Aurelio Uncini

Adaptive Systems and Neural Networks

Complements and Exercises

Department of Information Engineering,
Electronics and Telecommunications
Sapienza University of Rome

Additional material for students of the courses:
"Neural Networks"

DRAFT - September 2016

Preface

The term adaptation is used in biology in relation to how living beings adapt to their environments, but with two different meanings. First, the continuous adaptation of an organism to its environment, so as to maintain itself in a viable state, through sensory feedback mechanisms. Second, the development (through evolutionary steps) of an adaptation (an anatomic structure, physiological process or behavior characteristic) that increases the probability of an organism reproducing itself (although sometimes not directly).

Generally speaking, an adaptive system is a set of interacting or interdependent entities, real or abstract, forming an integrated whole that together are able to respond to environmental changes or changes in the interacting parts. Feedback loops represent a key feature of adaptive systems, allowing the response to changes; examples of adaptive systems include: natural ecosystems, individual organisms, human communities, human organizations, and human families. Some artificial systems can be adaptive as well; for instance, robots employ control systems that utilize feedback loops to sense new conditions in their environment and adapt accordingly

from Wikipedia
https://en.wikipedia.org/wiki/Adaptive_system

Contents

Chapter 1
Introduction to Adaptive Systems and Algorithms

These introductory exercises for the students of Neural Networks and Adaptive Algorithms courses are useful to introduce, considering a practical and naïve approach, of some fundamental concepts largely used in Neural Networks, Adaptive Filtering and, more in general, in Machine Intelligence framework.

In particular, is introduced in a simple linear in the data context, the problem of supervised learning from available data, i.e. the so called *data-driven* approach.

Moreover, is underlined that a learning from data paradigm, can be formalized as an optimization problem defined over a statistic or deterministic *cost* or *loss function* (CF) or (LF). In addition, as we shall see, such optimization problem can be solved with a on-line or batch or algorithms.

The real-time processing of stream of data represents a central aspect in the so called *data stream mining* (DSM) that is defined as the process of extracting knowledge structures from continuous and rapid data records. Usually the stream of data consists in sequence of instances that in many applications can be read only once or a small number of times using limited computing and storage capabilities.

Lets $\mathbf{w}$ define the vector of parameters that should be adaptively determined as a function of the available data stream, a general on-line adaptation procedure can be written as

$$\mathbf{w}_{new} = \mathbf{w}_{old} + \Delta\mathbf{w} \tag{1.1}$$

where $\Delta\mathbf{w}$ represents the *weights variation*, that can be computed according to a some *a priori* defined rule that usually has the form $\Delta\mathbf{w} = f(\text{new available data})$. In fact, the expression (1.1) represents a generic and simple *learning from data* paradigm, that is the basic *machine learning* on-line adaptation rule.

The applicability of adaptive methods (Neural Networks and other Computational Intelligence methods) is wide and very diversified across many real world sectors and, in principle, we can divide the applications in two main categories:

- Static data processing: patterns recognition, cluster analysis, associative memory, regression, etc;
- Dynamic data processing: non-linear filtering, prediction, dynamic pattern recognition, etc.

In the note are proposed some introductory exercises, both for static and dynamic data, that can be done by the reader with no special knowledge other than basic course on calculus, linear algebra, basic statistics and other basic courses of science and engineering schools.

The exercises have two objectives, the first is to habituate the student to think in terms of batch and on line adaptive algorithms; while the second is to use examples that allow the review of of some basic arguments such as: the theory of dynamical circuits and systems, signal processing, statistics and stochastic processes, the basic principles of optimization and, finally, the introduction the topics of regression, patterns recognition and the other problems that are usually solved by Computational Intelligence Methods.

1.1 Batch vs On-line Algorithms: an Example of Mean and Variance Estimation

In batch estimate is necessary to know *a priori* all the available data, while in the case of on-line estimate, as new data is available, the estimate value is update considering the new available information. The main characteristic of online approach are:

1. the estimation is done recursively, and there is no need to keep in memory the data available;
2. the estimated value is immediately available, and it is not necessary wait for the entire population analysis;
3. the run-time estimated value can be affected by errors;
4. the final error is usually equal or greater than that obtained using a batch algorithm.

As a first introduction of on-line and batch algorithm, we consider the simple estimate of the mean and variance values of a certain set of data referred to as population. However, before introducing the algorithms, provide the reader with a brief summary of some of the introductory statistical concepts [1].

Let x be a *random variable*[1] (RV) we define the *probability density function* (pdf) $f_x(x)$ as a non-negative integrable function such that

$$p(a \leq x \leq b) = \int_a^b f_x(x)dx \tag{1.2}$$

where $p(a \leq x \leq b)$ is the probability that the RV x falls in the interval a and b. For example if the RV represents the height of a population of students in meters, the term $p(1.75 \leq x \leq 1.85)$ denotes the probability that this population has a height between the extremes 1.75 [m] and 1.85 [m].

