: at each step an update of the distance (metric update) is carried out for each state, a comparison and a marking of the previous state, given the minimal distance. This is called an ACS (Add, Compare, Select) operation. Hence, the state 00 ($b_4b_3$) from Figure 6.20 is marked with a distance equal to 4 and a previous state $A$.

4. Reading backwards the path of best length: the path ($B$, $D$, $C$, $A$, $A$) which leads to the state with the shortest length gives us the emitted sequence 0 1 1 0 1. The minimum value 3 associated with the state $C$ gives the number of errors in the sequence.

Simulation

Let $b$ be the edge characterized by the triplet $r_0^{(b)}, r_1^{(b)}, r_2^{(b)}$. Let $R_0, R_1, R_2$ be the received triplet. The Hamming distance is given by:

$$D = \sum_{k=0}^{2} r_k^{(b)} \oplus R_k$$

(6.47)

In the program `decoderv.m`, the calculation of the distances on eight edges is carried out by `delta=sum(xor(prcodes,bl));` where `prcodes` gives the eight triplets (the 3-bit codes provided by the encoder) labeling the eight edges.
clear all

%==== Test sequence
Rnte=[0 0 0 1 1 0 1 1 0 1 0 0 0 0];
P=3; N=length(Rnte)/P;
prcodes=[[0;0;0],[1;1;0],[1;0;0],[0;1;0],...%
[1;1;1],[0;0;1],[0;1;1],[1;0;1]];
pstate=[1 3 1 3;2 4 2 4]; ostate=zeros(4,N-P+2);
wght=zeros(4,1); st=[];

%===== level 1
idxd=1; idxf=3;
bl=Rnte(idxd:idxf)'*ones(1,2);
delta=sum(xor([[0;0;0] [1;1;1]],bl));
wght=[wght(1)+delta(1);0;wght(1)+delta(3);0];
ostate(:,1)=[1;0;1;0]; st=[st wght];

%===== level 2
idxd=4; idxf=6;
bl=Rnte(idxd:idxf)'*ones(1,4);
delta=sum(xor([[0;0;0] [1;0;0],[1;1;1] [0;1;1]],bl));
wght=[wght(1)+delta(1); wght(3)+delta(2);... %
wght(1)+delta(3); wght(3)+delta(4)];

ostate(:,2)=pstate(ti+[0:2:6])';

%===== level 3...
for k=3:N
    idxd=P*(k-1)+1; idxf=idxd+2;
    bl=Rnte(idxd:idxf)'*ones(1,8);
    delta=sum(xor(prcodes,bl));
    [td,ti]=min(reshape(delta+[wght' wght'],2,4));
    wght=td'; st=[st wght];
    ostate(:,k)=pstate(ti+[0:2:6])';
end

%===== backtracking
mpath=zeros(1,N+1); rseq=mpath; mbits=zeros(1,N);
[minm, idx]=min(st(:,end));
idxm=idx(1); mpath(N+1)=idxm;
for k=N:-1:1
    mpath(k)=ostate(idxm,k);
    mbits(k)=floor((idxm-1)/2);
    idxm=ostate(idxm,k);
end

st, ostate, mpath, mbits,

Running the program decoderv.m yields:

<table>
<thead>
<tr>
<th>st</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 2 5 4 4</td>
</tr>
<tr>
<td>0 4 3 2 4</td>
</tr>
<tr>
<td>3 1 2 5 3</td>
</tr>
<tr>
<td>0 5 2 3 5</td>
</tr>
</tbody>
</table>
where:

- \( \text{st}(i,k) \), \( i = 1 \) to 4, \( k = 1 \) to \( N \), gives the length from state \((0,0)\) to the current state \( i \) at step \( k \) on the “best” path to \( i \);

- \( \text{ostate}(i,k) \), \( i = 1 \) to 4, \( k = 1 \) to \( N \), gives the state from which it is reached on the path from \((0,0)\) to \( i \) at step \( k \) (0 if no previous state);

- \( \text{mpath} \) gives the sequence of states on the minimum-length path;

- \( \text{mbits} \) gives the reconstructed sequence bits.

The states are numbered from 1 to 4. The array \( \text{ostate} \) gives the original state during the construction of the trellis. This enables us to backtrack when reconstructing the emitted sequence.

For very long sequences, decoding is done not on the whole received sequence but on blocks (e.g. 189 bits for voice data in the GSM norm).
1. The expression (1.21) for the power can be seen as the result of the filtering of the positive signal $y_n = x_n^2$ by the filter with the finite impulse response $h(k) = 1/N$ for $k \in \{0, \ldots, N-1\}$, and 0 otherwise. The filter function is used as follows to achieve this filtering operation:

$$\text{hp}b=\text{ones}(N,1)/N; \text{pn}=\text{filter}(\text{hp}b,1,x.*x);$$

The program that performs the complete signal activity detection operation is the following:

```matlab
%===== detectkey.m
% input x=DTMF signal
N=100; hp=ones(N,1)/N; % estimation
pn=filter(hpb,1,(x .* x)); % power for N points
pmax=max(pn); mthresh=0.5*pmax;
marker=(pn>mthresh); lmarker=length(marker);
begend=marker(2:lmarker)-marker(1:lmarker-1);
begs=find(begend==+1)-N/2; % start indices
ends=find(begend==-1)-N/2; % stop indices
%===== plotting of the signal
subplot(211),plot(tps,x); set(gca,'xlim',[0 (lx-1)/Fs])
%===== plotting of the transitions
hold on
for k=1:length(begs)
```

```matlab
end
```
The filter output is compared to a threshold chosen equal to half of the maximum instantaneous power. Using the logical expression $p_n > m\text{thresh}$, we then determine the sequence of the parts of the signal where $P_n$ is above the threshold value. By subtracting this sequence from itself translated by 1, we get a sequence of values, where $+1$ indicates the beginning of a signal and $-1$ the end of a signal. This operation corresponds to a numerical derivative that could just as well have been written \[ \text{begend} = \text{marker}(2:1:\text{marker}) - \text{marker}(1:1:\text{marker}-1) \] or \[ \text{begend} = \text{diff}(\text{marker}) \]. In expression (1.21), the calculated power corresponds to the signal portion going from the indices $(n - N + 1)$ to $n$. It is therefore better to consider this measure as the median position $n - N/2$. This is why we subtracted $N/2$ to the positions that were found, which is done by the two following program lines:

\[
\begin{align*}
\text{begs} &= \text{find}(\text{begend}==+1) - N/2; \\
\text{ends} &= \text{find}(\text{begend}==-1) - N/2;
\end{align*}
\]

We represented in Figure H1.1 the estimated instantaneous power.

![Figure H1.1 – Estimated instantaneous power](image.png)
corresponding number. All the frequencies extracted are already known, so there is no use doing a spectral study on the entire frequency axis. It is sufficient to evaluate the spectrum at frequencies that might be contained in the signal, by calculating these four quantities:

\[
Q = \left| \sum_n x(n)e^{2j\pi fn} \right|
\]  \hspace{1cm} (7.1)

for \( f \in \{697/F_s, 770/F_s, 852/F_s, 941/F_s\} \), with \( F_s = 8,000 \) Hz, and to choose the frequency that corresponds to the highest value. The same is done with the group of three high frequencies.

The program detectnum.m determines the phone number associated with the signal:

```matlab
%===== detectnum.m
nbDigitdet=min([length(begs) length(ends)]);
foundNum=[];
for k=1:nbDigitdet
    QB=zeros(4,1); QH=zeros(3,1);
sig=x(begs(k):ends(k)); % signal associated with a number
lsig=length(sig);
%===== for each of the 4 freq.: correlation
for n=1:length(FreqB) % for each freq.
    ps=sig .* exp(2*j*pi*FreqB(n)*(1:lsig)/Fs);
    QB(n)=abs(sum(ps));
end
%===== for each of the 3 freq.: correlation
for n=1:length(FreqH)
    ps=sig .* exp(2*j*pi*FreqH(n)*(1:lsig)/Fs);
    QH(n)=abs(sum(ps));
end
%===== maxima
[bid, indB]=max(QB); [bid, indH]=max(QH);
detF=[FreqB(indB) FreqH(indH)];
%===== table look-up
jj=1;
while sum(Freqskeys(jj,:)-detF), jj=jj+1; end
foundNum=[foundNum keys(jj)];
end
disp(sprintf('******* The number is : %s',foundNum))
```

**H2 Additional information on filtering**

H2.1 (Filter architecture) (see p. 19)
1. We can write: 

\[
\begin{align*}
  x_p(n) &= b_p i(n) - a_p o(n) \\
  x_{p-1}(n) &= b_{p-1} i(n) - a_{p-1} o(n) + x_p(n-1) \\
  &\vdots \\
  x_1(n) &= b_1 i(n) - a_1 o(n) + x_2(n-1) \\
  0 &= b_0 i(n) - o(n) + x_1(n-1)
\end{align*}
\]

\[
\Rightarrow \left\{ \begin{array}{l}
  x_1(n) = b_1 i(n) - a_1 o(n) + b_2 i(n-1) - \cdots \\
  \quad \cdots + b_p i(n-p+1) - a_p o(n-p+1) \\
  x_1(n-1) + b_0 i(n) = o(n)
\end{array} \right.
\]

In terms of the \( z \)-transform, we get, as expected, the transfer function:

\[
H(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_p z^{-p}}{1 + a_1 z^{-1} + \cdots + a_p z^{-p}}
\]

2. The state representation associated with this architecture is:

\[
\begin{pmatrix}
  x_1(n) \\
  \vdots \\
  x_p(n)
\end{pmatrix} =
\begin{pmatrix}
  -a_1 & 1 & 0 & \cdots & 0 \\
  -a_2 & 0 & \ddots & \ddots & \vdots \\
  \vdots & \vdots & \ddots & \ddots & 0 \\
  -a_p & 0 & \cdots & 1 & 0
\end{pmatrix}
\begin{pmatrix}
  x_1(n-1) \\
  \vdots \\
  x_p(n-1)
\end{pmatrix} +
\begin{pmatrix}
  b_1 - b_0 a_1 \\
  \vdots \\
  b_p - b_0 a_p
\end{pmatrix}
\]

\[
o(n) = \begin{pmatrix}
  1 & 0 & \cdots & 0
\end{pmatrix} x(n-1) + b_0 i(n)
\]

The corresponding filtering program is given below. Of course, this design is far from being the optimal one in terms of execution. It would be preferable to have a “\texttt{mex}” function. You can check that it leads to the same output sequence as the one obtained with the filtering function described in the text:

```matlab
function [xout, zs] = filtrerII(num, den, xinp, zi)
%!=================================================!
%! Filtering (Transposed form IIR filter structure)!
%! SYNOPSIS: [xout, zs] = FILTRERII(num, den, xinp, zi)
%! num = [b0 b1 ... bP]
%! den = [1 a1 a2 ... aP]
%! xinp = input sequence
```


%! zi = initial state !
%! xout = output sequence !
%! zs = final state !
%===========================================!
lden=length(den); lnum=length(num);
if lden < lnum, den(lnum)=0; lden=lnum; end
if lnum < lden, num(lden)=0; end
ld=lden-1; N=length(xinp); av=zeros(ld,1); bv=av;
av(:)=den(2:lden); bv(:)=num(2:lden);
if nargin==3, zi=zeros(ld,1); end;
if length(zi)<ld, zi(ld)=0; end
zzi=zeros(ld,1); zzi(:)=zi; zs=zzi;
%===== state representation
b0=num(1); ma=compan([1;av])';
vb=bv - b0 * av; vc=[1 zeros(1,ld-1)]; cd=b0;
%===== filtering
for k=1:N,
   zsn =ma * zs + vb * xinp(k);
   xout(k)=vc * zs + cd * xinp(k); zs=zsn;
end

3. We can express the initial state reconstruction by:

\[
    x_k(0) = - \sum_{\alpha=0}^{k-1} b_\alpha i(k - \alpha) + \sum_{\alpha=0}^{k-1} a_\alpha o(k - \alpha)
\]

This leads us to the state reconstruction program:

```matlab
function zi=filtricII(num,den,xinp,xout)
%===========================================!
% Reconstruction of the initial state for a Transpose-Form IIR structure
% SYNOPSIS: zi=FILTRICII(num,den,xinp,xout)
% num = [b0 b1 ... bP] !
% den = [1 a1 a2 ... aP] !
% xinp = input sequence !
% xout = output sequence !
% zi = reconstructed initial state !
%===========================================!
lden = length(den); lnum = length(num);
if lden<lnum, den(lnum)=0; lden=lnum; end
if lnum<lden, num(lden)=0; end
ld=lden-1; numv=zeros(lden,1); denv=numv;
umv(:)=num; denv(:)=den;
% lnx = length(xinp); lny = length(xout);
if lnx<ld, xinp(ld)=0; end
if lny<ld, xout(ld)=0; end```
ysv=zeros(1,ld); xev=ysv; ysv(:)=xout(1:ld);
xev(:)=xinp(1:ld);
zi=filtrerII(denv,1,ysv)+filtrerII(-numv,1,xev);

H2.2 (Parallel implementation of the FIR filtering) (see p. 21)

Type:

```matlab
%===== polyphase.m
x0=[1:103]; lx0=length(x0); M=4;
b=0.3; N=25; h=rif(N,b);
%===== M-polyphase filters (with insertion of zeros
% for the processing)
hp=zeros(M,N);
for k=1:M, hp(k,1:M:N-k+1)=h(k:M:N); end
%===== result of the filtering without polyphase
z1=filter(h,1,x0);
%===== polyphase processing
z2=zeros(M,lx0);
for k=1:M,
  xx = [zeros(1,k-1) x0(1:lx0-k+1)];
  z2(k,:)=filter(hp(k,:),1,xx);
end
xx = sum(z2); [xx(1:lx0)' z1(1:lx0)'
```

H2.3 (FFT filtering) (see p. 28)

1. The gain at the frequency 0 is equal to the sum of the impulse response coefficients.

2. Type:

```matlab
%===== filtragefft1.m
nfft=256; freq=[0:nfft-1]/nfft;
hn=[0.0002 0.0134 0.0689 0.1676 0.2498 ...
  0.2498 0.1676 0.0689 0.0134 0.0002];
nh = length(hn);
N=128-nh; temps=[0:N-1]; f0=.15; f1=.3;
x=sin(2*pi*f0*temps) + sin(2*pi*f1*temps);
%===== processing using the convolution
hn = hn / sum(hn); y=filter(hn,1,x);
subplot(311); plot(temps,[x' y'])
subplot(312); plot(freq, abs(fft(hn,nfft)));
subplot(313); plot(freq, abs(fft([x' y'],nfft)));
```

3. Then type:

```matlab
%===== filtragefft2.m
xcompl = [zeros(1,nh) x]; nxc = length(xcompl);
```
Hints and Solutions 193

```matlab
hns = fft(hn,nxc); xcs = fft(xcompl,nxc);
yns = xcs .* hns; yn = real(ifft(yns));
figure(3)
plot(y); hold; plot(yn(1+nh:nxc),'or'); hold;
```

4. Then type:

```matlab
%===== filtragefft3.m
% Processing with blocks of length P
kp=floor(nxc / (P-nh)); % number of blocks
y=[];
for k =0:kp-1,
    kdb=(P-nh)*k;
    xbloc=xcompl(kdb+1:kdb+P); % overlap
    xbs=fft(xbloc,P); yns=hns .* xbs;
    yn = real(ifft(yns)); y=[y yn(nh+1:P)];
end
hold; plot(y,'ob'); hold; grid
```

**H2.4 (Band-pass filter based on a comb filter)** (see p. 32)

A pole was placed in the first cell of the low-pass filter from Figure 2.10 in order to cancel the zero placed at the frequency 0 in the second cell. Therefore, all we need to do to design a real pass-band filter around the frequency $m/M$ is to precede the filter with another filter with the transfer function:

$$F_m(z) = \frac{1}{(1 - w_mz^{-1})(1 - w_m^*z^{-1})} = \frac{1}{1 - 2\cos(2\pi m/M)z^{-1} + z^{-2}}$$

This leads to $H_z(z) = F_m(z)(1 - z^{-M})$ which is still an FIR filter. Figure H2.1 shows, for $M = 16$, the frequency response of the low-pass filter and of the band-pass filter for $m = 3$.

The following program plots the frequency responses of the two filters:

```matlab
%===== mcomb.m
M=16; m=(0:M-1); Lfft=512; fq=(0:Lfft-1)/Lfft-1/2;
fq1=(Lfft/2+1:Lfft);fq2=(1:Lfft/2);
%===== low-pass and band-pass comb filters
ht=ones(1,M); k=4; gt=2 * ht .* cos(2*pi*k*m/M);
hf=abs(fft(ht,Lfft)); hf=10*log10(hf / max(abs(hf)));
gf=abs(fft(gt,Lfft)); gf=10*log10(gf / max(abs(gf)));
subplot(211); plot(fq,[hf(fq1) hf(fq2)]); ```
**Figure H2.1** – Frequency response: \( m = 0 \) (low-pass) and \( m = 3 \) (band-pass)

```
set(gca,'ylim',[-20 0]); grid
subplot(212); plot(fq,[gf(fq1) gf(fq2)]);
set(gca,'ylim',[-20 0]); grid
```

**H3 Image Processing**

**H3.1 (Projection onto a screen (text on page 51))**

1. Calculating the projection:

   ```
   function [pos2D,lm]=projconique(Ds,Dr,tb3N)
   %!---------------------------------------------------------------!
   %! Dr = distance object reference origin - observer 
   %! Ds = distance from screen to observer 
   %! tb3N = array(3,N) [[x1;y1;z1],...,[xN;yN;zN]] 
   %! pos2D = coordinates in the plane 
   %!---------------------------------------------------------------!
   delta=Ds ./ (Dr-tb3N(1,:,:));
   pos2D = [tb3N(2,:,:) .* delta ; tb3N(3,:,:) .* delta];
   ```

2. The rotations around the axes by an angle \( \varphi_x, \varphi_y \) where \( \varphi_z \) are represented by the three matrices:

\[
R_x = \begin{bmatrix}
1 & 0 & 0 \\
0 & \cos \varphi_x & -\sin \varphi_x \\
0 & \sin \varphi_x & \cos \varphi_x
\end{bmatrix}, \quad R_y = \begin{bmatrix}
\cos \varphi_y & 0 & \sin \varphi_y \\
0 & 1 & 0 \\
-\sin \varphi_y & 0 & \cos \varphi_y
\end{bmatrix}
\]
and \( R_z = \begin{bmatrix} \cos \varphi_z & -\sin \varphi_z & 0 \\ \sin \varphi_z & \cos \varphi_z & 0 \\ 0 & 0 & 1 \end{bmatrix} \)

which give us the coordinates of the points transformed into the chosen system of axes.

Functions of rotation:

```matlab
function vxp=rotx(vp,phix)
% !=========================================!
% ! phix = rotation around x-axis (degrees) !
% ! vp=(3,N)-matrix, vxp=(3,N)-matrix       !
% !=========================================!
phix=phix * pi / 180;
Rx=[1 0 0;...
    0 cos(phix) -sin(phix);...
    0 sin(phix) cos(phix)];
% yp=Rx*vp;
return

function vyp=roty(vp,phiy)
% !=========================================!
% ! phiy = rotation around y-axis (degrees) !
% ! vp=(3,N)-matrix, vyp=(3,N)-matrix       !
% !=========================================!
phiy=phiy * pi / 180;
Ry=[cos(phiy) 0 sin(phiy);...
    0 1 0;...
    -sin(phiy) 0 cos(phiy)];
% yp=Ry*vp;
return
```

Figure H3.1 – Example of projection onto the plane orthogonal to \( Ox \)
function vzp=rotz(vp,phiz)
    %!=========================================!
    %! phiz = rotation around z-axis (degrees)  
    %! vp=(3,N)-matrix, vzp=(3,N)-matrix        
    %!=========================================!
    phiz=phiz * pi / 180;
    Rz=[cos(phiz) -sin(phiz) 0;...
        sin(phiz) cos(phiz) 0;...
        0 0 1];
    vzp=Rz*vp;
return

3. The difficulty stems from the fact that the function `surf` accepts only the functions defined by \( z = f(x, y) \). The test program giving Figure H3.2 is:

```matlab
%===== tstproj3D.m
xrate=2.5; sapert=24;
%===== original image
[tx,ty,tz,zz,xx]=newimg(); L=length(tx);
subplot(221), surf(tz,tx,ty);
xlabel('z'), ylabel('x'), zlabel('y')
view(135,30)
%=======
pos2D=[]; tzm=max(tz); Dr=xrate*tzm; Ds=sapert;
tb3=zeros(3,L,L);
tb3(1,:,:)=zz; tb3(2,:,:)=xx; tb3(3,:,:)=ty;
tb3N=reshape(tb3,3,L*L);
%===== conic projection
subplot(222)
[pos2D]=conicproj(Ds,Dr,tb3N);
plot(pos2D(1,:),pos2D(2,:),'.'); grid on
%===== translation + rotation / Ox
vyp=roty([tb3N(1,:);tb3N(2,:)-7;tb3N(3,:)],30);
[pos2D]=conicproj(Ds,Dr,vyp);
subplot(223), plot(pos2D(1,:),pos2D(2,:),'.')
grid on
%===== using rotate function
alpha=20; % degrees
subplot(224), hh=surf(tz,tx,ty);
zdir=[0 0]; % phi=0, theta=0
rotate(hh,zdir,alpha);
```
H4 Numerical calculus

H4.1 (Full-wave rectifier and simulation) (page 71)

1. The full-wave rectifier is periodic with period $T_d = T_0/2$ where $T_0 = 1/F_0$. It is expandable in a Fourier series. The Fourier coefficients are:

$$X_n = \frac{A}{T_d} \int_0^{T_d} \sin(2\pi F_0 t)e^{-2j\pi nt/T_d} dt$$

$$= \frac{1}{T_d} \int_{-\infty}^{+\infty} x_T(t)e^{-2j\pi nt/T_d} dt = \frac{1}{T_d} X_T(n/T_d)$$

where $x_T(t) = A\text{rect}_{T_d}(t-T_d/2)\sin(2\pi F_0 t)$. The Fourier transform of $x_T(t)$ is denoted by $X_T(F)$. We can write:

$$x_T(t) = A\text{rect}_{T_d}(t-T_d/2)\frac{1}{2j}(\exp(2j\pi F_0 t) - \exp(-2j\pi F_0 t))$$

If we use the modulation property satisfied by the Fourier transform of
rect_{T_d}(t - T_d/2), we get:

$$X_T(F) = \frac{A}{2\pi} \frac{\sin(\pi(F - F_0)T_d)}{F - F_0} e^{-j\pi(F - F_0)T_d}$$

$$-\frac{A}{2\pi} \frac{\sin(\pi(F + F_0)T_d)}{F + F_0} e^{-j\pi(F + F_0)T_d}$$

and therefore:

$$X_n = \frac{1}{T_d} X_T(n/T_d) = -\frac{2A}{\pi} \frac{1}{4n^2 - 1}$$

The continuous component is $X_0 = 2A/\pi$.

2. If we apply the Fourier transform to the differential equation, we get:

$$2j\pi RC FY(F) + Y(F) = X(F) \Rightarrow H(F) = \frac{1}{1 + 2j\pi RC F}$$

3. If the RC filter has a constant $RC \gg 1/F_0$ such that the components with the frequency $\pm 2kF_0$, where $k \geq 2$, are negligible, the continuous component and the first harmonic are all that is left in the output. By remembering that the signal $\exp(2j\pi F t)$, after filtering, leads to $H(F) \exp(2j\pi F t)$, the output signal has the expression:

$$y(t) \approx H(0)X_0 + H(-2F_0)X_{-1}e^{-4j\pi F_0 t} + H(2F_0)X_1e^{4j\pi F_0 t}$$

$$= \frac{2A}{\pi} + 2|H(2F_0)||X_1|\cos(4\pi F_0 t + \Phi)$$

4. Because $T_s \ll RC$, approximating the derivative by:

$$\frac{\Delta y(t)}{\Delta t} = F_s(y_s(n) - y_s(n - 1))$$

is acceptable. The differential equation becomes the recursive equation:

$$F_s(y_s(n) - y_s(n - 1)) + \frac{1}{RC} y_s(n) = \frac{1}{RC} x_s(n)$$

Hence the simulation is equivalent to the filtering:

$$y_s(n)(1 + \tau) - y_s(n - 1) = \tau x_s(n)$$

if we choose $\tau = T_s/RC$. 
5. The following program illustrates the cases of the half-wave and full-wave rectifiers (Figure H4.1):

```matlab
%===== C2altern.m
% Rectifiers
Fs=5000; Ts=1/Fs; N=800; fa=50; A=220*sqrt(2);
t=(0:N-1)/Fs; xa=A*sin(2*pi*fa*t);
xrS=.5*(sign(xa)+1).*xa; % half-wave
xrD=abs(xa); % full-wave
%===== simulation of RC(dy/dt) + y = x
RC=.02; tau=Ts/RC; mu=1/(1+tau); nu=tau*mu;
yS=filter(nu,[1 -mu],xrS); yD=filter(nu,[1 -mu],xrD);
subplot(211); plot(t,xrS,'-',t,yS,[0 max(t)],[A/pi A/pi]); grid;
subplot(212); plot(t,xrD,'-',t,yD,'-',...
[0 max(t)],[2*A/pi (2*A)/pi]); grid
```

Figure H4.1 – Simulation of the half-wave and full-wave rectifiers
H4.2 (Simulation in the presence of a ZOH) (see p. 74)

1. If we let $\Phi(t) = e^{At}$ and notice that $e(u)$ remains constant between $kT$ and $(k+1)T$, we have:

$$x((k+1)T) = \Phi(T)x(kT) + \int_{kT}^{(k+1)T} \Phi((k+1)T - u)b e(kT)du$$

with:

$$\Psi_{kT} = \int_{kT}^{(k+1)T} \Phi((k+1)T - u)du = \int_0^T \Phi(u)du = \Psi(T)$$

The output at the sampling times is given by $s(kT) = c^T x(kT)$.

2. The inter-sample response is given by:

$$\begin{cases} 
  x(t) = \Phi(t - kT)x(kT) + e(kT)\Psi(t)b \\
  s(t) = c^T x(t)
\end{cases}$$

with $\Psi(t) = \int_0^{t-kT} \Phi(u)du$. In order to know the system’s behavior between $kT$ and $(k+1)T$, we need to know $x(kT)$ and $\Psi(t)$.

3. Type the following program:

```matlab
%===== Crepetats.m
% Sampling frequency (1/T)=10 Hz
% Simulation duration tmax=10 s
% Initial conditions x0
A=[0 1;-1 -1.4]; b=[0;1]; c=[1 0];
T=.1; tmax=10; x0=zeros(2,1);
[t,s]=Crepind(A,b,c,T,tmax,x0);
plot(t,s,'x'); grid
```

The `Crepind` function computes the system’s step response:

```matlab
function [realt,xout] = Crepind(A,b,c,Ts,tmax,x0)
%!=================================================================
%! Step response of a linear system
%! SYNOPSIS: [realt,xout]=CREPIND(A,b,c,Ts,tmax,x0)
%! Input: (A,b,c) = state representation
%! Ts = sampling frequency
%! tmax = observation duration
%! x0 = initial state
%! Output: realt = real time
%! xout = response
```
4. Changing from continuous-time over to discrete-time requires the use of the bilinear transform obtained with the `nbilin` function:

```matlab
4. Changing from continuous-time over to discrete-time requires the use of
the bilinear transform obtained with the `nbilin` function:

```
H4.3 (Non-minimal system) (see p. 75)

1. Simulation:

```matlab
%===== repetat.m
clear
%===== system definition
atc=[-11/4 -11/8 -5/4;27/4 11/8 21/4;15/8 19/16 5/8];
btc=[1;-1;-1/2]; ctc=[3/8 1/2 -1/4];
%===== parameters
T=.2;  % sampling period
x0=[0;0;0];  % initial state
tpm=130;  % duration of the simulation
Npts=tpm/T; tps=[0:Npts]*T; e=ones(1,Npts);
%===== discrete time equivalence
atd=expm(atc*T); abs(eig(atd))
btd=inv(atc)*(atd-eye(3,3))*btc;
ctd=ctc;
%===== simulation
x=x0; s=[ctd*x0]; m=[max(max(abs(x0)))]; zz=[x0'];
for k=1:Npts
    x=atd*x+btd*e(k); s(k+1)=ctd*x;
    zz=[zz;x']; m(k+1)=max(max(abs(x)));
end
figure(1); title('Norm of the state vector')
plot(tps,s,'-'); grid
```

2. The state vector’s evolution indicates an instability.
3. The transfer function calculation ends up with:

\[ G(s) = c^T(sI - A)^{-1}b = \frac{1}{s^2 + s + 1} \]

An “unstable” pair is eliminated. From an input-output point of view, the system is stable. However, the presence of initial conditions causes the output to diverge because the unstable pole is not simplified in the free part of the response. It can be shown that the discrete time system has the same property.

4. The system diverges because of the computation noise (Figure H4.3). Calculating the roots of \( \text{poly(phi-psib*c')-poly(phi)} \) and of \( \text{poly(phi)} \) shows that there are a pole and a zero which are almost identical. As there is not really – physically – a simplification, this is another reason for the divergence of the output. In the area of the control of a system, here the filter, we can show that the simplification is linked to the nonobservable and/or noncontrollable nature of the system.

