Digital Signal and Image Processing using MATLAB®

Gérard Blanchet
Maurice Charbit

Digital Signal and Image Processing using MATLAB®

Digital Signal and Image Processing using MATLAB®

Gérard Blanchet
Maurice Charbit

Part of this book adapted from “Signaux et images sous Matlab : méthodes, applications et exercices corrigés” published in France by Hermès Science Publications in 2001
First published in Great Britain and the United States in 2006 by ISTE Ltd
Translated by Antoine Hervier

ISTE Ltd
6 Fitzroy Square
London W1T 5DX
UK

ISTE USA
4308 Patrice Road
Newport Beach, CA 92663
USA

www.iste.co.uk

Library of Congress Cataloging-in-Publication Data

Blanchet, Gérard.
[Signaux et images sous Matlab. English]
Digital signal and image processing using Matlab / Gérard Blanchet, Maurice Charbit.
p. cm.
Translation of: Signaux et images sous Matlab.
Includes index.
ISBN-13: 978-1-905209-13-2
ISBN-10: 1-905209-13-4
1. Signal processing--Digital techniques--Data processing. 2. MATLAB. I.Charbit, Maurice. II. Title.
TK5102.9.B545 2006
621.382'2--dc22
2006012690

British Library Cataloguing-in-Publication Data
A CIP record for this book is available from the British Library
ISBN 10: 1-905209-13-4
ISBN 13: 978-1-905209-13-2

Printed and bound in Great Britain by Antony Rowe Ltd, Chippenham, Wiltshire.

Contents

This page intentionally left blank

Preface

A practical approach through simulation

Simulation is an essential tool in any field related to engineering techniques, whether it is used for teaching purposes or in research and development.

When teaching technical subjects, lab works play an important role, as important as exercise sessions in helping students assimilate theory. The recent introduction of simulation tools has created a new way to work, halfway between exercise sessions and lab works. This is particularly the case for digital signal processing, for which the use of the MATLAB® language, or its clones, has become inevitable. Easy to learn and to use, it makes it possible to quickly illustrate a concept after introducing it in a course.

As for research and development, obtaining and displaying results often means using simulation programs based on a precise "experimental protocol", as it would be done for actual experiments in chemistry or physics.

These characteristics have led us, in a first step, to try to build a set of exercises with solutions relying for the most part on simulation; we then attempted to design an introductory course mostly based on such exercises. Although this solution cannot replace the traditional combination of lectures and lab works, we do wonder if it isn't just as effective when associated with exercise sessions and a few lectures. There is of course no end in sight to the debate on educational methods, and the amount of experiments being conducted in universities and engineering schools shows the tremendous diversity of ideas in the matter.

Basic concepts of DSP

The recent technical evolutions, along with their successions of technological feats and price drops have allowed systems based on micro-controllers and microprocessors to dominate the field of signal and image processing, at the expense of analog processing. Reduced to its simplest form, signal processing amounts to manipulating data gathered by sampling analog signals. Digital

Signal and Image Processing, or DSIP, can therefore be defined as the art of working with sequences of numbers.

The sampling theorem

The *sampling theorem* is usually the first element found in a DSIP course, because it justifies the operation by which a continuous time signal is replaced by a discrete sequence of values. It states that a signal can be perfectly reconstructed from the sequence of its samples if the *sampling frequency* is greater than a fundamental limit called the *Nyquist frequency*. If this is not the case, it results in an undesired effect called *spectrum aliasing*.

Numerical Sequences and DTFT

The *Discrete Time Fourier Transform*, or DTFT, introduced together with the sampling theorem, characterizes the spectral content of digital sequences. The analogy between the DTFT and the continuous time Fourier transform is considered, with a detailed description of its properties: linearity, translation, modulation, convolution, the Parseval relation, the Gibbs phenomenon, ripples caused by windowing, etc.