Moreover, we have that

[1] Given an event indicated as ζ a RV is a number $x \in (\mathbb{R}, \mathbb{C})$ assigned to (every) ζ. This number, sometime indicated as $x(\zeta)$, can be the gain in a game of chance, the voltage of a random source, ..., or any numerical quantity that is of interest in the performance of the experiment [1].

$$\int_{-\infty}^{+\infty} f_x(x)dx = 1. \tag{1.3}$$

The above property derives directly from the definition (1.2), and it is indicated as *certain event*. Considering the example of the height of the students, we have that the height is definitely between the extremes $(-\infty, \infty)$.

The probability density function completely characterize a RV. However, in many situations is convenient or necessary to represent more concisely the RV through a few specific parameters that describe its average behavior. These numbers, defined as *statistical averages* or *moments*, are functional defined as

$$\mu_n = \int_{-\infty}^{+\infty} (x-c)^n f_x(x)dx \tag{1.4}$$

where n is the moment's order and c is a suitable constant and, without further explanation, usually refers to the above expression with $c = 0$.

The *first order moment*, denotes as *mathematical expectation*, is a mathematical functional defined as in Eqn. (1.4) is $n = 1$ and $c = 0$. That is

$$E\{x\} = \int_{-\infty}^{\infty} x f_x(x)dx \tag{1.5}$$

where $E\{x\}$ indicates the *expected value* or *mean value*. The expected value is also indicated as $\mu = E\{x\}$, or sometime indicated as $\bar{x} = E\{x\}$.

We define *variance*, indicated as σ_x^2 or $\text{var}(x)$, when in Eqn. (1.4) is $n = 2$ and $c = \mu$. So, the value variance, or second order central moment, is

$$\sigma_x^2 = \text{var}(x) = E\left\{[x - E\{x\}]^2\right\} = \int_{-\infty}^{\infty} (x-\mu)^2 f_x(x)dx \tag{1.6}$$

where μ is the expected value (1.5); while the positive constant $\sigma_x = \sqrt{\sigma_x^2}$ is defined as *standard deviation* of $\mathbf{x}$. The variance is never negative and is zero only when the x is a constant value. Moreover, expanding the square the variance can be written in term of expectation of x and x^2

$$\sigma_x^2 = E\{x^2\} - [E\{x\}]^2 \tag{1.7}$$

i.e. the variance is the *mean of square minus square of mean.*

Remark 1.1. Even if for the determination of statistical moments is formally necessary known the pdf, these statistical averages can be estimated somehow, without explicit pdf knowledge.

In many real cases of interest, the pdf of the RV is unknown. In the presence of experimental data one can proceed, at times even in a simple empirical way, one can proceed to estimates of statistical quantities of interest such as the average value, the variance, etc. So, in this first part of the chapter we want to introduce some simple intuitive and empirical methods for estimating some statistics quantity by a *batch* and *on-line* approach, referred to as *naïve algorithms.*

Given a set of values of a discrete RV that define a population, indicated as $x[n]$ for $n = 1, 2, \ldots N$, the Eqn. (1.5) can be interpreted in terms of *relative frequency* by the following expression

$$w = \frac{1}{N}\sum_{n=1}^{N} x[n] \tag{1.8}$$

which represents the mean value calculated on a sample of N data using a block (or batch) approach. Note that, the previous expression is an *estimate* of the expected value $E\{x\}$, and if the estimator is *consistent*, we have that $\lim_{N\to\infty} \frac{1}{N}\sum_{n=1}^{N} x[n] \to E\{x\}$.

Thus, with similar reasoning, it is also possible to estimate the the second order moment in terms of relative frequency. In fact, considering the Eqn. (1.6), the formula for estimate the variance, can be written as

$$\sigma_x^2 = \frac{1}{P}\sum_{n=1}^{N}(x[n] - w)^2 \tag{1.9}$$

where, regardless of the type of estimator used, biased or unbiased we have that $P = N$ or $P = N - 1$.

From previous expressions for the variance calculation, you must first calculate the mean value with the Eqn.(1.8) and then variance with the Eqn. (1.9). Because it requires double the passage of data, this approach is referred to as standard *two-pass algorithm*. Moreover, in order to avoid the two-pass nature of standard algorithm, an alternative formulation for the variance estimation is based on the following form

$$\sigma_x^2 = \frac{\sum_{n=1}^{N} x^2[n] - \frac{\left(\sum_{n=1}^{N} x[n]\right)^2}{N}}{P}. \tag{1.10}$$

The above formula is often referred to as *one-pass algorithm* and is recommended in many statistics textbooks.

Remark 1.2. Although formally the Eqn. (1.10) is identical to the two-steps form, the quantities $\sum_{n=1}^{N} x^2[n]$ and $\frac{1}{N}(\sum_{n=1}^{N} x[n])^2$ may assume very large values and may be calculated with rounding error. Furthermore, for small variance values such terms may cancel each other in subtraction operations leading to inaccurate results or negative estimation of variances value (see [4] for details).