![Figure H4.3 – Response for a longer simulation time (tpm=130)](image)

**H4.4 (RK-2 solver and ode15i function)** (see p. 83)

1. For Heun’s algorithm, we have \( h_1 = 3T/4 \), from which we obtain \( a_{2,0} = T/4 \) and \( a_{2,1} = 3T/4 \).

```matlab
% testheun.m
% fixed points: [0;0] and [c/d;a/b] (omeg^2=ac)
a=1; b=1; c=1; d=1;
y0=[3/2;3]; t0=0;
T=0.1; t=[0:T:2.35*pi]; Lt=length(t);
y=[y0,zeros(2,Lt-1)];
for k=1:Lt-1
```

2. use of ode15i:

```matlab
function [t,y]=solve_lv2(a,b,c,d,y0)
    tspan=[0:0.1:2.35*pi]; yp0=zeros(2,1);
    options = odeset('RelTol',1e-4,'AbsTol',[1e-6 1e-10]);
    [t,y] = ode15i(@lv,tspan,y0,yp0,options);
    %==== Lotka-Volterra equation
    function res=lv(t,y,yp)
end
```

```matlab
%==== testlv2.m
% fixed points: [0;0] and [c/d;a/b] (omeg^2=ac)
a=1; b=1; c=1; d=1;
xinit=[3/2 1.1 1.1 1.1 1.1 1.3 1.3 1.3 1.2 4];
t0=0; t=[0:0.1:2.35*pi];
for k=1:size(xinit,2)
    y0=xinit(:,k);
    [t,y] = solve_lv2(a,b,c,d,y0); hold on
    plot(y(:,1),y(:,2))
    plot(y0(1),y0(2),'or')
end
set(gca,'xlim',[0 4.5],'ylim',[0 4.5])
grid on; hold off
```
Figure H4.4 – Comparison of the result obtained with Heun’s algorithm and that obtained with ode45 (exercise 4.3)

```matlab
res=[yp(1)-a*y(1)+b*y(1)*y(2);
    yp(2)+c*y(2)-d*y(1)*y(2)];
end
```

H4.5 (Newton’s method (see p. 88))

1. Recurrence relation:

\[ x_{n+1} = x_n - \frac{y_n}{y'_n} \]

Note that there is convergence – fixed-point theorem – if \( 0 < |\dot{h}(x)| < A < 1 \) on \((V)\) \((h(x) = x - f(x)/\dot{f}(x))\). Then, the solution is given by \( h(x) = x \), therefore \( f(x) = 0 \). The condition on \( \dot{h}(x) \) is also written as:

\[ 0 < \left| f(x) \frac{\ddot{f}(x)}{\dot{f}^2(x)} \right| < A < 1 \tag{7.2} \]

which causes the first and second derivatives not to take a value of zero over \((V)\) and therefore the concavity does not change.
function xnp1=recn(xn,mp)
    mdpot=polyder(mp); % derivative
    yn=polyval(mp,xn); ydotn=polyval(mdpot,xn);
    if ydotn==0 then
        error('Divide by zero, modify the initial value')
    end
    xnp1=xn-yn/ydotn;
end

2. Program:

%==== exonewton.m
% roots: 1,sqrt(2),3
myp=[1,(-4-sqrt(2)),(3+4*sqrt(2)),-3*sqrt(2)];
mddeg=length(myp)-1;
nrts=zeros(1,mdeg); % estimated roots
xn=3; xnp1=2; % initial value
myeps=2*eps;
for k=1:mdeg
    disp('Poly='),disp(myp)
    while abs(xnp1-xn)>myeps
        xn=xnp1; xnp1=recn(xn,myp);
    end
    nrts(k)=xnp1; myp=deconv(myp,[1,-xnp1]);
    xn=3; xnp1=2;
end
    disp('Poly='), disp(myp)
    disp('Roots='), disp(nrts)

H4.6 ( Newton–Raphson method (see p. 90))

function [xc,yc,cErr,poss,k,dv]=CalcXY(pol,x0,y0,teps,nblps,verb)
%!=================================================================
%! pol = polynomial, coefficients in descending powers
%! (x0,y0) = initial value (x0+i*y0)
%! teps = used by the test: |Delta x|+|Delta y|<teps
%! nblps = max. number of loops
%! verb = true or false, for debugging purposes
%! (xc,yc) = one root (xc+i*yc)
%! cerr = error code
%! k = number of loops
%! dv = value of |Delta x|+|Delta y| when exiting
%!=================================================================
if nargin<6, verb=false; end
deg=length(pol)-1; P=zeros(1,deg+1); P(:)=pol;
pp=P; ppd=[deg:-1:1] .* P(1:end-1); % deriv. (see polyder)
df=1; dv=1; poss=zeros(nblps,4);
%===== tstCalcXY.sce
% calculates one root (for others polynomial division
% is implemented by the deconv function)
clear all
pol=[2 -3 1]; % Pol(z)=2z^2-3z+1
verb=true;
deg=length(pol)-1;
m=1; mroots=zeros(deg,2);
while deg>1
    [xc,yc,cErr,poss,k,dv]=CalcXY(pol,pi,exp(1),10^(-5),100,verb);
    mroots(m,1)=xc; mroots(m,2)=yc;
    if verb,
        xc,yc,cErr,poss,k,dv
        subplot(211); plot(poss(1:k,1),poss(1:k,2),'-'), hold on
        plot(poss(1:k,1),poss(1:k,2),'-'), grid on
        subplot(212); plot(poss(1:k,3),poss(1:k,4),'-'), hold on
        plot(poss(1:k,3),poss(1:k,4),'-'), grid on
        pause
    end
    m=m+1;
    pol=deconv(pol,[1 (-xc-1i*yc)]); % pol. division
deg = deg - 1;
end
mroots

Figure H4.5 – Evolution of $z$ and $P(z)$ during the calculation of the root $z_c = 1$ for the chosen initial condition

H4.7 (Plot of a natural cubic spline)  (see p. 99)

1. Main program:

```
%===== natcubicspline.m
pts=[0;0;2;3;1]+1i*[0;2;0;2;1];
msd=secondderiv(pts);
figure(1)
npts=20; drawcs(pts,msd,npts)
grid on
```

Calculation of the second derivatives on the basis of the control points $A$, $B$, $C$, $D$ and $E$:

```
function msd=secondderiv(ptlist)
%!===================================================================
%! SYNOPSIS: msd=SECONDDERIV(ptlist)
%!===================================================================
```
%! ptlist = vector (N*1) of control points !
%! (complex) !
%! msd = vector (N*1) of second derivatives !
%! (msd(1)=0, msd(N)=0) !
%!======================================================================!
N=length(ptlist);
\tt=toeplitz(ptlist(3:N),...
[ptlist(3),ptlist(2),ptlist(1)]);
b=tt*[1;-2;1];
C=[4;1;zeros(N-4,1)]; R=C.^
A=toeplitz(C,R)/6;
msd=A\b; msd=[0;msd;0];

The derivatives can be calculated by:

function md=calcderiv(ptlist,msd)
%!======================================================================!
%! SYNOPSIS: CALCDERIV(ptlist,msd) !
%! ptlist = vector (N,1) of control points !
%! msd = second derivatives !
%!======================================================================!
N=length(ptlist); md=zeros(N,1);
md=diff(ptlist)+diff(msd)/3-msd(2:N)/2;
md(N)=ptlist(N)-ptlist(N-1)+msd(N-1)/6;

2. Plot function:

function drawcs(ptlist,msd,npts)
%!======================================================================!
%! SYNOPSIS: DRAWCS(ptlist,msd,npts) !
%! ptlist = vector (N,1) of control points !
%! msd = second derivatives !
%! npts = number of points for each !
%! segment P(k),P(k+1) !
%!======================================================================!
N=length(ptlist); t=[0:npts]/npts;
t2=t.*t; t3=t2.*t;
hold on
for k=1:N-1
pt=(msd(k+1)-msd(k))*t3/6 + msd(k)*t2/2 + ptlist(k);
pt=pt+(ptlist(k+1)-ptlist(k)-msd(k)/3-msd(k+1)/6)*t;
plot(real(pt),imag(pt),':r')
plot(real(ptlist(k)),imag(ptlist(k)),'o')
end
plot(real(ptlist(N)),imag(ptlist(N)),'o')
hold off

H4.8 (Relaxation method) (see page 102)
1. In the Jacobi method, we have $M = D$ and $N = L + U$ and, in the Gauss–Seidel method, $M = D - L$ and $N = U$.

\[
\Delta_j = \det(\lambda I - (D)^{-1}(L + U)) = \det(D^{-1})\det(\lambda D - L - U) \quad (7.3)
\]

and

\[
\Delta_{gs} = \det(\lambda I - (D - L)^{-1}U) = \det((D - L)^{-1})\det(\lambda D - \lambda L - U) \quad (7.4)
\]

**Example:** in the case of $N = 5$, the dimension of the matrix is $(N - 2) \times (N - 2) = (3 \times 3)$. If we write $d_j = \det(\lambda D - L - U)$ and $d_{gs} = \det(\lambda D - \lambda L - U)$:

\[
d_j = \det \begin{bmatrix} 4\lambda & 1 & 0 \\ 1 & 4\lambda & 1 \\ 0 & 1 & 4\lambda \end{bmatrix} \quad \text{and} \quad d_{gs} = \det \begin{bmatrix} 4\lambda & 1 & 0 \\ \lambda & 4\lambda & 1 \\ 0 & \lambda & 4\lambda \end{bmatrix}
\]

and we verify the relation between spectral radii:

\[
d_j = 4\lambda(16\lambda^2 - 1) - 4\lambda = 8\lambda(8\lambda^2 - 1) \Rightarrow \rho_j = \sqrt{1/8}
\]

\[
d_{gs} = 4\lambda(16\lambda^2 - \lambda) - 4\lambda^2 = 8\lambda^2(8\lambda - 1) \Rightarrow \rho_{gs} = 1/8
\]
The program \texttt{testtridiag.m}:

\begin{verbatim}
%===== testtridiag.m
for P=3:8
    [pJ,pGS,rhoJ,rhoGS]=dettridiag(P);
    [rhoGS,rhoJ^2]
pJ,pGS
end
\end{verbatim}

with the function \texttt{dettridiag}:

\begin{verbatim}
function \[pJ,pGS,rhoJ,rhoGS\]=dettridiag(P)
%!=============================================!
%! SYNOPSIS \[pJ,pGS,rhoJ,rhoGS\]=dettridiag(P) !
%! P : system dimension !
%! pJ,pGS : polynomials (Jacobi, Gauss-Seidel) !
%! rhoJ,rhoGS : spectral radius !
%!=============================================!
R=[4 1 zeros(1,P-2)];
A=toeplitz(R',R)/6;
D=diag(diag(A)); L=-tril(A,-1); U=-triu(A,1);
%!===== Jacobi
M=D; N=L+U; mM=inv(M)*N;
vJ=eig(mM); rhoJ=max(abs(vJ));
pJ=poly(mM)*(4^P);
%!===== Gauss-Seidel
M=D-L; N=U; mM=inv(M)*N;
vGS=eig(mM); rhoGS=max(abs(vGS));
pGS=poly(mM)*(4^P);
\end{verbatim}

gives:

<table>
<thead>
<tr>
<th>n</th>
<th>(d_j)</th>
<th>(d_{gs})</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>(64\lambda^3 - 8\lambda)</td>
<td>(64\lambda^3 - 8\lambda^2)</td>
</tr>
<tr>
<td>4</td>
<td>(256\lambda^4 - 48\lambda^2 + 1)</td>
<td>(256\lambda^4 - 48\lambda^3 + \lambda^2)</td>
</tr>
<tr>
<td>5</td>
<td>(1024\lambda^5 - 256\lambda^3 + 12\lambda)</td>
<td>(1024\lambda^5 - 256\lambda^4 + 12\lambda^3)</td>
</tr>
<tr>
<td>6</td>
<td>(4096\lambda^6 - 1280\lambda^4 + 96\lambda^2)</td>
<td>(4096\lambda^6 - 1280\lambda^5 + 96\lambda^4 - \lambda^3)</td>
</tr>
<tr>
<td>7</td>
<td>(16384\lambda^7 - 6144\lambda^5 + 640\lambda^3 - 16\lambda)</td>
<td>(16384\lambda^7 - 6144\lambda^6 + 640\lambda^5 - 16\lambda^4)</td>
</tr>
<tr>
<td>8</td>
<td>(65536\lambda^8 - 28672\lambda^6 + 3840\lambda^4 - 160\lambda^2 + 1)</td>
<td>(65536\lambda^8 - 28672\lambda^7 + 3840\lambda^6 - 160\lambda^5 + \lambda^4)</td>
</tr>
</tbody>
</table>
More generally, consider a matrix of the form \( 7.3 \) and of dimensions \((P \times P)\). The notation \(d_j^{(n)}\) represents the determinant of a submatrix of dimension \((n \times n)\). We have:

\[
d_j^{(n)} = 4\lambda d_j^{(n-1)} - d_j^{(n-2)} \quad \text{with} \quad d_j^{(1)} = 4\lambda \quad \text{and} \quad d_j^{(2)} = 16\lambda^2 - 1
\]

The expression of \(d_j^{(n)}\) is:

\[
d_j^{(n)} = \alpha \left(2\lambda + \sqrt{4\lambda^2 - 1}\right) + \beta \left(2\lambda - \sqrt{4\lambda^2 - 1}\right)^n
\]

We know that \(d_j^{(1)} = 4\lambda\) and \(d_j^{(2)} = 16\lambda^2 - 1\). By setting \(\Delta = 4\lambda^2 - 1\), we obtain:

\[
d_j^{(n)} = \left[\left(2\lambda + \sqrt{\Delta}\right)^{n+1} - \left(2\lambda - \sqrt{\Delta}\right)^{n+1}\right] \frac{1}{2\sqrt{\Delta}} \quad (7.5)
\]

The determinant of the \((n \times n)\) matrix associated with the Gauss–Seidel method satisfies:

\[
d^{(n)}_{gs} = 4\lambda_{gs}^{(n-1)} - \lambda d^{(n-2)}_{gs} \quad \text{with} \quad d^{(1)}_{gs} = 4\lambda \quad \text{and} \quad d^{(2)}_{gs} = 16\lambda^2 - \lambda
\]

The expression of \(d^{(n)}_{gs}\) is:

\[
d^{(n)}_{gs} = \alpha \left(2\lambda + \sqrt{4\lambda^2 - \lambda}\right) + \beta \left(2\lambda - \sqrt{4\lambda^2 - \lambda}\right)^n
\]

By setting \(\Delta' = 4\lambda^2 - \lambda\), we find:

\[
d^{(n)}_{gs} = \left[\left(2\lambda + \sqrt{\Delta'}\right)^{n+1} - \left(2\lambda - \sqrt{\Delta'}\right)^{n+1}\right] \frac{1}{2\sqrt{\Delta'}} \quad (7.6)
\]

The eigenvalues are given by \(d^{(P)}_j(\lambda) = 0\) and \(d^{(P)}_{gs}(\lambda) = 0\) with the expressions 7.5 and 7.6:

\[
\lambda_j^{(k)} = \frac{\cos(k\pi/(P + 1))}{2}, \quad k = 1 : P
\]

\[
\lambda_{gs}^{(k)} = \frac{\cos^2(k\pi/(P + 1))}{4}, \quad k = 1 : \lfloor P/2 \rfloor
\]
The eigenvalues of $d_{gs}^{(P)}$ are either positive or null. Type:

\[
\%===== spectralradius.m
P=8; \ [pJ,pGS,rhoJ,rhoGS]=dettridiag(P);
\%===== Jacobi
rr=roots(pJ); rrt=cos([1:P]'*pi/(P+1))/2;
[rr,abs(rr),rrt]
\%===== Gauss-Seidel
rr3=zeros(P,1); mP=fix(P/2);
rr3(1:mP)=rrt(1:mP).^2;
rr2=roots(pGS);
[rr2,abs(rr2),rr3]
\]

2. Convergence of the relaxation method: the state change matrix is given by:

\[
\left( \frac{D}{\omega} - L \right)^{-1} \left( \frac{1-\omega}{\omega} D + U \right) \tag{7.7}
\]

whose determinant is $(1 - \omega)^P$. From this, we deduce that the spectral radius $\rho_r$ is $\geq (1 - \omega)$.