In practice, signals are only observed for a finite period of time. This "time truncation" creates ripples in the spectrum and makes it more difficult to the separate two close frequencies in the presence of noise. This leads to the concept of frequency resolution. The DTFT is a simple way of separating two frequencies, but only if the observation time is greater than the inverse of the difference between the two frequencies. The frequency resolution will allow us to introduce the reader to weighting windows. However, a more complete explanation of the concept of resolution can only be made if noise disturbing the signal is taken into account, which is why it will be studied further when random processes are considered.

The Discrete Fourier Transform, or DFT is the tool used for a numerical computation of the DTFT. Because this calculation involves a finite number of frequency values, the problem of precision has to be considered. There are a few differences in properties between the DFT and the DTFT, particularly regarding the indexing of temporal sequences that are processed modulo N. Some examples of this are the calculation of the DTFT and the DFT of a sinusoid, or the relation between discrete convolution and the DFT. At this point, the fast algorithm calculation of the DFT, also called FFT (Fast Fourier Transform), will be described in detail.

Filtering and Elements of Filter Design

Linear filtering was originally used to extract relevant signals from noise. The basic tools will be introduced: the discrete convolution, the impulse response, the frequency response, the z-transform. We will then focus on the fundamental relation between linear filtering with rational transfer functions and linear constant-coefficient recursive equations.

Filter design is described based on a few detailed examples, particularly the window method and the bilinear transform. The concepts of over-sampling and under-sampling are then introduced, some applications of which are frequency change and the reduction of quantization noise. From a broader perspective, multi-rate processing and filter banks which are described here, are two subjects that attract a lot of attention in the field of DSIP.

An introduction to images

Image processing is described in its own separate chapter. Many of the concepts used in signal processing are also used in image processing. The only difference is that two indices are used instead of one. However images have particular characteristics that require specific processing: erosion, expansion, etc. The computation time is usually much longer for images than it is for signals. It is nevertheless possible to conduct image processing with MATLAB® or one of its clones. This theme will be discussed using examples on 2D filtering, contour detection, and other types of processing in cases where the 2D nature of the images does not make them too different from a 1D signal. This chapter will also be the opportunity to discuss image compression and entropic coding.

Random Processes

Up until now, the signals used as observation models have been described by functions that depend on a finite number of well known parameters and on simple known basic functions: the sine function, the unit step function, the impulse function... This type of signal is said to be deterministic.

There are other situations where deterministic functions cannot provide us with a relevant apprehension of the variability of the phenomena. Signals must then be described by characteristics of a probabilistic nature. This requires the use of random processes, which are time-indexed sequences of random variables. Wide sense stationary processes, or WSSP, are an important category of random processes. The study of these processes is mainly based on the essential concept of *power spectral density*, or PSD. The PSD is the analog for WSSP of the square module of the Fourier transform for deterministic signals. The formulas for the linear filtering of WSSP are then laid down. Thus, we infer that WSSPs can also be described as the linear filtering of a white noise. This result leads to a large class of stationary processes: the AR process, the MA process, and the ARMA process.

Spectral Estimation

One of the main problems DSIP is concerned with is evaluating the PSD of WSSPs. In the case of continuous spectra, it can be solved by using non-parametric approaches (smooth periodograms, average periodograms, etc.) or parametric methods based on linear models (AR, MA, ARMA). As for line spectra, the most commonly used methods are the periodogram and what are called high resolution methods, which use the structures of the signal and the noise: *Prony*, *Pisarenko*, *MUSIC*, *ESPRIT*, etc.

The least squares

This chapter discusses the use of the least squares method for solving problems. This method is used in a number of problems, in fields such as spectral analysis, modelling, linear prediction, communications... We will discuss such methods as Wiener, RLS, LMS, Kalman...

Applications

This last chapter presents case studies that go a little further in depth than the examples described earlier. The emphasis is set on audio signal processing, on compression as well restoring and denoising for speech and music, and on modulation, demodulation and equalization issues for digital communications. This chapter is also an opportunity to discover typical approaches and algorithms: pitch detection, PSOLA, DTW, ACP, LBG, Viterbi...