1.1.1 On-line Mean Value Estimation

The estimated of the mean value of a population using an on-line approach is based on a recursive algorithm. To start the recurrence we consider that the estimated mean value is identical to the first value available; i.e. $w_1 = x[1]$ and this estimate is updated when new samples of the sequence are available. Reasoning intuitively, at the arrival of the second sample the estimate is updated as: $w_2 = (x[1] + x[2])/2$, $w_3 = (x[1] + x[2] + x[3])/3$ and so on. Expressing more formally this process we can write

$$\begin{aligned} n=1 \qquad & w_1 = x[1] \\ n=2 \qquad & w_2 = \frac{x[1]+x[2]}{2} = \frac{w_1+x[2]}{2} \\ n=3 \qquad & w_3 = \frac{x[1]+x[2]+x[3]}{3} = \frac{2\cdot w_2+x[3]}{3} \\ \vdots \qquad & \vdots \qquad\qquad\qquad \vdots \end{aligned}$$

Generalizing the previous expressions, for the k-th value of the sequence, we can write the following recursive formula

$$\begin{aligned} w_k &= \frac{(k-1)\cdot w_{k-1}+x[k]}{k} \\ &= \frac{1}{k}x[k]+\frac{k-1}{k}w_{k-1}. \end{aligned} \tag{1.11}$$

In addition, note that from Eqn. (1.11) we can implement the algorithm using a single multiplication with the following formula:

$$w_k = w_{k-1} + \frac{x[k]-w_{k-1}}{k}. \tag{1.12}$$

Remark 1.3. Note that the previous formula can be interpreted as

$$w_{new} = w_{old} + \Delta w$$

where $\Delta w = \frac{x[k]-w_{k-1}}{k}$, that is the basic form of an *adaptive algorithm* (1.1).

1.1.2 On-line Variance Estimation

The on-line estimation of variance can be derived from Eqn. (1.10). The following formulas can be used to update the estimated both mean and variance values of a sequence, given a new sample $x[k]$ (see [2], [4] for details).

$$\begin{aligned} \delta_w &= \frac{x[k]-w_{k-1}}{k} \\ w_k &= w_{k-1}+\delta_w \\ \sigma_k^2 &= \frac{(k-1)\sigma_{k-1}^2}{k} + \delta_w(x[k]-w_k). \end{aligned} \tag{1.13}$$

Note that the above expression is numerically more robust than the direct implementation of the one-pass algorithm of Eqn. (1.10).

Example 1.1. Below is a simple code written in MATLAB language that implements Eqn.s (1.13) for the on-line estimation of the mean and variance values of a given population of N elements.

```
%--------------------------------------------------------------------------
%  Naive Introduction of Adaptive Algorithms
%  Demo program for on-line mean and variance estimation
```

```
%
%   Copyright 2016 - A. Uncini
%   DIET Dpt - University of Rome 'La Sapienza' - Italy
%   $Revision: 1.0$  $Date: 2016/03/15$
%-------------------------------------------------------------------------
clear all;  close all;
%
N  = 100000;              % Population size
w0 = rand(1,1);           % Target random mean value [0, 1]
s2 = 1 + rand(1,1) - 0.5; % Target random variance value [0.5, 1.5]
% Generate a Gaussian process with a given mean value and variance
x = w0 + sqrt(s2)*randn(N,1);
%
%-------------------------------------------------------------------------
% on-line mean value and variance estimation ----------------------------
%-------------------------------------------------------------------------
w = 0;  s = 0; % Initial conditions
for k=1:N
    % Mean value estimation
    deltaW = (x(k) - w)/k;
    w = w + deltaW;
    % Variance estimation
    s = (k-1)*s/k + deltaW*(x(k) - w);
end
%Print results and comparison with mean() and var() Matlab routines -------
fprintf('Estimated mean and variance values of a given data samples\n');
fprintf('On-line alg. w0 = %f  est.(w=%f)  - Var s2 = %f est.(s=%f)\n', ...
    w0,w,s2,s);
fprintf('Batch alg.   w0 = %f  est.(w=%f)  - Var s2 = %f est.(s=%f)\n', ...
    w0,mean(x),s2,var(x));
% ------------------------------------------------------------------------
```

The program output is:

```
Estimated mean and variance values of a given data samples
On-line w0 = 0.304697 est.(w=0.305027) - Var s2 = 0.701216 est.(s=0.704861)
Batch   w0 = 0.304697 est.(w=0.305027) - Var s2 = 0.701216 est.(s=0.704868)
»
```

Remark 1.4. In on-line estimation the partial result is available immediately, while in the batch approach the mean value is calculated based on the knowledge of all sequence. For which the result is available after a delay equal to the length of the analysis window.

1.2 Introduction to Least Squares Methods

Given set of data in form of numerical samples *independent and identically distributed* (IID), for example derived from observations of a given phenomenon, which can be a physical system, a sociological phenomenon, an ecological system, an economic system, ...; a basic form to create some knowledge structure, is to determine a mathematical model that can be associated to the available data.