The condition (4.48) ($|\rho_r| < 1$) gives us $0 \leq \omega < 2$.

3. Programming:

\[
\%===== relaxtest.m =====
\]

\[
clear
N=5; eps=1e-6; maxiter=100;
A = 4*eye(N) + diag(ones(1,N-1),1) + diag(ones(1,N-1),-1);
A(1,1)=2; A(N,N)=2;
xex=ones(N,1); b=A*xex;
x0=zeros(N,1); maxit=100;
stcr0=1e-6; merr=[]; w0=(.5:.025:1.74); rhot=[];
for w=w0
\[x,rho,eltime]=relaxmethod(A,b,x0,w,stcr0,maxit);
merr=[merr;sqrt((x-xex)'*(x-xex))];
rhot=[rhot;rho];
end
\]

subplot(211), plot(w0,merr), grid
subplot(212),
plot(w0,rhot,'-','w0,rhot,'xr'), grid
Figure H4.7 – Top: Evolution of the error, and bottom: evolution of the spectral radius, as a function of $\omega$

H4.9 (Cholesky factorization) (see p. 103)

Type:

```matlab
function L=CHOfact(A,L,N0)
%!===================================================================
%! CHOfact calculates in L a lower triangular matrix 
%! so that $A=L*L^H$ (Cholesky factorization) 
%! with $A$ symmetric definite positive 
%! SYNOPSIS: L=CHOFACT(A,L,N0) 
%! N0=size(A,1) 
%! A = symmetric definite positive 
%! L = used as an input allows 
%! the recursivity (initial value []) 
%! EXAMPLE: A=rand(4,4); A=A*A.'; N0=size(A,1); 
%! L=[]; L=CHOfact(A,L,N0); 
%!===================================================================
N=size(A,1); if N==0, return, end 
Lp=zeros(N,1); Lp(1)=sqrt(A(1,1)); 
Lp(2:N)=A(2:N,1)/Lp(1); vec=Lp(2:N); 
B=A(2:N,2:N)-(vec * vec'); 
L=CHOfact(B,L,N0);
```
H5  Speech processing

H5.1 (Compression of a speech signal) (see page 114)

1. Pitch detection:

```matlab
function [vnv,pitch]=detectpitch(sig,trhld,tmin,tmax,energm)
%!Pitch detection using correlation
%!SYNOPSIS:
%! [vnv,pitch]=DETECTPITCH(sig,trhld,tmin,tmax,energm)
%! sig = signal block
%! trhld = correlation treshold
%! tmin,tmax = correlation window
%! energm = energy threshold
%! vnv = TRUE if voiced, otherwise FALSE
%! pitch = pitch period
%!nfa=length(sig); x=zeros(nfa,1); x(:)=sig; ae=x'*x;
if (ae > energm), % energy>trhld
    for T=tmin:tmax
        stmT=x(T:nfa); s0T=x(1:nfa-T+1);
        autoc=stmT'*s0T; etmT=stmT'*stmT; e0T=s0T'*s0T;
        correl(T-tmin+1)=autoc/sqrt(etmT*e0T);
    end
    [corrmax,imax]=max(correl); tfond = imax+tmin-1;
    if (corrmax < trhld),
        vnv=(0==1); pitch=0; return;
    else
        pitch=tfond; vnv=(0==0);
    end
else
    pitch=-1; vnv=(0==1);
end
```

Figure H5.1 shows the shape of the autocorrelation for a block. The maximum is located in $T = 82$, which means that the pitch frequency is roughly $f_0 = 8,000/82 \approx 97.5$ Hz. Determining $T$ can become a difficult task when there are maxima present at the multiples of the pitch period. One solution is to check for the possible presence of a high maximum at sub-multiples of the $x$-coordinate found for the maximum. We can also study the evolution over different consecutive windows by comparing the obtained fundamental frequencies. Bear in mind, finally, that the accuracy can be improved by oversampling the signal beforehand.
2. Coding program (using the xtoa function from page 115):

```matlab
%===== CODE.M
%!===================================================================
%! code.m: Coding a speech signal based on an AR-model
%! INPUT:
%! Signal sampled at 8000 Hz
%! OUTPUT: array tab_cod(N,XX):
%! tab_cod(N,1): energy in the block of signal
%! tab_cod(N,2): pitch period
%! tab_cod(N,3:12): AR coefficients (AR-ordv if voiced sound, AR-ordnv otherwise) or reflection coeffs.
%! Each block has a 240 sample length (30 ms) with an overlap of 60 samples.
%! Uses: xtoa : AR-model coeffts
%! detectpitch : pitch detection
%! ai2ki : reflection coeffs
%!===================================================================
clear
load phrase; %===== vector y
enerm=std(y)^2*.1;
% AR-model orders for voiced and non voiced sounds
ordv=20; ordnv=10;
NbParam=ordv+2;
phrase=phrase-mean(phrase);
%===== parameters
lBloc=240; % block length
recouv=60; % overlap
ltr=lBloc-recouv;
nBlocs=floor((length(phrase)-recouv)/ltr); % nb of blocks
reste=rem(length(phrase)-recouv,ltr);
```

Figure H5.1 – Example of autocorrelation graph
phrase=phrase(1:length(phrase)-reste);
tmin=40; tmax=150; seuil=0.7; % for pitch detection
vnv=zeros(1,nblocs); % boolean "voiced/non voiced"
pitch=zeros(1,nblocs); % pitch period
tab_cod=zeros(nblocs,NbParam); % coeffts of the model
%===== detection "voiced/non voiced"
sprintf('%"Voiced/non voiced" on %5.0f blocks', nblocs)
TIC
for k=1:nblocs,
    ind=(k-1)*ltr;
blocan=phrase(ind+1:ind+lbloc); % analysis block
    [vnv(k) pitch(k)]=detectpitch(blocan,seuil,tmin,tmax,enerm);
end;
TOC
%===== AR-model
sprintf('AR-model')
TIC
preacpar=filter([1 -0.9375],1,phrase); % pre-emphasis
for k=2:(nblocs-1),
    %=======
    if (vnv(k-1) == vnv(k+1)), % correction of
        vnv(k)=vnv(k-1); % errors of detection
        if (vnv(k)==1)
            %==== "voiced" with pitch=mean
                pitch(k)=floor((pitch(k-1)+pitch(k+1))/2);
        else
            %==== "non voiced" with pitch=0
                pitch(k)=0;
        end
    end
end
%===== analysis block
sigbloc=preacpar((k-1)*ltr+1:(k-1)*ltr+lbloc);
if (vnv(k)==1)
    [pcoeff, enrg]=xtoa(sigbloc,ordv); %=======
    %======= coeff_refl=ai2ki(pcoeff); % reflection
    %======= tab_cod(k,3:NbParam)=coeff_refl; % coeffts
    tab_cod(k,3:NbParam)=pcoeff(2:ordv+1)';
    tab_cod(k,1)=enrg;
    tab_cod(k,2)=pitch(k);
else
    [pcoeff, enrg]=xtoa(sigbloc,ordnv);
    %======= coeff_refl=ai2ki(pcoeff); % reflection
    %======= tab_cod(k,3:NbParam)= coeff_refl; % coeffts
    tab_cod(k,1)=enrg;
    tab_cod(k,2)=0;
    tab_cod(k,3:NbParam)=[pcoeff(2:ordnv+1)'
    ... zeros(1,ordv-ordnv)];
end;
end;
Notice the presence of a high-pass type pre-emphasis filter preceding the operations for estimating the model’s parameters.

The prediction coefficients, obtained by analyzing the signal, are stored as “double floating point” numbers (8 bytes). If we had to use a “fixed-point” processor, it might be better to consider the reflection coefficients with values between $-1$ and $+1$. Speech coders also use what are called \textit{lsp} coefficients, short for \textit{Line Spectrum Pair} ([26]).

3. Decoding program:

```matlab
%===== decode.m
% Decoding the file tab_cod.mat
clear
TIC
load tab_cod; % tab_cod(nblocs,XX);
excg1=eye(1,40); % glottal signal
lbloc=240; % block length
recouv=60; % overlap
ltr=lbloc-recouv;
OvlRec=lbloc/3; % overlap reconstruction(1/3)
LBrec=lbloc+2*(OvlRec-recouv); % reconstructed block length
nblocs=size(tab_cod,1); NbParam=size(tab_cod,2);
outsig=[]; finalsig=zeros(1,nblocs*ltr+OvlRec);
%===== Reconstruction window
fen_rec=[(1:OvlRec)/OvlRec ones(1,lbloc-2*recouv) ... 
(0.25:(OvlRec:-1:1))/OvlRec];
ImpGlPtr=0;
LgExcGl=length(excg1);
NbSmpTot=LBrec+ LgExcGl; % because of the filtering
drap_vnv=0;
%===== for k=2:(nblocs-1),
if (tab_cod(k,2)==0) %===== voiced block
if (drap_vnv==1) % the previous one is voiced
  %===== continuity of the input signal
  trame=[TmpSig(ltr+1:NbSmpTot), zeros(1,ltr)];
  NbSmp=NbSmpTot-ltr+ImpGlPtr;
else
  % The previous one is not voiced
  trame=zeros(1,NbSmpTot); NbSmp= 0;
end
PitchPeriod=tab_cod(k,2); % block pitch
while (NbSmp<LBrec),
  trame((NbSmp+1):(NbSmp+LgExcGl))=excg1;
  NbSmp=NbSmp+PitchPeriod;
end
```
Hints and Solutions 219

```matlab
end
drap_vnv=1; ImpGlPtr=NbSmp-NbSmpTot;
TmpSig=trame; trame=trame(1:LBrec);
trame=trame/std(trame);  % normalization
else  %===== non voiced
    ImpGlPtr=0;
drap_vnv=0;  % gaussian
    trame=randn(1,LBrec);  % white noise
end;
trame=sqrt(tab_cod(k,1))*trame;  % power
%den=ki2ai(tab_cod(k,3:NbParam));  %<==========
den=[1 tab_cod(k,3:NbParam)];
outsig=filter(1,den,trame); outsig=fen_rec.*outsig;
st=(k-1)*ltr;
%==== construction with an overlap
finalsig((st+1):(st+LBrec))=...  
    finalsig((st+1):(st+LBrec)) + outsig;
end;
finalsig=filter(1,[1 -0.9375],finalsig);  % de-emphasis
TOC, soundsc(finalsig,8000);
```

The length of the block can be modified in the instruction `lbloc=240` of the `decode.m` program. By typing for example `lbloc=180`, the reconstructed windows are shorter and the same sentences are uttered faster. In both cases, this modification of the utterance speed occurs without a change in the timbre of the voice, which keeps its original, natural aspect. This is no longer the case if the sampling frequency is modified suddenly, with the same ratio by typing for example `soundsc(y,Fe*3/4)` to slow down the sentence. You can listen to the results and compare.

H5.2 (DTW) (see p. 119)
Type the following function:

```matlab
function [Dmin,DTWway,CD]=DTW1(xx,yy)
%!========================================================================!
%! Synopsis: [Dmin,DTWway,CD]=DTW1(xx,yy)  
%!        xx,yy = cepstrum of x and y signals  
%!        Dmin = minimal cumulative distance  
%!        DTWway = DTW way  
%!        CD = array of cumulative distances  
%!========================================================================!
mxx = 1; mxy = 2; myy = 1;
dd=size(xx,1); Ix=size(xx,2); Jy=size(yy,2);
distance=zeros(Ix,Jy);  
for ix=1:Ix
    for jy=1:Jy
        diffe=xx(:,ix)-yy(:,jy);
        distance(ix,jy)=sqrt(diffe'*diffe);  
%;========================================================================!
```
%===== cumulative distance
CD = zeros(Ix, Jy);
%===== parent to keep
Parent = zeros(Ix,Jy,2);
%===== CD initialization
CD(1,1) = distance(1,1);
Parent(1,1,:) = [1 1];
for ix = 2:Ix
    CD(ix,1) = distance(ix,1)+mxx*CD(ix-1,1);
    Parent(ix,1,:) = [ix-1 1];
end
for jy = 2:Jy
    CD(1,jy) = distance(1,jy)+myy*CD(1,jy-1);
    Parent(1,jy,:) = [1 jy-1];
end
%===== main loop
nT = min(Ix, Jy);
for tt = 2:nT
    for ix = tt:Ix
        DD = [CD(ix-1,tt)+mxx*distance(ix,tt), ...
              CD(ix-1,tt-1)+mxy*distance(ix,tt), ...
              CD(ix,tt-1)+myy*distance(ix,tt)];
        [val,ind] = min(DD);
        CD(ix,tt) = val;
        switch ind
            case 1
                Parent(ix,tt,:) = [ix-1,tt];
            case 2
                Parent(ix,tt,:) = [ix-1,tt-1];
            case 3
                Parent(ix,tt,:) = [ix,tt-1];
        end
    end
    for jy = tt+1:Jy
        DD = [CD(tt-1,jy)+mxx*distance(tt,jy), ...
              CD(tt-1,jy-1)+mxy*distance(tt,jy), ...
              CD(tt,jy-1)+myy*distance(tt,jy)];
        [val,ind] = min(DD);
        CD(tt,jy) = val;
        switch ind
            case 1
                Parent(tt,jy,:) = [tt-1,jy];
            case 2
                Parent(tt,jy,:) = [tt-1,jy-1];
            case 3
                Parent(tt,jy,:) = [tt,jy-1];
        end
    end
end
Hints and Solutions  221

%===== normalization
%===== minimal sum
Dmin = CD(Ix,Jy);
%===== backtracking inverse
bi=zeros(nT+1,2);
%===== we start at the end
bi(1,:)=[Ix Jy]; k=2;
while (Parent(bi(k-1,1),bi(k-1,2),1) ~= 1 ... & Parent(bi(k-1,1),bi(k-1,2),2) ~= 1)
    bi(k,:) = Parent(bi(k-1,1),bi(k-1,2),:);
    k = k + 1;
end
DTWway = bi(k-1:-1:1,:);

H5.3 (DTW word recognition)  (see page 120)