As a Conclusion

One of the issues raised by many of those who use signal processing has to do with the artificial aspect introduced by simulation. For example, we use sampling frequencies equal to 1, and therefore frequencies with no dimension. There is a risk that the student may lose touch with the physical aspect of the phenomena and, because of that, fail to acquire the intuition of these phenomena. That is why we have tried, at least in the first chapters, to give exercises that used values with physical units: seconds, Hz, etc.

This work discusses important properties and theorems, but its objective is not to be a book on mathematics. Its only claim, and certainly an excessive one, is to show how interesting signal and image processing can be, by providing themes of study we chose because they were good examples, because they were simple, while trying not to be too trivial.

All of the subjects discussed far from cover the extent of knowledge required in this field. However they seem to us to be a solid foundation for an engineer who would happen to deal with DSIP problems.

Notations and Abbreviations

$\emptyset$		Empty Set
$\sum_{k,n}$	$=$	$\sum_k \sum_n$
$\mathrm{rect}_T(t)$	$=$	$\begin{cases} 1 & \text{when} \quad \lvert t\rvert < T/2 \\ 0 & \text{otherwise} \end{cases}$
$\mathrm{sinc}(x)$	$=$	$\dfrac{\sin(\pi x)}{\pi x}$
$\mathbb{1}(x \in A)$	$=$	$\begin{cases} 1 & \text{when } x \in A \\ 0 & \text{otherwise} \end{cases}$ (Indicator Function of A)
$(a,b]$	$=$	$\{x : a < x \leq b\}$
$\delta(t)$		$\begin{cases} \text{Dirac Distribution when } t \in \mathbb{R} \\ \text{Kronecker Symbol when } t \in \mathbb{Z} \end{cases}$
$\mathrm{Re}(z)$		Real Part of z
$\mathrm{Im}(z)$		Imaginary Part of z
i or j	$=$	$\sqrt{-1}$
$x(t) \rightleftharpoons X(f)$		Fourier Transform
$(x \star y)(t)$		Continuous Time Convolution
	$=$	$\int_{\mathbb{R}} x(u)y(t-u)du$
$(x \star y)(t)$		Discrete Time Convolution
	$=$	$\sum_{u \in \mathbb{Z}} x(u)y(t-u)$

$\mathbf{I}_N$	$(N \times N)$-dimension Identity Matrix
$\mathbf{A}^*$	Complex Conjugate of $\mathbf{A}$
$\mathbf{A}^T$	Transpose of $\mathbf{A}$
$\mathbf{A}^H$	Transpose-Conjugate of $\mathbf{A}$
$\mathbf{A}^{-1}$	Inverse Matrix of $\mathbf{A}$
$\mathbf{P}(X \in A)$	Probability that $X \in A$
$\mathbb{E}\{X\}$	Expectation Value of X
$X_c = X - \mathbb{E}\{X\}$	Zero-mean Random Variable
$\text{var}(X) = \mathbb{E}\{\lvert X_c\rvert\}^2$	Variance of X
$\mathbb{E}\{X\lvert Y\}$	Conditional Expectation of X given Y
ADC	Analog to Digital Converter
ADPCM	Adaptive Differential PCM
AMI	Alternate Mark Inversion
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

HDB	High Density Bipolar
IDFT	Inverse Discrete Fourier Transform
i.i.d./iid	Independent and Identically Distributed
IIR	Infinite Impulse Response
ISI	InterSymbol Interference
LDA	Linear discriminant analysis
lms	Least mean squares
MA	Moving Average
MAC	Multiplication ACcumulation
OTF	Optical Transfer Function
PAM	Pulse Amplitude Modulation
PCA	Principal Component Analysis
p.d.	Probability Distribution
ppi	Points per Inch
p.s.d./PSD	Power Spectral Density
PSF	Point Spread Function
PSK	Phase Shift Keying
QAM	Quadrature Amplitude Modulation
rls	Recursive least squares
rms	Root mean square
r.p./rp	Random process
SNR	Signal to Noise Ratio
r.v./rv	Random variable
STFT	Short Term Fourier Transform
TF	Transfer Function
WSS	Wide (Weak) Sense Stationary (Second Order) Process
ZOH	Zero-Order Hold
ZT	z-Transform