The *method of least squares* (LS) with its variants, represents a wide class of methods to '*best fit*' a given parametric model $\mathbf{y} = f(\mathbf{X}, \mathbf{w})$ (sometime called *hypotheses* f)

to a set of observations $\mathbf{d} \in \mathbb{R}^{N\times 1}$, where $\mathbf{X} \in \mathbb{R}^{N\times M}$ is the known *data matrix* and $\mathbf{w}$ is the unknown set of parameters of the supposed model, to be estimated according to some quality index.

The term "least squares" means that the quality index is related to the sum of the squares of the error, where the error is defined as the difference between the *desired response* of the model (that somehow you have to know or hypothesize) and that obtained from the assumed model, that is: $\mathbf{e} = \mathbf{d} - f(\mathbf{X}, \mathbf{w})$.

The method of LS can by summarized as:

> *Given the model of the data* $\mathbf{y} = f(\mathbf{X}, \mathbf{w})$ *that is* a priori *known, and a set of observations* $\mathbf{X}, \mathbf{d}$*, the parameter vectors* $\mathbf{w}$ *can be estimated by minimizing some quality index related to the squares of the measurement error (or residual)* $\mathbf{e}^T\mathbf{e}$*, through an optimization algorithm.*

The LS method can be seen as a simple *supervised learning algorithm* on the available data (see Fig. 1.1); i.e. based on the so called *data-driven* approach. The method of LS represents a key computational tool in many fields, such as statistics, machine learning, signal processing, control theory, and so on.

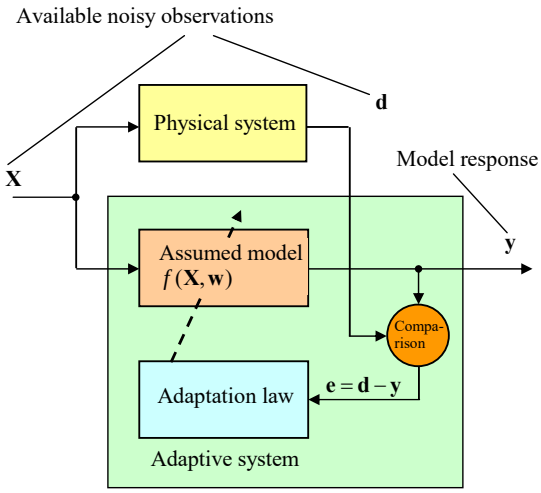

Fig. 1.1 An adaptive system that can learn from examples: the assumed model $f(\mathbf{X}, \mathbf{w})$ that represents the physical system, can be learned through an adaptive procedure that minimize an index quality that, in this case, is a function of the difference between the system and the model response.

The LS estimation problem can be formalized as an optimization algorithm defined over a statistic or deterministic *loss function* (LF). In addition, as we shall see, such optimization problem can be solved with a batch or on-line algorithm.

Least squares problems fall in three main categories:

- linear or *ordinary linear least squares*, where $\mathbf{y} = \mathbf{X}\mathbf{w}$,
- *nonlinear least squares*, where $\mathbf{y} = f(\mathbf{X}, \mathbf{w})$,

- *linear in the data least squares* in the case that given the nonlinear function $\mathbf{y} = f(\mathbf{X}, \mathbf{w})$, through a change of variables, e.g. $\mathbf{v} = g(\mathbf{w})$, there is a linear representation of the problem, e.g. $\mathbf{y} = f(\mathbf{X}, g^{-1}(\mathbf{w})) = \mathbf{X}\mathbf{v}$, depending on whether or not the problem of free parameters estimation can be solved by a set of linear or non linear equations.

In the following we introduced, with a bottom-up approach, the fundamental concepts of batch and on-line data-driven modeling.

1.2.1 Naïve Approach to LS Methods: Fitting Curve to Experimental Data

To introduce intuitively, the LS method, consider a simple problem of data fitting.

The *curve fitting*, is a data-driven process for the determination the free parameters of a curve *a priori* defined as a specific mathematical function like: polynomial, exponential, ...; that describes a possible model of the data, i.e. which has the best correspondence (or best fitting) to a number of assigned points coming from experimental observations or measurements.

Given a set of data related to experimental results or measurements, the curve fitting is a simple *data-driven* process for the determination of a curve (or a specific mathematical function) that describes a possible model of data, which has the best correspondence, or *best fits*, to a number of assigned points coming from the available data set.

For more general and deeper analysis, considering the observed data affected by uncertainty called *measurement error* or *residue*. In the case where the estimation of the curve fitting is carried out by minimizing the effect of this uncertainty, the problem is referred to as *regression analysis* and is dealt in the framework of *estimation theory*.

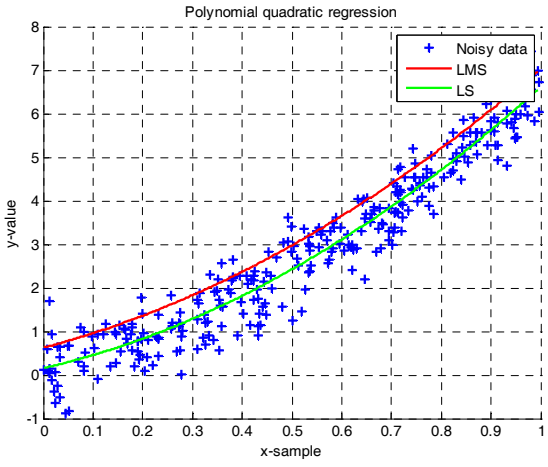

Fig. 1.2 Example of quadratic polynomial fitting curve determined by least mean squares (LMS) and least squares (LS) algorithms.