1. Type the following function:

```
function cepstre=extractCEPSTRE(xt,Fe)
%!=========================================!
%! Synopsis: cepstre=EXTRACTCEPSTRE(xt,Fe) !
%! xt = audio signal !
%! Fe = sampling frequency !
%! cepstre = cepstral coefficients !
%!=========================================!
xt=xt(:);
%===== parameters
pp=10;  % cepstrum order
duree=15;  % window duration in ms
Lfen=fix(Fe*duree/1000);  % window size
decal=fix(Lfen/2);  % shift for overlapping
Lfft=2^nextpow2(Lfen);  % FFT size
hamm=0.5-0.5*cos(2*pi*(0:Lfen-1)'/Lfen);
Lx=length(xt);
nbfen=fix(Lx/decal);
cepstre = zeros(pp,nbfen);
for k = 1:nbfen-1
    inddeb=(k-1)*decal+1;indfin=inddeb+Lfen-1;
    xaux=xt(inddeb:indfin).*hamm;
    %===== compute standard cepstral coefficients
    Sx = log(abs(fft(xaux,Lfft)));
    %===== power cepstrum
    Cx = real(ifft(Sx));
    cepstre(:,:,k)=Cx(2:pp+1);  % without energy
end
```
2. Type the following program:

```matlab
%===== DTWtry.m
clear all
[x,fe]=wavread('utter2.wav');
[y,fe]=wavread('utter4.wav');
cepx=extractCEPSTRE(x,fe);
cepy=extractCEPSTRE(y,fe);
[Dmin,wayDTW,CD]=DTW1(cepx,cepy);
dfigure(1); imagesc(CD);
set(gca,'ydir','normal')
hold on
plot(wayDTW(:,1),wayDTW(:,2),'k'); hold off
```

H5.4 (PSOLA) (see page 122)

Type the program:

```matlab
%===== PSOLAtry.M
clear all;
[x,Fs]=wavread('desgens.wav');
gamma=0.8;
x_m=psola(x,Fs,gamma);
soundsc(x_m,Fs);
```

which calls the following function:

```matlab
function s_synt=psola(s_orig,Fs,gamma)
%!=========================================!
%! SYNOPSIS: s_synt=PSOLA(s_orig,Fs,gamma) !
%! s_orig = signal !
%! Fs = sampling Frequency (Hz) !
%! gamma = modification rate !
%! s_synt = modified Signal !
%! Uses the F0cor function !
%!=========================================!
seuil_pitch=0.7;
Rsurech=1;  % for improving the pitch's evaluation
L10ms=fix(Fs/100);  % constant size window (10 ms)
fp_min=70; fp_max=400; Lfen=2*fix(Fs/fp_min);
N=
length(s_orig); Namax=fix(Ns*fp_max/Fs);
ta=zeros(Namax,1); ta(1)=1;
Pa=L10ms; inda=1;
%!===== analysis
while ta(inda)<Ns-Lfen
    indsd=ta(inda);
    % The length Lfen must be large enough
    % to allow the estimation of the lowest frequency
    indsfin=indsde+Lfen; sextrait=s_orig(indsde:indsfin);
```
[Fpitch, corr]=...
    f0cor(sextrait,Fs,Rsurech,seuil_pitch,fp_min,fp_max);
if isnan(Fpitch)
    Pa=L10ms;
else
    Pa=fix(Fs/Fpitch);
end;
inda=inda+1; ta(inda)=ta(inda-1)+Pa;
end

%===== time scale modification and Synthesis

s_synt=zeros(fix(Ns/gamma),1);
ii=1;ts=1;ie=1;te=1;
while ie<Na-2
    ii=ii+1;
    te=te+gamma; ie=ceil(te);
    Pa=ta(ie+1)-ta(ie); ts=ts+Pa;
    winHann=sin(pi*(0:2*Pa)'/(2*Pa)) .^2;
    sola=s_orig(ta(ie):ta(ie)+2*Pa) .* winHann;
    s_synt(ts-Pa:ts+Pa)= s_synt(ts-Pa:ts+Pa)+sola;
end

H5.5 (Hann window) (see page 123)

Type the following program:

```matlab
%% fenHann.m
L=300; alpha=1/6;
n0=fix(alpha*L);

%%% Hann window
hn=abs(sin(pi*(0:L-1)'/L)) .^2;
gn=hn.^2; pp=500; x=zeros(pp*n0,1);
for k=0:pp-ceil(L/n0)
    id1=k*n0;
    x(id1+1:id1+L)=x(id1+1:id1+L)+gn;
end
figure(1);plot(x)
max(x), sum(gn)/n0

%%% plotting the DTFT of gn
figure(2); Lfft=4*1024; Gf=abs(fft(gn,Lfft));
plot((0:Lfft-1)/Lfft,20*log10(Gf))
set(gca,'xlim',[0 0.2]); hold on;
plot(ones(2,1)*(1:5)/n0,[-140*ones(1,5);40*ones(1,5)],':');
hold off
```

As you can see on the resulting graph, the sequence \( x(n) \) is constant for any value \( L \). The constant is equal to the sum of the elements of the sequence \( g(n) \) divided by \( n_0 \). Therefore it depends on \( \alpha \). Notice that the property is still true for other powers of \( h(n) \). By using the Poisson formula, we can show that this property is equivalent to the fact that the sequence \( g(n) \) is equal to zero.
for the multiples of $1/n_0$. This amounts to choosing the inverse of an integer as the value of $\alpha$ and to choosing $L$ so as to have $L\alpha$ equal to an integer.

**H5.6 (Phase vocoder)** (see page 124)

Type the program:

```
%% H5.6 (Phase vocoder) (see page 124)
Type the program:  

function s_synt=phasevoc(s_orig, gamma, Lfft)  
%!======================================================================!  
%! SYNOPSIS: s_synt=PHASEVOC(s_orig,gamma,Lfft)  
%! s_orig = audio source  
%! gamma = modification rate  
%! Lfft = FFT length  
%! s_synt = modified audio signal  
%! Uses stft.m, specinterp.m and spec2sig.m  
%!======================================================================!  
% alpha is the shift rate relative to the FFT length  
unsuralpha=8;    % power of 2  
n0=fix(Lfft/unsuralpha);  
win=hann(Lfft);  
%synt= initial STFT  
spec_a = stft(s_orig, Lfft, n0, win);  
%synt= calculus of the modified DTFT  
spec_s = specinterp(spec_a, gamma);  
%synt= inversion  
s_synt = spec2sig(spec_s, n0, win);  
```

```
function sig=spec2sig(spec, n0, win)  
%!======================================================================!  
%! Synthesis of a signal from its spectrogram !  
%! SYNOPSIS: sig=SPEC2SIG(spec,ovlap,win)  
%! spec = spectrogram  
%! n0 = shift value  
%! win = weighting window  
%! sig = signal  
%!======================================================================!  
[Lfft,nbcol] = size(spec);  
ispec=real(ifft(spec));  
```
\texttt{sig = zeros(Lfft+(nbcol-1)*n0,1);}
%==== re-synthesis using a window
for icol = 1:nbcol
    sigfen = ispec(:,icol) .* win;
%==== overlap-Add
    ix = (icol-1)*n0+1;
    sig(ix:ix+Lfft-1) = sig(ix:ix+Lfft-1) + sigfen;
end

\texttt{function spec_s=specinterp(spec_a, gamma)}
%!===================================================!
%! Interpolation of a Short Term FT array
%! SYNOPSIS: spec_s=SPECINTERP(spec_a, gamma)
%! spec_a = original spectrogram (Short Term FT)
%! gamma = temporal modification rate
%! spec_s = modified spectrogram
%!===================================================!
\[Lfft,nbcol\] = size(spec_a);
\texttt{ts=1:gamma:nbcol-1;}
\texttt{spec_s = zeros(Lfft,length(ts));}
%==== phase and phase increase
\texttt{phase_a = angle(spec_a);}
\texttt{module_a = abs(spec_a);}
\texttt{diffp = zeros(Lfft,1);}
\texttt{phase_s=phase_a(:,1);}
\texttt{indcol = 1;}
\texttt{for tt = ts}
    \texttt{%==== two adjacent columns}
    \texttt{ta_min=floor(tt); ta_max=floor(tt)+1;}
    \texttt{%==== weighted Mean}
    \texttt{pond = tt - floor(tt);}
    \texttt{modul = (1-pond)*module_a(:,ta_min) +...}
    \texttt{pond *module_a(:,ta_max);}
    \texttt{spec_s(:,indcol) = modul .* exp(j*phase_s);}
    \texttt{%==== phase diff and accumulation}
    \texttt{diffp = phase_a(:,ta_max)-phase_a(:,ta_min);}
    \texttt{phase_s = phase_s + diffp;}
    \texttt{indcol = indcol+1;}
end

\texttt{function xspec = stft(x,Lfft,ns,win)}
%!======================================!
%! SYNOPSIS: xspec = STFT(x,Lfft,ns,win)!
%! x : signal (N*1)
%! Lfft : FFT length
%! ns : shift
%! win : weighting window (L*1)
%! xspec : spectrogram !%
%!======================================!
N=length(x); L=length(win);
nbb=fix((N-L)/ns);
xsvec=zeros(Lfft,nbb);
for k=1:nbb
    idb=k*ns+1; idf=idb+L-1;
xw=x(idb:idf) .* win;
    xsvec(:,k)=fft(xw,Lfft);
end

H5.7 (Detecting impulse clicks) (see page 126)

1. We have:

\[ \rho = \frac{|z_d(n)|^2}{E(|z_b(n)|^2)} = \frac{\left| \sum_{u=-\infty}^{+\infty} g(u)d(n-u) \right|^2}{\sigma^2 \int_{-1/2}^{1/2} |G(f)|^2 df} = \frac{1}{\sigma^2} \left( \frac{\sum_{u=-\infty}^{+\infty} g(u)d(n-u)}{\sum_{u=-\infty}^{+\infty} g^2(u)} \right)^2 \]

\[ d(n) + y(n) \xrightarrow{\otimes} y(n) \xrightarrow{g(n)} z(n) = z_d(n) + z_b(n) \]

**Figure H5.2 – Matched filter**

2. If we apply the Schwarz inequality to the numerator, we get:

\[ \left| \sum_{u=-\infty}^{+\infty} g(u)d(n-u) \right|^2 \leq \sum_{u=-\infty}^{+\infty} g^2(u) \sum_{u=-\infty}^{+\infty} d^2(u) \]

and therefore \( \rho \leq E_d/\sigma^2 \) where \( E_d = \sum_{u=-\infty}^{+\infty} d^2(u) \). The resulting upper bound is reached if we assume \( g(u) = d(n-u) \). It is therefore the maximum with respect to \( g(u) \). Note that the optimal solution is the reversed copy of the signal \( g(n) \). In the case where \( d(u) \) has a finite duration \( k \), we will assume \( g(u) = d(k-u) \) in order for the filter \( g(n) \) to be causal.
3. The filter with the transfer function \( A(z) = 1 + a_1 z^{-1} + \cdots + a_K z^{-K} \) is a linear filter with the finite impulse response \( \{h_1(n)\} = \{h_1(0) = 1, h_1(1) = a_1, \ldots, h_1(K) = a_K\} \). If we feed the signal \( x(n) = \delta(n) + s(n) \) into this filter’s input, we get the signal \( y(n) = h_1(n) + w(n) \), which is the sum of the deterministic signal \( h_1(n) \) and a white noise.

If we apply the result of the previous question, the conclusion is that we have to filter the signal \( y(n) \) by the filter with the impulse response \( h_1(-n) \). Aside from a \( K \) sample delay, we get the causal filter with the impulse response \( \{h_2(n)\} = \{h_2(0) = a_K, h_2(1) = a_{K-1}, \ldots, h_2(K) = 1\} \).

4. In the absence of clicks, the input signal \( y(n) \) of the matched filter is a white noise with the variance \( \sigma^2 \). Hence the output signal is centered and the output spectral density has the expression \( S(e^{2j\pi f}) = \sigma^2 |A(e^{2j\pi f})|^2 \). The output power is obtained by integrating the spectral density. Using the Parseval formula, we get:

\[
P_z = \sigma^2(1 + a_1^2 + \cdots + a_K^2) \tag{7.8}
\]

5. In the absence of clicks, the output signal \( z(n) \) of the matched filter \( h_2(n) \) is a centered, Gaussian noise with the variance \( P_z \). The probability of deciding the presence of a click is therefore given by:

\[
\Pr(|z(n)| > s|H_0) = \frac{1}{\sqrt{2\pi P_z}} \int_{s}^{+\infty} \exp\left(-u^2/2P_z\right)du \\
= \frac{1}{\sqrt{2\pi P_z}} \int_{s/\sqrt{P_z}}^{+\infty} \exp\left(-v^2/2\right)dv \\
= 2Q\left(s/\sqrt{P_z}\right)
\]

where \( Q(c) \) is the integral function of the centered, Gaussian distribution with the variance 1. If we choose \( Q(c) = 0.005 \), we have \( c \approx 3 \) and therefore:

\[
s = 3\sqrt{P_z}
\]

This threshold guarantees that the probability of deciding in favor of the presence of a click, when there is no click, is less than 1%: this is called the probability of false alarm. In order to set this level to satisfy a sound criterion, we have to compare the matched filter’s output with a threshold of the type \( \lambda\sqrt{P_z} \). The choice of \( \lambda \) will then be done by listening to the denoised signal. \( P_z \) can be estimated using expression (7.8).
The following program generates the useful signal comprising 500 samples of an AR-10. The impulses used to simulate clicks have an amplitude equal to 1.5 times the square deviation of the signal. The program then estimates the parameters, and computes the residual and the matched filter. Finally, the resulting signal is compared to a threshold (Figure H5.3):

```matlab
%===== Ccraq.m
clear
%===== original signal (order 10-AR)
a=[1 -1.6507 0.6711 -0.1807 0.6130 -0.6085 0.3977 ...
 -0.6122 0.5412 0.1321 -0.2393];
K=length(a); N=500; w=randn(1,N);
s=filter(1,a,w);
srms=sqrt(s*s'/N);
%===== NBCRAC clicks with an amplitude +/-1.5 srms
nbcrac=5; poscrac=[73 193 249 293 422];
ampcrac=1.5*srms*(2*round(rand(1,nbcrac))-1);
sig=s; sig(poscrac)=s(poscrac)+ampcrac;
subplot(311); plot(s); grid
subplot(312); plot(sig); grid
%===== detection of the clicks
[aest sw2est]=xtoa(sig,K); % estimation of the AR
y=filter(aest,1,sig); % whitening: estim. of residual
z=filter(aest(K:-1:1),1,y); % matched filtering
subplot(313); plot(z); grid
V0eff=sqrt(sw2est*aest'*aest);
lambda=3; threshold=lambda*V0eff;
izthreshold=find(abs(z)>threshold); % threshold
izthreshold=izthreshold-K; % filter delay
lzs=length(izthreshold);
%===== extraction of the maxima (3 samples from each other)
dist=izthreshold - [0 izthreshold(1:lzs-1)];
mpl3=find(dist>3); lm3=length(mpl3); mp13=[mpl3 1 lzsi+1];
for ii=1:lm3
t1=izthreshold(mp13(ii)); t2=izthreshold(mp13(ii+1)-1);
[zmax(ii) im]=max(z(t1:t2));
posEstim(ii)=im+t1;
end
izthreshold,poscrac,posEstim
```

Note that the instruction `izthreshold=izthreshold-K`, which substracts $K$ from the detected positions, takes into account the $K$ sample delay caused by the causal implementation of the matched filter $h_2(n)$. 
**H5.8 (Restoring “missing values”)** (see page 129)

1. Let $K$ be the order of the AR model and $N$ the sample size. We wish to minimize the square deviation between the sequence of values $x(n)$ and the sequence of predicted values $\hat{x}(n) = -a_1 x(n-1) - \cdots - a_K x(n-K)$, for $n$ from 1 to $N$, that is to say the quantity:

$$\sum_{u=1}^{N-K} (x(u) + a_1 x(u-1) + \cdots + a_K x(u-K))^2$$

The minimization is done with respect to $m$ unknown values, with indices from $\ell$ to $\ell + m - 1$ (Figure H5.4).