This page intentionally left blank

Introduction to MATLAB

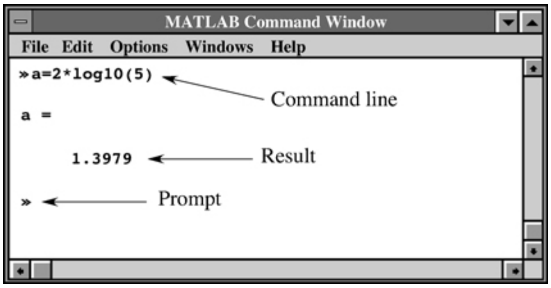
In this book the name MATLAB® (short for Matrix Laboratory) will refer to:

– the program launched by using the command `matlab` in Dos or Unix environments, or by clicking on its icon in a graphic environment such as x11, Windows, MacOS...,

– or the language defined by a vocabulary and syntax rules.

MATLAB® is an interpreter, that is to say a program that remains in the computer's memory once it is launched. MATLAB® displays a command window used for interpreting commands. If they are considered correct, MATLAB® will execute them. This execution will itself lead to verifications.

Example 1 (Direct interpretation) Type `a=2*log10(5)` then <`return`>. The result is shown in a PC environment (Figure 1).

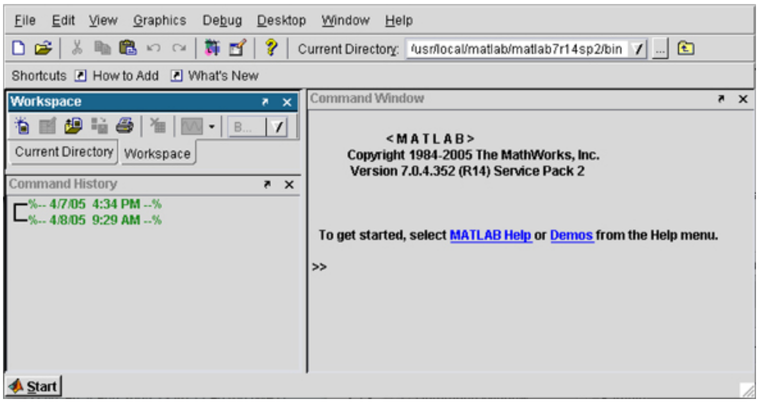


```
»a=2*log10(5)

a =

      1.3979

»
```

Figure 1 – *The* MATLAB® *command window on MS-Windows*

Commands can be gathered together in text files called *matlab programs*. The user gives them a name that can be called from the prompt line. The MATLAB® documentation explains how to use an editor to create such files.

This editor may either be integrated in the software or kept external (the user's favorite editor). Program files use the extension `.m`. If a program is called `prog1.m`, all the user has to do is type `prog1` in the MATLAB® command window to have it executed. MATLAB® then searches for the file in the routine directory. If it doesn't find the file there, it looks for `prog1.m` in the various files specified in the directory path. The latter can be defined directly in the command prompt window, or by using a program and executing commands such as `path`, `addpath`, `rmpath`, `genpath`, `pathtool`, `savepath` (see documentation, online help, or type `help path`).