1.2.2 Polynomial Regression

Given N pairs of experimental data or observations

$$\mathbf{x}_d = [\, x_i \; d_i \,]_{i=1}^{N} \tag{1.14}$$

the P-degree polynomial function defined at the observed data point x_i:

$$f(x_i, \mathbf{w}) = w_0 + w_1 x_i + w_2 x_i^2 + \cdots w_P x_i^P \tag{1.15}$$

is indicated as *polynomial fitting* function and it is assumed, as *a priori* hypotheses, as the *model of the available data*.

For a more compact formulation, useful for the development of calculation methods, the previous expression can be written as dot product:

$$f(x_i, \mathbf{w}) = \mathbf{w}^T \mathbf{x}_i \tag{1.16}$$

where the vectors $\mathbf{x}$ and $\mathbf{w}$ are, respectively, defined as

$$\begin{aligned} \mathbf{w} \in \mathbb{R}^{(P+1)\times 1} &= \left[\, w_0 \; w_1 \; \cdots \; w_P \,\right]^T \\ \mathbf{x}_i \in \mathbb{R}^{(P+1)\times 1} &= \left[\, 1 \; x_i^1 \; \cdots \; x_i^P \,\right]^T \end{aligned} \tag{1.17}$$

so, with the notation (1.16), the polynomial fitting is represented as a *linear in the data model*.

The observed data are always affected by *measurement noise*. To take account of this uncertainty, it is defined the *measurement error* or *residual* e_i, as the difference between the observed data value d_i and the value obtained considering the chosen model of fitting, both evaluated with input $\mathbf{x}_i$

$$e_i = d_i - \mathbf{w}^T \mathbf{x}_i \tag{1.18}$$

in other words, e_i represents the vertical distance between the available data pairs and the estimated fitting function.

Considering the above assumptions and definitions, the problem of polynomial regression, can be formalized by the following definition.

Definition 1.1. Given a data set related to some noisy observations, a polynomial function of pre-assigned degree $f(x_i, \mathbf{w}) = \mathbf{w}^T \mathbf{x}_i$ *a priori* assumed as data model, the problem of polynomial regression can be formalized as the estimate of the polynomial coefficients $\mathbf{w} = \left[\, w_0 \; w_1 \; \cdots \; w_P \,\right]$, with the criterion of minimizing the effect due to measurement uncertainty e_i.

1.2.3 On-line Algorithm: Naïve Least Mean Squares (LMS) Derivation

In order to introduce a simple learning algorithm, consider a simple online adaptation procedure where the recursive estimator has a form of Eqn. (1.1), i.e. starting from

a "small" values random (or all zero) configuration, the estimated results is obtained after a "sufficient" number of iterations.

In order to determine the "best" polynomial curve fitting we minimize the residual error between the regression line and the samples. So, with simple and intuitive approach, we may define a loss function as the square error over each input pair

$$J(\mathbf{w}) \triangleq e_i^2 = \left(d_i - \mathbf{w}^T \mathbf{x}_i\right)^2, \quad \text{for } i = 1,\ 2, \cdots . \tag{1.19}$$

and developing the square of the error, we get the following quadratic form

$$e^2[n] = d^2[n] - \mathbf{w}^T \mathbf{x} d[n] - \mathbf{x}^T \mathbf{w} d[n] + \mathbf{w}^T \mathbf{x}\mathbf{x}^T \mathbf{w}. \tag{1.20}$$

In order to minimize the LF (1.19) we have that for each iteration step the weights variation $\Delta\mathbf{w}$ should be proportional to the negative LF's gradient direction $\Delta\mathbf{w} \propto -\nabla J(\mathbf{w})$.

Table 1.1 LMS Algorithm for polynomial regression problems

Given $[x_i\ d_i]_{i=1}^N$;
Initialization $\mathbf{w}$, μ
for i = 1, 2, ... { *// for each observation*

$\mathbf{x}_i = [1\ x_i\ x_i^2\ \cdots]$; *// vector* $\mathbf{x}_i$ *composition*
$e_i = d_i - \mathbf{w}^T \mathbf{x}_i$; *// compute the error*
$\mathbf{w} = \mathbf{w} + \mu \cdot \mathbf{x}_i \cdot e_i$ *// LMS adaptation rule*
}

Example: Matlab code

```
. . . . .
w  = zeros(P+1,1);  % weights initial conditions (i.c.)
xx = zeros(P+1,1);  % vector containing various powers of the input signal
                    % x[n]^0 x[n]^1  x[n]^2 . . .
xx(1) = 1;          % x^0, always one
for k=1:K
    for p=1:P  % add elements x^p to xx vector for high degree polynomial
        xx(p+1) = x(k)^p;
    end
    e =  d(k) - w'*xx;  % on-line error computation
    w = w + mu*e*xx;    % LMS weights adaptation rule
end
. . . . .
```