These values are only involved in a limited number of terms of this sum which are:

$$J = \sum_{u=\ell}^{\ell+m+K-1} (x(u) + a_1 x(u-1) + \cdots + a_K x(n-K))^2$$

Notice that $J$ appears as the norm of the vector:

$$e = T x$$
Figure H5.4 – Several values are restored around the detected position

where $T$ is an $(m + K) \times (m + 2K)$ Toeplitz matrix constructed from the coefficients $(a_1, \ldots, a_K)$:

$$
T = \begin{bmatrix}
a_K & a_{K-1} & \cdots & a_1 & 1 & 0 & \cdots & \cdots \\
0 & a_K & a_{K-1} & \cdots & a_1 & 1 & 0 & \cdots \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
\end{bmatrix}
$$

and $x$ is a size $(m + 2K)$ vector defined by:

$$
x = \begin{bmatrix}
x(\ell - K), \ldots, x(\ell - 1), x(\ell), \ldots, x(\ell + m - 1), \\
x_0 : \text{known} & y : \text{unknown} \\
x(\ell + m), \ldots, x(\ell + m + K - 1)\\
x_1 : \text{known}
\end{bmatrix}^T
$$

The size $k$ vectors $x_0$ and $x_1$ are comprised of values that are known. The size $m$ vector $y$ is comprised of values we have to reconstruct.

Hence we can partition $T$ in three matrices of the adequate size such that $e = A_0 x_0 + A_1 x_1 + By$, which we can also write $x = A_0 x_0 + A_1 x_1 = -By + e$. In the end:

$$
x = -By + e
$$

2. Minimizing the norm of $e$ with respect to $y$ involves the usual formulation of a least squares problem. The solution is:

$$
\hat{y} = -(B^T B)^{-1} B^T x = -(B^T B)^{-1} B^T (A_0 x_0 + A_1 x_1)
$$
H6 Selected topics

H6.1 (Contour ellipse: the least squares method) (see page 143)

1. Type:

```matlab
%===== detell.m
% run preprocesscoin before
function detell()
load yim02 %===== from preprocesscoin
yim=yim02(5:145,8:295);
[ligyim, colyim] = size(yim);
imagesc(yim), colormap('gray')
dd=abs(diff(yim)); vv=find(dd>00);
   yy1=rem(vv,(ligyim-1)); xx1=fix((vv-yy1)/(ligyim-1));
%===== undersampling
   xx1=xx1(1:15:end); yy1=yy1(1:15:end);
N=length(xx1); hold on, plot(xx1,yy1,'ob')
   xy=[xx1';yy1']; % observations
   GOBS=[xy(1,:).^2;xy(2,:).^2;xy(1,:).*xy(2,:);...
          xy(1,:);xy(2,:)];
%===== coefficients
   alphak= - GOBS' \ ones(N,1);
draw_ellipse(alphak,N); hold off

%===== drawing an ellipse
function X=draw_ellipse(alpha,N)
M=zeros(2,2);
M(1,1)=alpha(1); M(2,2)=alpha(2);
M(1,2)=alpha(3)/2; M(2,1)=alpha(3)/2;
   center=-inv(M)*[alpha(4);alpha(5)]/2;
   rho2=center'*M*center-1;
%===== theta = (0:N-1) * (2*pi) ./ (N-1) ;
   Y = sqrt(rho2)*[cos(theta);sin(theta)];
   X = diag(center)*ones(2,N)+sqrtm(M)\Y;
   hpl=plot(X(1,:),X(2,:),'y'); set(hpl,'linewidth',.5)
```

2. Theoretically the points of the ellipse obey the equation
\[
ax_1^2 + bx_2^2 + cx_1x_2 + dx_1 + ex_2 - 1 = 0.
\]
Hence the idea of estimating the coefficients based on \(N\) pairs \(\{x_1(n), x_2(n)\}\) by determining the value of
\[
\theta = [a \ b \ c \ d \ e]^T
\]
that minimizes:
\[
(X\theta - u)^T (X\theta - u)
\]
where \(X\) is the \(N \times 5\) matrix constructed from the sequences \(x_1(n)\) and \(x_2(n)\), and where \(u\) refers to the length \(N\) vector containing nothing but
the components 1. The solution is given by:

$$\theta = X\#u$$

Once $\theta$ has been estimated, we draw the ellipse using the equation $(x - x_0)^T E (x - x_0) - \gamma = 0$. In order to do this, we have to determine the expressions which lead from $\theta$ to the parameters $x_0$, $E$ and $\gamma$. By expanding $(x - x_0)^T E (x - x_0) - \gamma = 0$, we get $e_{11}x_1^2 + e_{22}x_2^2 + 2e_{12}x_1x_2 - 2x^T E x_0 + x_0^T E x_0 - \gamma = 0$ ($e_{ij}$ refers to the generating element of $E$ where $E = E^T$). If we identify this expansion as

$$ax_1^2 + bx_2^2 + cx_1x_2 + dx_1 + ex_2 - 1 = 0,$$

we first have $e_{11} = a$, $e_{22} = b$ and $e_{12} = e_{21} = c/2$. Then, for any pair $x = [x_1 \ x_2]^T$, we have

$$-2x^T E x_0 = dx_1 + ex_2 = x^T [d \ e]^T,$$

meaning that:

$$x_0 = -\frac{1}{2} E^{-1} \begin{bmatrix} d \\ e \end{bmatrix}$$

Finally, we have $x_0^T E x_0 - \gamma = -1$.

H6.2 (Contour ellipse: the covariance method) (see page 145)

1. Let $U_1$ and $V_1$ be the two components of $y_1$. The mean vector $\mathbb{E}\{y_1\}$ has two components. The first one is:

$$\mathbb{E}\{U_1\} = \int_{\mathbb{R}^2} u_1 p_Y(u_1, u_2) du_1 du_2 = \frac{1}{\pi} \int_{\mathcal{C}} u_1 du_1 du_2$$

If we assume $u_1 = \rho \cos(\theta)$, we get:

$$\mathbb{E}\{U_1\} = \frac{1}{\pi} \int_0^{2\pi} \int_0^1 \rho \cos(\theta) \rho d\rho d\theta = 0$$

We also have to check that the second component $\mathbb{E}\{V_1\} = 0$. In the end, $\mathbb{E}\{y_1\} = 0$.

The covariance matrix $\mathbb{E}\{y_1 y_1^T\}$ requires the calculation of three quantities $\mathbb{E}\{U_1^2\}$, $\mathbb{E}\{V_1^2\}$ and $\mathbb{E}\{U_1 V_1\}$. If we assume $u_1 = \rho \cos(\theta)$, we get:

$$\mathbb{E}\{U_1^2\} = \int_{\mathbb{R}^2} u_1^2 p_Y(u_1, u_2) du_1 du_2 = \frac{1}{\pi} \int_0^{2\pi} \int_0^1 \rho^2 \cos^2(\theta) \rho d\rho d\theta = \frac{1}{4}$$

Likewise, we have $\mathbb{E}\{V_1^2\} = 1/4$ et $\mathbb{E}\{U_1 V_1\} = 0$. Therefore:

$$\mathbb{E}\{y_1 y_1^T\} = \frac{1}{4} I_2$$
2. The *law of large numbers* states that when $N$ tends to infinity:

$$\frac{1}{N} \sum_{n=1}^{N} (y_n - \hat{\nu}_N)(y_n - \hat{\nu}_N)^T \xrightarrow{a.s.} \frac{1}{4} I_2$$

If we multiply on the left and on the right by $M^{1/2}$ and according to (6.6) we infer that:

$$\frac{1}{N} \sum_{n=1}^{N} (x_n - \hat{\mu}_N)(x_n - \hat{\mu}_N)^T \xrightarrow{a.s.} M/4$$

where $\hat{\mu}_N = N^{-1} \sum_{n=1}^{N} x_n$.

3. Type:

```matlab
clear all, close all
load mcoin2
[nbline nbcol]=size(pixc); [yy1,xx1]=find(pixc<70);
%===== points=[xx1 yy1]; NN=length(xx1);
mymean=mean(points);
points_centres=points-ones(NN,1)*mymean;
RR=points_centres'*points_centres/NN;
image(pixc); colormap('gray'), hold on
MM=4*RR; E=inv(MM); ellipse(mymean,E,1)
hold off
```

**H6.3 (Optimization of a portfolio with two assets)** (see page 156)

Preliminary remark: due to the fact that there are only two assets and that $x_1 + x_2 = 1$, we can replace $x_2$ with $1 - x_1$ in $J$ and plot $J$ as a function of $x_1 \in (0, 1)$. However, for pedagogical reasons, we propose to solve the problem by using the KKT conditions. The Lagrangian is written as:

$$\mathcal{L}(x, \lambda, \mu) = \frac{1}{2} x^T Q x + \lambda^T (x_1 + x_2 - 1) + \mu_1 (R_e - m_1 x_1 - m_2 x_2) - \mu_2 x_1 - \mu_3 x_2$$

1. To solve the problem, we need to look at the different cases as a function of the conditions of complementarity:

   (a) If $x_1 > 0, x_2 > 0$ and $x^T m > R_e$, then $\mu_1 = \mu_2 = \mu_3 = 0$. The Lagrangian is written:

   $$\mathcal{L} = \frac{1}{2} x^T Q x + \lambda (1 - x^T u)$$
By canceling the derivative with respect to $x$, we find
\[ \partial_x L = Qx - \lambda u = 0 \]

This gives us
\[ x = \lambda Q^{-1} u \]

If we write that $x^T u = 1$, we find
\[ x_o = \frac{1}{u^T Q^{-1} u} Q^{-1} u \]
on condition that both components are positive and that
\[ m^T x_o = \frac{1}{u^T Q^{-1} u} m^T Q^{-1} u > R_e \]
i.e. on condition that $R_e < B/A$. In this case, the risk is given by $1/u^T Q^{-1} u$ and is therefore constant.

(b) $x_1 > 0$, $x_2 > 0$ and $x^T m = R_e$ in this case $\mu_2 = \mu_3 = 0$. The Lagrangian is written
\[ L = \frac{1}{2} x^T Q x + \lambda (1 - x^T u) + \mu_1 (R_e - x^T m) \]

By canceling the derivative with respect to $x$, we find
\[ \partial_x L = Qx - \lambda u - \mu_1 m = 0 \]

This gives
\[ x = \lambda Q^{-1} u + \mu_1 Q^{-1} m \]

By writing that $x^T u = 1$ and that $x^T m = R_e$, we obtain the two equations with two unknowns:
\[ \begin{cases} \lambda u^T Q^{-1} u + \mu_1 u^T Q^{-1} m = 1 \\ \lambda m^T Q^{-1} u + \mu_1 m^T Q^{-1} m = R_e \end{cases} \iff \begin{cases} A\lambda + B\mu_1 = 1 \\ B\lambda + C\mu_1 = R_e \end{cases} \]

From this, we deduce that
\[ \begin{align*} \lambda &= \frac{C - BR_e}{AC - B^2} \\ \mu_1 &= \frac{AR_e - B}{AC - B^2} \end{align*} \quad (7.9) \]

By substituting $\lambda$ and $\mu_1$ into $x$, we obtain the solution on condition that $\mu_1 \geq 0$ - i.e. $R_e \geq B/A$ - and that $x_1$ and $x_2$ are positive.

(c) If $x_1 = 0$, then $x_2 = 1$: the risk is equal to $\sqrt{q_2}$ and the return is $x^T m = m_2$.

Similarly, if $x_2 = 0$, then $x_1 = 1$: the risk is equal to $\sqrt{q_1}$ and the return is $x^T m = m_1$.

The two solutions must be compared to the values found previously as a function of the numerical values of $Q$, $m$ and $R_e$. 
Figure H6.1 represents the curve of optimal portfolios. The linear part of the curves corresponds to the case $R_e < B/A$ where the constraint $R_e - x^Tm_a < 0$ – i.e. non-active. Its abscissa value is given by $1/\sqrt{u^TQ^{-1}u}$, which is the minimum possible risk according to example 6.8. The circle indicates the portfolio $(0,1)$ and the triangle the portfolio $(1,0)$. That portfolio is not optimal because it is strictly below the curve.

Figure H6.1 – Market frontier for a market with 2 assets

2. Type:

```matlab
%===== twoassets.m
clear all
q1 = 1;
q2 = 2;
rho = 0.3;
Q = [q1 rho*sqrt(q1*q2);rho*sqrt(q1*q2) q2];
m = [0.1;0.2];
u = ones(2,1);
A = u'*(Q\u);
B = m'*(Q\u);
C = m'*(Q\m);
Rebound = B/A;
M = [A B;B C];
nbvalRe = 20;
listRe = linspace(0,max(m),nbvalRe);
sigmao = zeros(nbvalRe,1);
xo = zeros(2,nbvalRe);
for ir = 1:nbvalRe
    Re = listRe(ir);
    if Re < Rebound
        xo(:,ir) = (Q\u) /A;
    
```
\begin{align*}
\text{sigmo}(\text{ir}) & = \sqrt{\text{xo}(\cdot,\text{ir})' \times \text{Q} \times \text{xo}(\cdot,\text{ir})}; \\
\text{else} & \\
\gamma & = \text{M}[1;\text{Re}] ; \\
\lambda & = \gamma(1) ; \\
\mu & = \gamma(2) ; \\
\text{xo}(\cdot,\text{ir}) & = \lambda (\text{Q} \cdot \text{u}) + \mu (\text{Q} \cdot \text{m}) ; \\
\text{sigmo}(\text{ir}) & = \sqrt{\text{xo}(\cdot,\text{ir})' \times \text{Q} \times \text{xo}(\cdot,\text{ir})}; \\
\text{end} & \\
\text{end} & \\
\text{x} & = [0;1]; \\
\text{Re} & = \text{m}(2); \\
\text{sigmo} & = \sqrt{\text{xo0} ' \times \text{Q} \times \text{xo0}} ; \\
\text{xo} & = [1;0]; \\
\text{Re} & = \text{m}(1); \\
\text{sigmo} & = \sqrt{\text{xo1} ' \times \text{Q} \times \text{xo1}} ; \\
\text{plot} & (\text{sigmo}, \text{listRe},',-'); \\
\text{hold on} & \\
\text{plot} & (\text{sigmo0}, \text{Re0},'or', 'markerf', 'r', 'markers', 10) \\
\text{plot} & (\text{sigmo1}, \text{Re1},'vg', 'markerf', 'g', 'markers', 10) \\
\text{hold off} & \\
\text{grid on} & \\
\end{align*}

**H6.4 (Negative correlation)** (see page 157) Type:

\begin{verbatim}
#========== negative correlation
clear all
q1 = 1;
q2 = 2;
nbrho = 10;
listrho = linspace(-0.7,0.3,nbrho);
m = [0.05;0.3];
u = ones(2,1);
nbvalRe = 10;
listRe = linspace(min(m),max(m),nbvalRe);
sigmo = zeros(nbvalRe,1);
xo = zeros(2,nbvalRe);
for irho = 1:nbrho
  rho = listrho(irho);
  Q = [q1 rho*sqrt(q1*q2);rho*sqrt(q1*q2) q2];
  A = u'*(Q\u);
  B = m'*(Q\u);
  C = m'*(Q\m);
  Rebound = B/A;
  M = [A B;B C];
  for ir = 1:nbvalRe
    Re = listRe(ir);
    if Re < Rebound
      xo(:,ir) = (Q\u) / A;
      sigmo(ir) = sqrt(xo(:,ir)'*Q*xo(:,ir));
    end
  end
end
\end{verbatim}
else
    gamma = M[1;Re];
    lambda = gamma(1);
    mu = gamma(2);
    xo(:,ir) = lambda * (Q\u) + mu * (Q\m);
    sigmao(ir) = sqrt(xo(:,ir)'*Q*xo(:,ir));
end
end

plot(sigmao,listRe,'.-')
hold on
end
hold off
grid on

We have shown, in Figure H6.2, the Market boundaries for different values of \( \rho \). We observe that, for a given level of risk (on the abscissa axis), the more negative the correlation is, the better the return becomes.