File Edit View Graphics Debug Desktop Window Help
Current Directory: /usr/local/matlab/matlab7r14sp2/bin
Shortcuts How to Add What's New
Workspace
Current Directory Workspace
Command History
Command Window
Start

```
%-- 4/7/05  4:34 PM --%
%-- 4/8/05  9:29 AM --%
```

```
                < M A T L A B >
    Copyright 1984-2005 The MathWorks, Inc.
     Version 7.0.4.352 (R14) Service Pack 2

To get started, select MATLAB Help or Demos from the Help menu.

>>
```

Figure 2 – *The* MATLAB® *window in an X-windows environment. The definition of the routine folder can be done directly by clicking on the icon with "..." in the top-right corner of the window. The definition of the directory path can be done by selecting the item* `set path ...` *in the menu* `file`

Clones of MATLAB® are now available. Some belong to the public domain. There also exists a compiler that allows the user to translate MATLAB® programs in machine language, making the execution quicker, and meaning that it is not required to own the interpreter.

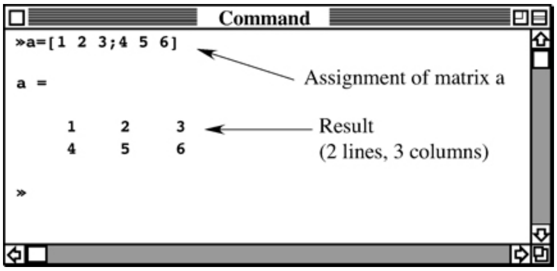
1 Variables

1.1 Vectors and matrices

The MATLAB® language is dedicated to matrix calculations and was optimized in this perspective. The variables handled as a priority are real or complex matrices. A *scalar* is a 1×1 matrix, a *column vector* is a matrix with only one column, and a *line vector* a matrix with only one line.

The notation $(\ell \times c)$ indicates that the considered variable has ℓ lines and c columns.

Example 2 (Assignment of a real matrix) Type `a=[1 2 3; 4 5 6]` at the MATLAB® prompt in the command window. The answer is shown in Figure 3.



```
»a=[1 2 3;4 5 6]

a =

     1     2     3
     4     5     6

»
```

Figure 3 – *Assigning a matrix*

Values are assigned to the elements of a matrix by using brackets. A space (or a comma) is a separator, and takes you to the next column, while the semi-colon takes you to the next line. **Elements are indexed starting from 1.** The first index is the line number, the second one is the column number. In our example, `a(1,1)=1` and `a(2,1)=4`. The assignment `a=[1 2;3 4 5]` will of course lead to an error message, since the number of columns is different for the first and second lines.

Character strings can also be assigned to the elements of a matrix. However, the string length must be compatible with the structure of the matrix. For example, `N=['paul';'john']` would be correct, whereas `N=['paul';'peter']` would cause an error.

When the vector's components form a sequence of values separated by regular intervals, it is easier to use what is called an “implicit” loop of the type `(indD:step:indF)`. This expression refers to a list of values starting at `indD` and going up to `indF` by increments of `step`. Values cannot go beyond `indF`. The increment value `step` can be omitted if it is equal to 1.

Example 3 (Implicit enumeration) Type `a=(0:1:10)` or `a=(0:10)`. MATLAB® returns:

```
a =
     0   1   2   3   4   5   6   7   8   9   10
```

Example 4 (Incremented implicit enumeration) Type `a=(0:4:10)`. MATLAB® returns:

```
a =
     0   4   8
```

The last element of a vector is indicated by the reserved word `end`. In the previous example, `a(end)` indicates that its value is `8`.

It is possible to extend the size of a matrix. The interpreter takes care of available space by dynamically allocating memory space during the analysis of the typed phrase.

Example 5 (Extension of matrix) Type the following commands one after the other:

```
>>a=[1 2 3; 4 5 6]
a =
     1     2     3
     4     5     6
>>a=[a a]
a =
     1     2     3     1     2     3
     4     5     6     4     5     6
>>a=[1 2 3; 4 5 6];
>>a=[a;a]
a =
     1     2     3
     4     5     6
     1     2     3
     4     5     6
```

COMMENTS:

– When defining variables and objects, the language takes into account whether letters are capital or lowercase.

– Typing ";" at the end of a command line prevents the program from displaying the results of an operation.

– The display format can be modified by using the `format` command. Executing `format long`, for example, changes the number of significant digits from 5 to 15.