The gradient of (1.19) can be easily determined as

$$\nabla J(\mathbf{w}) \triangleq \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = -2(d_i - \mathbf{w}^T \mathbf{x}_i)\mathbf{x}_i \tag{1.21}$$

and, for (1.18) we can write

$$\nabla J(\mathbf{w}) = -\mu e_i \mathbf{x}_i \tag{1.22}$$

where μ is a "small constant" defined as *step size* or *learning rate*. So, the recursive formula (1.1) for each adaptation step, can be written as

$$\mathbf{w}_i = \mathbf{w}_{i-1} + \mu e_i \mathbf{x}_i, \quad \text{for} \quad i = 1, \ ,2 \ \ldots \ N. \tag{1.23}$$

This algorithm is the well known *least mean squares* (LMS) that can be formulated as reported in Table 1.1

Remark 1.5. In the optimal filter theory of Wiener, the loss function to be minimized is the expectation of the squared error: $J(\mathbf{w}) = E\{|e[n]|^2\}$ and, in this case, the iterative algorithms to find the minimum of this function are said *gradient descent* learning algorithms. While in the case, as in the LMS algorithm, the loss function is directly the squares error $J(\mathbf{w}) = |e[n]|^2$ (i.e. an approximation of the "true" gradient). In this case the iterative algorithms to find the minimum are said *stochastic gradient descent* learning algorithm (see [2] for details).

1.2.4 Block Algorithm: Naïve Least Squares (LS) Derivation

In order to derive a batch or block algorithm, we write the LF in vector form. Starting form the error definition (1.18) we can write the expression of error for each pair of the training set (1.14)

$$\begin{aligned} e_1 &= d_1 - \mathbf{w}^T \mathbf{x}_1 \\ e_2 &= d_2 - \mathbf{w}^T \mathbf{x}_2 \\ &\vdots \\ e_N &= d_N - \mathbf{w}^T \mathbf{x}_N \end{aligned} \tag{1.24}$$

where for the case of polynomial regression, the vector $\mathbf{x}_i$ is defined as in Eqn. (1.17). It follows that, the previous relations can be expressed in matrix form as

$$\begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_N \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_N \end{bmatrix} - \begin{bmatrix} 1 & x_1^1 & \cdots & x_1^P \\ 1 & x_2^1 & \cdots & x_2^P \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_N^1 & \cdots & x_N^P \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_P \end{bmatrix}. \tag{1.25}$$

Defining the vectors

$$\begin{aligned} \mathbf{e} \in \mathbb{R}^{N\times 1} &= \begin{bmatrix} e_1 \ e_2 \ \cdots \ e_N \end{bmatrix}^T \\ \mathbf{d} \in \mathbb{R}^{N\times 1} &= \begin{bmatrix} d_1 \ d_2 \ \cdots \ d_N \end{bmatrix}^T \end{aligned}$$

and defining the data matrix, that contains the values of the available data and their respective powers, as

$$\mathbf{X} \in \mathbb{R}^{N\times(P+1)} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} = \begin{bmatrix} 1 & x_1^1 & \cdots & x_1^P \\ 1 & x_2^1 & \cdots & x_2^P \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_N^1 & \cdots & x_N^P \end{bmatrix} \tag{1.26}$$

the set of equations (1.24) can be written in compact form as the linear in the data model defined as

$$\mathbf{e} = \mathbf{d} - \mathbf{X}\mathbf{w} \tag{1.27}$$

The minimization criterion (1.19) can be extended to all observed data and the LF can be written as the *sum of square error* (SSE)

$$\begin{aligned} J(\mathbf{w}) &= \sum_{n=0}^{N-1} e^2[n] \\ &= \mathbf{e}^T\mathbf{e} \\ &= [\mathbf{d} - \mathbf{X}\mathbf{w}]^T [\mathbf{d} - \mathbf{X}\mathbf{w}] \end{aligned} \tag{1.28}$$

developing the product, we get the following quadratic form

$$J(\mathbf{w}) = \mathbf{d}^T\mathbf{d} - 2\mathbf{X}\mathbf{d}\mathbf{w} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} \tag{1.29}$$

For the gradient of $J(\mathbf{w})$, we have that

$$\nabla J(\mathbf{w}) \triangleq \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = -2\mathbf{X}\mathbf{d} + 2\mathbf{X}^T\mathbf{X}\mathbf{w}$$

and the minimum of the LF is when the gradient is null $\nabla J(\mathbf{w}) \to 0$; i.e. for

$$\mathbf{X}^T\mathbf{X}\mathbf{w} = \mathbf{X}\mathbf{d}. \tag{1.30}$$

The previous expression are indicated as *normal equations* of Yule-Walker, and solving for $\mathbf{w}$, we obtain the LS solution defined as

$$\mathbf{w}_{LS} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}\mathbf{d}. \tag{1.31}$$

However, due to its nature, the matrix $\mathbf{X}^T\mathbf{X}$ may be ill-conditioned. So, in order to have reliable and "more regular" solution, it prefers to solve the system of equations, by summing a diagonal matrix such as,

$$\mathbf{w}_{LS} = (\mathbf{X}^T\mathbf{X} + \delta\mathbf{I})^{-1}\mathbf{X}\mathbf{d}. \tag{1.32}$$

where δ is a 'small' constant defined as *regularization parameter*.