The linear part of the curves corresponds to the case \( R_e < B/A \) where the constraint \( R_e - x^Tm_a < 0 \) – i.e. non-active. Its abscissa value is given by \( 1/\sqrt{u^TQ^{-1}u} \), which is the minimum according to example 6.8.

![Figure H6.2 - Market boundaries for a market with 2 assets and for different values of the correlation. From left to right, the correlation runs from -0.7 to 0.3. We can see that for a given level of risk (on the abscissa axis), the more the correlation \( \rho \) approaches -1, the higher the return.](image)

**H6.5 (Optimization of a portfolio) (see page 162)**

Type the program:

```matlab
%==== optimFolio
clear all
Q = 1e-2*[...  
   4 2 -1 -3 0 2 ; 2 7 0 1 4 -1;...
   -1 0 20 7 -1 13 ; -3 1 7 10 -2 4;...
   0 4 -1 -2 11 -7 ; 2 -1 13 4 -7 18];
ma = [0.03;0.04;0.08;-0.02;0.06;0.08];
U = ones(6,1);
```
alpha = U'*Q\U;
beta = U'*Q\ma;
[bmax, imax] = max(ma);
Rf = 0.02;
xi = zeros(6,1);
xi(imax) = 1;
J = @(x,Q) x'*Q*x;
listRe = linspace(Rf,max(ma),20);
nbRe = length(listRe);
s2 = zeros(nbRe,1);
s2za = zeros(nbRe,1);
xa = zeros(6,nbRe);
za = zeros(6,nbRe);
xf = zeros(nbRe,1);
Rp = zeros(nbRe,1);
A = [U';(Rf*U-ma)'

for ire = 1:nbRe
    Re = listRe(ire);
    B = [1;(Rf-Re)];
    [xa(:,ire), fval] = ...  
                    fmincon(@(x) J(x,Q), xi,A,B,[],[],zeros(6,1));
    xf(ire) = 1-sum(xa(:,ire));
    Rp(ire) = ma'*xa(:,ire)+xf(ire)*Rf;
    s2(ire) = xa(:,ire)'*Q*xa(:,ire);
    za(:,ire) = xa(:,ire)/sum(xa(:,ire));
    s2za(ire) = za(:,ire)'*Q*za(:,ire);
end

figure(1), plot(sqrt(s2),Rp,'.-b')
hold on, plot(sqrt(s2za),Rp,'.-r'), hold off

H6.6 (Face recognition) (see page 174)
First, make a catalog of the data corresponding to the photographs in levels of gray. In our example, the catalog called orlfaces contains the catalogs called s1, s2, ..., each one containing the 10 photographs that have to be processed.

Type the training program LDAPCAtraining.m. It is used by the test program LDAPCAtest.m:

```matlab
XredPCA2D = cell(1,nbindiv);
XredLDA = cell(1,nbindiv);
V = cell(nbindiv,1);
W = cell(nbindiv,1);
for ii=1:nbindiv
    grandX = zeros(d,nbimages_L);
    grandXcell = cell(nbimages_L,1);
    for kimg=1:nbimages_L
        pkimg = permutind(kimg);
        filename = sprintf('%s/s%i/%i.png',ImageFile,ii,pkimg);
image = imread(filename);
grandXcell{kim} = image;
end
[V{ii}, W{ii}] = PCA2D(grandXcell{kim}, k1, k2);
XredPCA2D{ii} = zeros(k1*k2, nbimages_L);
for kimg=1:nbimages_L
    VTGW = V{ii}'*double(grandXcell{kim})*W{ii};
    XredPCA2D{ii}(:, kimg) = reshape(VTGW, k1*k2, 1);
end
gLDA = LDA(XredPCA2D, dim_barycenter);
barycenter_nua = zeros(dim_barycenter, nb indiv);
for ii=1:nb indiv
    XredLDA{ii} = gLDA\XredPCA2D{ii};
barycenter_nua(:, ii) = XredLDA{ii}*ones(nbimages_L, 1)/... nbimages_L;
end

%==== LDAPCAtest.m
ImageFile = '../orlfaces';
nb indiv = 40; % number of individuals
nb images = 10; % number of photos
nb images_L = 4; % number of photos for training
d = 112*92; % photo size
k1 = 7;
k2 = 7;
%==== dimension of the LDA less than k1*k2
dim_barycenter = 20; cp = 0; Lruns = 10;
for irun = 1:Lruns
%==== training
% random permutation over the NBIMAGES images
% to use different training databases.
% The remaining images will be used for testing
%====
permutind = randperm(nbimages);
LDAPCAtraining
%==== testing
matriceconf = zeros(nb indiv);
aux = zeros(nb indiv, 1);
for ii = 1:nb indiv
    for jj = nbimages_L+1:nbimages
        pjj = permutind(jj);
        filename = sprintf(['../orlfaces/s%i/%i.png', ii, pjj]);
        grandXcell_T = double(imread(filename));
        AA = double(V{ii})*grandXcell_T*double(W{ii});
        Xred_T = reshape(AA, k1*k2, 1);
    end
end
XredLDA_T = gLDA \backslash Xred_T;
for kk=1:nbindiv
    aux(kk) = norm(barycenter_nua(:,kk) - XredLDA_T,'fro');
end
[minaux,indaux] = min(aux);
matriceconf(ii,indaux) = matriceconf(ii,indaux)+1;
end

end

% idxsr = sprintf('%i, ',sort(permutind(1:nbimages_L)));
% txtL = sprintf('Learning on images %s',idxsr);
% figure(1)
% imagesc(matriceconf)
% title(txtL,'fontsize',16)
if any(diag(matriceconf)-=nbimages-nbimages_L),
    disp('alarm');cp=cp+1;
end
end
cp/Lruns

H6.7 (GPS location) (see page 178)

Type the program:

```matlab
%===== exeGPS.m
clear all
toler = 0.1;
c_mps = 299792458;
true_loc_m = [248645.5722;-4828261.0758;4146460.5047];
sat_loc_m = 1.0e+07 * ...
[-1.731978829800000 -1.875361540500000 0.728987051300000;...
 -1.362041209000000 -0.813947623800000 2.149113287300000;...
 0.809023521200000 -2.399402432700000 -0.752593941400000;...
 1.929942650600000 -0.960140862800000 1.585822443500000;...
 2.345122552599999 -0.672073953600000 1.090961123000000;...
 1.165884015800000 -1.341861781800000 1.940808680800000];
pseudorange_s = [ ...
 0.075522044080650;...
 0.074908679917403;...
 0.079304017358102;...
 0.076286737020851;...
 0.080875081503129;...
 0.069734492372016];
L_m = c_mps*pseudorange_s;
%===== initialization
init_loc = zeros(3,1);
[estim_loc_m, ttestime, delta] = ...
    GPSloc(L_m, init_loc, sat_loc_m, toler);
disterr = norm(true_loc_m-estim_loc_m);
format long, disterr, format
```

and the function:
function [receiver_loc, tt, delta] = ...
    GPSloc(distance_m, init_loc_m, sat_loc_m, toler)
%!========================================================================!
%! distance_m : pseudo-range (s) * c (m/s) !!!
%! init_loc_m : initial location algorithm !!!
%! sat_loc_m : satellite location (m) (I by 3) !!!
%! toler : arret algorithme (typ. 0.1) !!!
%!========================================================================!
    tt = 0; % init value of the time shift tau
    receiver_loc = init_loc_m;
    c_mps = 299792458;
   I = length(distance_m); % nb_sat
    max_delta = 1; iter = 0; F = zeros(I,1);
    J = zeros(I,4); delta = zeros(4,1000);
    while max_delta>toler
        iter = iter + 1;
        mv=ones(I,1)*receiver_loc'-sat_loc_m;
        dmat=sum(mv.^2,2);
        F=sqrt(dmat)+c_mps*tt; % dist. rec-sat
        J=[mv ./ (F*ones(1,3)),c_mps*ones(I,1)];
        Df = distance_m-F;
        delta(:,iter)= J \ Df;
        receiver_loc = receiver_loc + delta(1:3,iter);
        tt = tt + delta(4,iter);
        max_delta = max(abs(delta(:,iter)));
    end
    delta=delta(:,1:iter);
    return
Chapter 8

Appendix

A1 A few properties of a matrix

Eigendecomposition

We consider a $N$ square matrix $A$ of complex values. A non-zero vector $v$ is an eigenvector of $A$ if and only if it exists a scalar $\lambda$ such that:

$$Av = \lambda v \iff (A - \lambda I)v = 0$$

(8.1)

$\lambda$ is called the eigenvalue associated to $v$. It is clear that if $v \in \mathbb{C}^N$ is an eigenvector then, for any $\alpha \in \mathbb{C}$, $\alpha v$ is an eigenvector. It would be better to speak of eigendirection. The equation (8.4) leads to the eigenvalues equation:

$$P(\lambda) = \det(A - \lambda I) = 0$$

$P(\lambda)$ is the characteristic polynomial. It has $N$ roots, which are distinct and/or multiple, and some of them may be zero. For each distinct root $\lambda_i$ we can solve with respect to $v$ the equation:

$$(A - \lambda_i I)v = 0$$

The set of solutions associated to $\lambda_i$ spans a subspace of $\mathbb{C}^N$. In the case where the multiplicity order of all eigenvalue is equal to the dimension of the associated subspace, the matrix is said diagonalizable and we have

$$A = PDP^{-1}$$

where $D$ is a diagonal matrix whose diagonal elements are the eigenvalues and where $P$ is invertible.

It is shown that if $A = A^H$, $A$ is diagonalizable. In particular if $A$ is positive it is diagonalizable, its eigenvalues are positive or null and the eigenvectors
are orthogonal with
\[ A = UD^H \]  \hspace{1cm} (8.2)

\[ = \begin{bmatrix} u_1 & \ldots & u_N \end{bmatrix} \begin{bmatrix} \lambda_1 & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & \lambda_N \end{bmatrix} \begin{bmatrix} u_1 & \ldots & u_N \end{bmatrix}^H \]

where \( U \) is unitary, that is \( UU^H = U^H U = I_N \).

If we denote \( U = \begin{bmatrix} u_1 & \ldots & u_N \end{bmatrix} \) equation (8.2) can be rewritten
\[ A = \sum_{i=1}^{N} \lambda_i u_i u_i^H \]  \hspace{1cm} (8.3)

**Generalized eigendecomposition**

A non-zero vector \( v \) is an generalized eigenvector of the pair \( (A, B) \) if and only if it exists a scalar \( \lambda \) such that
\[ Av = \lambda Bv \iff (A - \lambda B)v = 0 \]  \hspace{1cm} (8.4)

Similarly to the standard case where \( B = I \), we can define subspaces associated with the generalized eigenvalues solutions of \( P(\lambda) = \det(A - \lambda B) = 0 \).

If \( A \) and \( B \) are positive, it is shown that the set of the eigenvectors spans the all space \( \mathbb{C}^N \) and we have
\[ A = BUDU^H \]

where \( D \) is a diagonal matrix whose diagonal elements are the eigenvalues and where \( U \) is an unitary matrix.

**Singular value decomposition**

We consider a \( N \times M \) matrix \( A \) of complex values. It can be shown that there exists \( U, V \) and \( D \) such that:
\[ N \geq M : A = UD^V \]
\[ = \begin{bmatrix} u_1 & \ldots & u_N \end{bmatrix} \begin{bmatrix} \sigma_1 & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & \sigma_M \\ 0 & \ldots & 0 \end{bmatrix} \begin{bmatrix} v_1 & \ldots & v_M \end{bmatrix}^H \]

\[ N \leq M : A = UD^V \]
\[ = \begin{bmatrix} u_1 & \ldots & u_N \end{bmatrix} \begin{bmatrix} \sigma_1 & \ldots & 0 & 0 & \ldots & 0 \\ \vdots & \ddots & \vdots & \ddots & \ddots & \vdots \\ 0 & \ldots & \sigma_N & 0 & \ldots & 0 \end{bmatrix} \begin{bmatrix} v_1 & \ldots & v_M \end{bmatrix}^H \]
where \( U \) is an \( N \) square unitary matrix, \( V \) an \( M \) square unitary matrix and \( D \) an \( N \times M \) diagonal matrix such that the diagonal entry \( \sigma_i \geq 0 \). The diagonal elements \( \sigma_i \) are called the singular values. If \( r \) is the rank of \( A \), only \( r \) singular values are non-null.

It is shown that the squares of the singular values are equal to

- the \( N \) eigenvalues of \( AA^H \) if \( N \leq M \),
- the \( M \) eigenvalues of \( A^HA \) if \( N \geq M \).

If \( A \) has rank \( r \) and if we denote \((u_1,v_1), \ldots, (u_r,v_r)\) the \( r \) singular vector pair associated to the non-null singular values \( \sigma_1, \ldots, \sigma_r \), then

\[
A = \sum_{i=1}^{r} \sigma_i u_i v_i^H \tag{8.5}
\]

\section*{Pseudo-inverse}

We consider a \( N \times M \) matrix \( A \) of complex values. The pseudo inverse is the unique \( M \times N \) matrix \( A^\# \) such that the four following conditions are verified:

- \( AA^\#A = A \)
- \( A^\#AA^\# = A^\# \)
- \( (A^\#A)^* = A^\#A \)
- \( (AA^\#)^* = AA^\# \)

We have the following properties:

- if \( A \) is invertible, \( A^\# = A^{-1} \);
- if the rank of \( A \) is the number of rows \( N \) of \( A \), then \( A^\# = A^H(AA^H)^{-1} \);
- if the rank of \( A \) is the number of columns \( M \) of \( A \), then \( A^\# = (A^HA)^{-1}A^H \);
- if \( A = UDV^H \) is the svd decomposition of \( A \), then \( A^\# = UD^#V^H \), where \( D^# \) is a diagonal matrix whose non-zero elements are the reciprocal of the non-zero singular values.

\section*{Condition number}

Let us denote respectively \( \sigma_{\text{max}} \) and \( \sigma_{\text{min}} \) the highest and the lowest singular values. The ratio \( r = \sigma_{\text{max}}/\sigma_{\text{min}} \) is called the condition number. The higher the condition number, the more difficult the numerical calculation of the pseudo-inverse of a given matrix is.
A2  A few relations for matrices

For more results see [30].

- Let $A$ nonsingular, and $B$, $C$ and $D$ may be rectangular:

\[ (A + CBD)^{-1} = A^{-1} - A^{-1} C (I + BDA^{-1} C)^{-1} BDA^{-1} \]  

- Woodbury:

\[ \left( A + CBC^T \right)^{-1} = A^{-1} - A^{-1} C \left( B^{-1} + C^T A^{-1} C \right)^{-1} C^T A^{-1} \]  

- Kailath:

\[ (A + BC)^{-1} = A^{-1} - A^{-1} B (I + CA^{-1} B)^{-1} CA^{-1} \]  

- Bass-Gura:

\[
\begin{align*}
\det \begin{bmatrix} A & B \\ C & D \end{bmatrix} &= \det(A) \det(D - CA^{-1} B) \\
\Rightarrow \det(I_n - AB) &= \det(I_m - BA)
\end{align*}
\]  

- Searle’s identities:

\[
\begin{align*}
(I + A^{-1})^{-1} &= A (A + I)^{-1} \\
\left( A + BB^T \right)^{-1} B &= A^{-1} B \left( I + B^T A^{-1} B \right)^{-1} \\
(A^{-1} + B^{-1})^{-1} &= A (A + B)^{-1} B = B (A + B)^{-1} A \\
A^{-1} + B^{-1} &= A^{-1} (A + B) B^{-1} \\
(I + AB)^{-1} &= I - A (I + BA)^{-1} B \\
(I + AB)^{-1} A &= A (I + BA)^{-1}
\end{align*}
\]
Bibliography


[34] L. F. Richardson. *The approximate arithmetical solution by finite differences of physical problems including differential equations, with an application to the stresses in a masonry dam*, volume A 210. Philosophical Transactions of the Royal Society, 1911.


Index

ADC, 2
algorithm
  overlap-add, 28
  overlap-save, 25
aliasing, 5
all-pole, 111
ambiguity, 7
analog-to-digital converter (ADC), 2
analysis (principal components), 164
anti-aliasing, 7
AR, 106, 126, 218, 229
autoregressive, 106, 109, 126, 218, 229
band
  useful, 6
band-limited
  signal, 2
bi-orthogonality, 39
CAL, 159
capital allocation line, 159
cardiac rhythm, 131
CD-audio, 34
cepstral coefficients, 120
cepstrum, 120, 221
Cholevsky, 102
clicks, 125
code rate, 181
comb, 34
  filter, 32
compensator, 74
complex gain, 15
constraint length, 181
continued fraction, 91
control points (spline), 96
convergence
  DTFT, 7
conversion
  analog-to-digital, 2
convolution
  circular, 23
  linear, 23
critical frequency, 78
cubic spline, 96
Daubechies, 40
decibel, 8
decimation, 30
declicking, 125
DFT, 8, 9
digital processing, 1
direct form (filter), 16
discrete Fourier transform, 8
dot product, 8
DTFT, 4, 7
DTMF, 11
dual tone multi-frequency, 11
dynamic time warping, 116
ECG, 131
eigenfaces, 167
EKG, 131
energy spectral density, 8
equation
  state, 16
    Yule-Walker, 110, 111, 125
esd, 8
extrinsic parameters (calibration), 56
factorization
  Cholevsky, 102
fast Fourier transform, 8, 10
fetus (EKG), 131
FFT, 8
filter
  (Daubechies), 40
  all-pole, 111
  bank, 29
  definition, 15
  perfect reconstruction, 3
FIR, 20
FIR filter, 15
formant, 109
Fourier
  discrete-time transform, 4
frequency
  fundamental, 138
  fundamental (measuring the), 137
  image, 7
frequency resolution, 9
fricatives, 106
function
  BurtAdelsonInit, 63
calck, 49
calcderiv, 211
calcrn, 87
CalcXY, 208
calibparam, 60
CHOFact, 216
coman, 20
Crepind, 202
dergauss, 144
derivestimc, 87
detectpitch, 217
dettridiag, 213
drawcs, 211
DTW1, 221
expm, 73
extractCEPSTRE, 223
f0cor, 139
filter, 18
filtic, 18
filtrer, 17
filtrerII, 192
filtic, 18
filticII, 193
fsolve (non-linear equations), 90
fzero (non-linear equations), 90
KRdecomp, 61
LDA, 173
LPremes, 48
moygauss, 143
nbilin.m (continuous → discrete time), 77
nbilin, 203
newimg.m, 51
normM, 66
ode45 (linear system simulation), 78
PCA2D, 169
phasevoc, 226
projconique.m, 196
psola, 224
recn.m, 207
reconstruction, 3, 5
rotate, 198
rotx.m, 197
roty.m, 197
rotz.m, 198
secondderiv, 210
selnewfr, 45
solve_lv2, 206
solve_myls.m, 79
spec2sig, 226
specinterp, 227
stft, 227
xtoa, 115
gain
  filter, 15
Gauss-Seidel, 101
generating codes, 181
global positioning system (GPS), 176
glottal excitation, 108
GPS, 176
heart, 131

IDFT, 9
IIR filter, 15
insertion, 31
instantaneous
   power, 13
intrinsic parameters (calibration), 55
inverse DFT, 9
inverse difference (Thiele’s interpolation), 93

Jacobi, 100

Karush-Kuhn-Tucker, 149
KKT, 149

Lagrange multipliers, 166
Lagrangian, 146
LDA, 170
least squares (calibration), 58
lifting scheme, 66
line spectrum pair, 220
linear discriminant analysis, 170
linear filter
   anti-aliasing, 7
   by FFT, 23
   comb, 32, 34
   implementation, 16
   matched, 127
   state, 16
linear prediction, 125
Lotka-Volterra, 80
lsp, 220

MAC operation, 20, 26
matched
   linear filter, 127
matrix
   companion, 20
   confusion, 167, 175
mel frequency cepstral coefficients, 120
method
   Gauss-Seidel, 101
   Jacobi, 100
   Newton, 88
   Newton-Raphson, 89
   relaxation, 101
   Richardson (extrapolation), 85
   Runge-Kutta, 83
MFCC, 120
multifrequency, 29
multiplication-accumulation (MAC), 20, 23

Newton-Raphson, 89
noise
   impulse, 125
   quantization, 2
ode, 76
one-bit stream, 34
ordinary differential equation, 76
overlap, 25, 29
overlap-add, 27
overlap-save, 25
Parks-McClellan method, 43
Parseval
   formula, 8
PCA, 164, 166, 169
PCM, 113
perfect reconstruction, 3
periodogram, 10
   averaged, 10
   smoothed, 10
perspective projection, 51
phase
   vocoder, 123
   phase vocoder, 121
phone dialing, 11
pinhole camera model, 54
pitch, 108
plosives, 106
Poisson
   formula, 4
polynomial approximation, 43
polyphase, 21
polyphases components, 21
power spectral density, 10
predator-prey equation, 80
principal component analysis, 164, 166
processing (multifrequency), 29

program
C2altern.m, 201
Crepetats.m, 202
aa.m, 128
Ccraq.m, 230
correlnegative.m, 239
currences.m (continuous → discrete time), 76
data4exeGPS.m, 179
daub4.m, 41
decode.m, 220
detectkey.m, 189
detectnum.m, 191
detell.m, 233
digitBDDgene.m, 175
DTWtry.m, 224
dwtproc.m, 65
dwtproc.m, 66
estimderiv.m, 87
exeigenfaces.m, 167
exeminQ3linineg.m, 152
exonewton.m, 208
explefzero.m, 90
explefzero2.m, 91
extrythm.m, 140
fenHann.m, 225
fil2blocs.m, 18
filtragefft1.m, 194
filtragefft2.m, 194
filtragefft3.m, 195
genekey.m, 12
LDAPCAtest.m, 242
LDAPCAtraining.m, 241
lv.m, 206
mcomb.m, 195
natcubicspline.m, 210
optimFolio.m, 240
PHcoder.m, 226
polyphase.m, 194
preprocesscoin.m, 142
PSOLAtry.m, 224
rechellipsecovar.m, 235
relaxtest.m, 215
repcren.m (RC filter), 79
residuAR.m, 111
restau.m, 130
rk2lv.m, 206
RK4.m, 84
separecg.m, 135
solve_lv.m, 81
spectralradius.m, 215
testbilin.m (continuous → discrete time), 77
testheun.m, 205
testlv2.m, 206
testode05.m, 80
testremes.m, 47
testtridiag.m, 213
tstCalcXY.m, 209
tstproj3D.m (3D projection), 198
tstpyramide.m, 63
twoassets.m, 237

psd, 10
PSOLA, 121
pulse code modulation, 113

quality factor (Remes), 44

random
sequence, 10
signal, 10

random process
AR, 106, 126, 218, 229
autoregressive, 109
reconstruction (of a state), 18
reconstruction function, 3
reflection coefficients, 220
relaxation, 101
Remes, 43
residual, 111
Index 255

residual signal (prediction), 126
resolution
  frequency, 9
Runge-Kutta, 83

sampling
  deterministic signal, 2
  frequency, 2
  period, 2
satellite positioning, 176
Sharpe’s ratio, 159
signal
  digital, 2
  low-pass, 6
  periodic, 136
  residual, 111
  speech, 106
signal-to-noise ratio, 127
simulation, 74
SNR, 127
sound
  unvoiced, 106
  voiced, 106, 111
spectral radius, 100, 213
spectrum, 2, 8
  aliasing, 3, 5
  random signal, 10
speech
  AR model, 107
  model, 105, 106
  typology of sounds, 106
spline
  cubic, 96
spline curve, 96
state
  of a filter, 16
  reconstruction, 18
  representations, 73

Thiele’s interpolation, 92
transfert function, 15
transform
  discrete Fourier, 8, 9

vocal folds, 106
vocoder, 113
voice activity, 116
vowels, 106

waveform coding, 113
wavelet, 40
window
  Hann, 123
Yule-Walker
  equation, 110, 111, 125
zero-order hold, 74
zero-padding, 9
ZOH, 74
2014

AUGER François
*Signal Processing with Free Software: Practical Experiments*

BLANCHET Gérard, CHARBIT Maurice
*Digital Signal and Image Processing using MATLAB
Volume 1 – Fundamentals – 2nd edition*

FANET Hervé
*Medical Imaging Based on Magnetic Fields and Ultrasounds*

MOUKADEM Ali, OULD Abdeslam Djaffar, DIETERLEN Alain
*Time-Frequency Domain for Segmentation and Classification of Non-stationary Signals: The Stockwell Transform Applied on Bio-signals and Electric Signals*

NDAGIJIMANA Fabien
*Signal Integrity: From High Speed to Radiofrequency Applications*

PINOLI Jean-Charles
*Mathematical Foundations of Image Processing and Analysis – Volumes 1 and 2*
TUPIN Florence, INGLADA Jordi, NICOLAS Jean-Marie
Remote Sensing Imagery

VLADIANU Calin, EL ASSAD Safwan
Nonlinear Digital Encoders for Data Communications

2013
GOVAERT Gérard, NADIF Mohamed
Co-Clustering

DAROLLES Serge, DUVAUT Patrick, JAY Emmanuelle
Multi-factor Models and Signal Processing Techniques: Application to
Quantitative Finance

LUCAS Laurent, LOSCOS Céline, REMION Yannick
3D Video: From Capture to Diffusion

MOREAU Eric, ADALI Tulay
Blind Identification and Separation of Complex-valued Signals

PERRIN Vincent
MRI Techniques

WAGNER Kevin, DOROSLOVACKI Milos
Proportionate-type Normalized Least Mean Square Algorithms

FERNANDEZ Christine, MACAIRE Ludovic, ROBERT-INACIO Frédérique
Digital Color Imaging

FERNANDEZ Christine, MACAIRE Ludovic, ROBERT-INACIO Frédérique
Digital Color: Acquisition, Perception, Coding and Rendering

NAIT-ALI Amine, FOURNIER Régis
Signal and Image Processing for Biometrics

OUAHABI Abdeljalil
Signal and Image Multiresolution Analysis
2011
CASTANIÉ Francis
*Digital Spectral Analysis: Parametric, Non-parametric and Advanced Methods*

DESCOMBES Xavier
*Stochastic Geometry for Image Analysis*

FANET Hervé
*Photon-based Medical Imagery*

MOREAU Nicolas
*Tools for Signal Compression*

2010
NAJMAN Laurent, TALBOT Hugues
*Mathematical Morphology*

2009
BERTEIN Jean-Claude, CESCHI Roger
*Discrete Stochastic Processes and Optimal Filtering / 2nd edition*

CHANUSSOT Jocelyn et al.
*Multivariate Image Processing*

DHOME Michel
*Visual Perception through Video Imagery*

GOVAERT Gérard
*Data Analysis*

GRANGEAT Pierre
*Tomography*

MOHAMAD-DJAFARI Ali
*Inverse Problems in Vision and 3D Tomography*

SIARRY Patrick
*Optimisation in Signal and Image Processing*
2008

ABRY Patrice et al.
*Scaling, Fractals and Wavelets*

GARELLO René
*Two-dimensional Signal Analysis*

HLAWATSCH Franz et al.
*Time-Frequency Analysis*

IDIER Jérôme
*Bayesian Approach to Inverse Problems*

MAITRE Henri
*Processing of Synthetic Aperture Radar (SAR) Images*

MAITRE Henri
*Image Processing*

NAIT-ALI Amine, CAVARO-MENARD Christine
*Compression of Biomedical Images and Signals*

NAJIM Mohamed
*Modeling, Estimation and Optimal Filtration in Signal Processing*

QUINQUIS André
*Digital Signal Processing Using Matlab*

2007

BERTEIN Jean-Claude, CESCHI Roger
*Discrete Stochastic Processes and Optimal Filtering*

BLOCH Isabelle
*Information Fusion in Signal and Image Processing*

GLAVIEUX Alain
*Channel Coding in Communication Networks*

OPPENHEIM Georges et al.
*Wavelets and their Applications*
2006

CASTANIÉ Francis
Spectral Analysis

NAJIM Mohamed
Digital Filters Design for Signal and Image Processing