– The user must bear in mind that MATLAB® dedicates memory space every time a variable is used for the first time. All of the variables used during a work session are stored in the computer's memory, which means it is necessary to free space from time to time so as not to get the `OUT OF MEMORY` error message (see the `clear` command in the documentation or type `help clear`).

1.2 Arrays

Multidimensional arrays (not supported by all versions) are an extension of the normal two-dimensional matrix. One way to create such an array is to start with a 2-dimension matrix that already exists and to extend it. Type:

```
A=[1:3;4:6]
>> A
A =
     1     2     3
     4     5     6
>> A(:,:,2)=zeros(2,3), % or A(:,:,2)=0
A(:,:,1) =

     1     2     3
     4     5     6
A(:,:,2) =
     0     0     0
     0     0     0
```

The `repmat` and `cat` functions are provided in order to build multidimensional arrays.

1.3 Cells and structures

In the most recent versions of MATLAB®, there are two groups of data that are more elaborate than scalar arrays and character string arrays: the first one is called a *cell* and the second a *structure*.

In an array of *cells*, the elements can be of any nature, numerical value, character string, array, etc. Type:

```
langcell={'MATLAB',[6.5;2.3],2002}
>> langcell(2)
ans =
    [2x1 double]
>> langcell{2}
ans =
    6.5000
    2.3000
>> langcell{2}(1)
ans =
    6.5000
```

`langcell` is made up of three elements: the first one is a character string, the second one is a column vector, and the third one is a scalar. This example shows the difference in syntax between an array and a cell, a left brace ({) and a right brace (}) being used instead of a left square bracket ([) and a right square bracket (]). As for the content, `langcell(2)` refers to the vector `[6.5000;2.3]`, `langcell{2}` to the content of this vector, and `langcell{2}(1)` to the numerical value 6.5.

A *structure* is defined by the `struct` instruction. The following example defines a structure, called `langstruc`, comprising three *fields*: `Language`, `Version`, and `Year`. The instruction assigns the character string `MATLAB` to the first field, the character string `6.5` to the second field, and the numerical value 2002 to the third field:

```
>>langstruc=struct('Language','MATLAB','Version','6.5','Year',2002);
>>langstruc.Year
ans =
       2002
>>
```

The second instruction displays the content of `langstruc.Year`, which is `2002`. A `1 x 1` dimension structure is organized in the same way as an `n x 1` dimension array of cells, where `n` is the number of fields of the structure. Cells can therefore be compared to structures with unnamed fields.

The following example defines a structure named `langstruc`, comprised of two recordings. Each recording contains all three fields `Language`, `Version`, and `Year` to which were respectively assigned the sequences of two character strings `MATLAB` and `C`, of the two values 6.5 and 15.1, and of the two values 2002 and 2003:

```
>> langstruc=struct('Langage',{{'MATLAB','C'}},...
           'Version',[6.5;15.1],'Year',[2002;2003]);
>> langstruc
langstruc =
   Language: {'MATLAB'  'C'}
    Version: [2x1 double]
       Year: [2x1 double]
>> langstruc.Langage{1}
ans =
MATLAB
>> langstruc.Language(1)
ans =
    'MATLAB'
>>
```

These objects can be handled using certain functions: `isstruct`, `fieldnames`, `setfield`, `rmfield`, `cellfun`, `celldisp`, `num2cell`, `cell2mat`, `cell2struct`, `struct2cell`... An example of a conversion is as follows:

```
>> clear all
>> langcell={'MATLAB',[6.5;2.3],2002}
>> chps={'Langage','Version','Year'};
>> cell2struct(langcell,chps,2)
ans =
   Language: 'MATLAB'
    Version: 6.5000
       Year: 2002
>>
```

The 2 that is part of the instruction `cell2struct(langcell,chps,2)` indicates the dimension of `langcell` that needs to be taken into account to define the number of fields. Here, for example, `size(langcell,2)` means that the number of fields is 3.