1.2.5 Linear Regression: Batch vs On-line Solution

For example, if it is assumed that the data model is a simple straight line (P=1), the error can be expressed as the difference between the given d_i value and the *fitting*

function evaluated at $\mathbf{x}_i$

$$e_i = d_i - (w_0 + w_1 x_i)$$

where the term e_i is the residual error. The term e_i, as indicated in Fig. 1.3, represents the vertical distance between the known data and the fitting regression line function. Moreover, in Fig. 1.4, it is reported the regression of an experiment done on real data. In particular, the experiment is related to the cost of renting an apartment in the historic center of Rome.

Example 1.2. Below is reported a simple MATLAB demo code that implements the LMS and LS algorithms, used to generate Fig. 1.4.

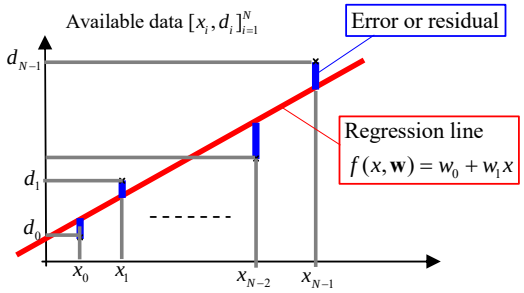

Fig. 1.3 Schematization of linear regression problem.

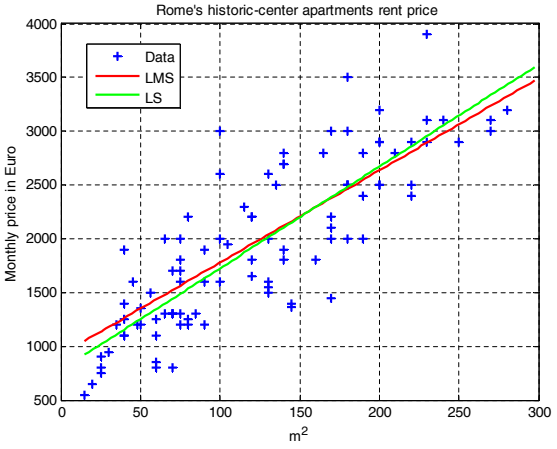

Fig. 1.4 Example: cost in Euro of renting an apartment in the historic center of Rome (data from *Porta Portese*). Line fitting determined by LMS and LS algorithms.

```
%--------------------------------------------------------------------------
%  Naive Introduction of Least Squares Method
%
% $Revision: 1.0$    $Date: 2016/10/11$
%--------------------------------------------------------------------------
clear all;  close all;
%
```

```
% ---- Data from 'PORTA PORTESE' Sept. 2015 -------------------------------
Data = [ ...
120 2200; ...
75  1700; ...
75  1600; ...
40  1900; ...
. . . . . .
];
%
% Polynomial Regression order and Learing Rate ----------------------------
P = 1;  mu = 0.2;

% internal vectors definition x,d -----------------------------------------
d = Data(:,2);
x = Data(:,1);
K = length(d);

% % Data normalization ------------------------------------------------------
max_d =  max( Data(:,2));
max_x =  max( Data(:,1));
% force zero mean (optional)
mean_d = mean(d);
mean_x = mean(x);
d = (d - mean_d) ./ max_d;
x = (x - mean_x) ./ max_x;

%-------------------------------------------------------------------------
% Simple LMS Algorithm ----------------------------------------------------
%-------------------------------------------------------------------------
w  = zeros(P+1,1);  % initial weights
xx = zeros(P+1,1);  % vector for the calculation of the dot product  w'*xx
xx(1) = 1;                % x^0, always one
for nn = 1 : 500
    for k=1:K
        for p=1:P   % add elements x^p to xx vector for high degree polynomial
            xx(p+1) = x(k)^p;
        end
        e =  d(k) - w'*xx;  % on-line error computation
        w = w + mu*e*xx;    % LMS adaptation rule
    end
end

%-------------------------------------------------------------------------
% LS solution -------------------------------------------------------------
%-------------------------------------------------------------------------
X = zeros(K,P+1); % Data Matrix definition
% Build Data matrix X
for i = 1 : K
    X(i,1) =    1;
    X(i,2) =  x(i);
    for p = 2:P    % add raw to X matrix for high degree polynomial
        X(i,p+1) = x(i)^p;
    end
end
Rxx = 0.0001*eye(P+1) + X'*X;
wLS = inv(Rxx)*X'*d;

% Print results
fprintf('Estimated polynomial coeff.s via LMS');
w
fprintf('Estimated polynomial coeff.s via LS');
wLS

% -------------------------------------------------------------------------
% Plot results
% -------------------------------------------------------------------------
%
dxx = (max(x) - min(x)) /K;
x1(1) = min(x);
```

```
xx(1) = 1;
for n=1:K-1
    for p=1:P   % add elements x^p to xx vector for high degree polynomial
        xx(p+1) = x1(n)^p;
    end
    y1(n) =   w'*xx;
    yLS(n) =  wLS'*xx;
    x1(n+1) = x1(n)  + dxx; % x-axis for plot
end
n=K; % last point
for p=1:P   % add elements x^p to xx vector for high degree polynomial
    xx(p+1) = x1(n)^p;
end
y1(n)  = w'*xx;
yLS(n) = wLS'*xx;

% De-normalize for plot ------------------------------------------------------
x   = x   .* max_x + mean_x;
x1  = x1  .* max_x + mean_x;
d   = d   .* max_d + mean_d;
y1  = y1  .* max_d + mean_d;
yLS = yLS .* max_d + mean_d;

% Scttar plot and regression curve  ------------------------------------------
figure;
hold on; grid on;
plot(x ,d,'+','LineWidth',2);
plot(x1, y1 , '-', 'color','r', 'LineWidth',2);
plot(x1, yLS,'-', 'color','g', 'LineWidth',2);
title('Rome''s historic-center apartments rent price ');
xlabel('m^2');
ylabel('Monthly price in Euro');
legend('Data','LMS','LS');
% ----------------------------------------------------------------------------
```

1.2.5.1 Data normalization

To overcome possible numerical errors is sometimes necessary to normalize the data. For example, as introduced in Example 1.2, in high-order polynomial fitting the presence of large data value ($\gg 1$) would lead to high values of the polynomial and this can be the font o possible numerical errors. On the contrary, in presence of data normalization for example in the range $[-1,\ 1]$, the high degree polynomial's coefficients may have to small influence in the determination of the regression curve. In these cases the choice of the proper normalization is decisive for the accuracy of the results.

Normalization can be done with respect to the mean and variance of the features values or with respect to the amplitude of the data. In the former, instances are normalized such that the normalized data have zero mean and unitary variance.

$$x_i' = \frac{x_i - \bar{x}_i}{\sigma_i}$$

where x_i indicates the i-th data instance, x_i' is its normalized value, $\bar{x}_i$ is the mean value, and σ_i is the standard deviation of x_i.

In the case of data normalization, instances are simply divided by the max entry value so that all feature values are restricted to a $[-1,\ 1]$ range. Moreover, if great magnitude differences exist between the individual feature values, a logarithmic

transformation, for example, can be used to reduce the dynamic range of the feature values.

1.2.6 The Adaptive Linear Combiner

The previous examples of polynomial and line regression, are based on a criterion that minimizes a certain error function, where the error is defined by Eqn.(1.18), that is the difference between the desired output d_i (the rent price of the i-th apartment) and the linear combination of the data: $e_i = d_i - \mathbf{w}^T \mathbf{x}_i$.

Considering a more 'systemistic' approach, as illustrated in the scheme of Fig. 1.5, the linear combination can be seen as the output of a system (e.g. a circuit or an algorithm) described by the relation $\mathbf{w}^T \mathbf{x}$, which represent the assumed model of the given data set. With this interpretation the vector $\mathbf{x}_i$ represents the input of such system while $y = \mathbf{w}^T \mathbf{x}$ its output.

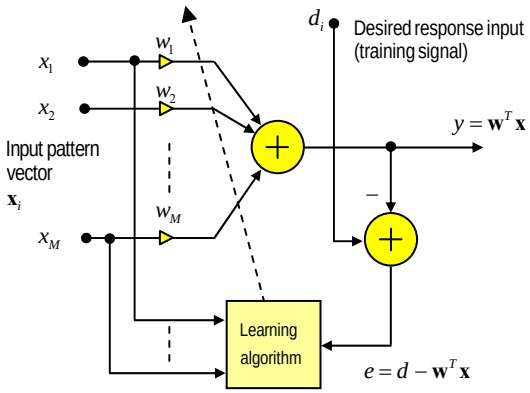

Fig. 1.5 Scheme of the Adaptive Linear Combiner.

The error-correction learning procedure can be summarized in the following steps:

1. system's output computation,
2. error computation,
3. weights update.

Introduced by Widrow in 1959 (see [5],[23] for more details) the ALC is the an adaptive structure that is used as basic element for most Machine Learning methods as the Adaptive Filter and Neural Networks.

1.3 Simple Patterns Recognition Problem by LS Methods

One of the main applications of neural networks (and other machine learning models) is to determine an estimate of an unknown classification function. In this case, at the

network's input is presented a set of data, for example a signal, image or other some vector of characteristic features, denoted as *input patterns*, associated with the object that should be classified.

The goal of a *pattern recognition system* is therefore to estimate the correct label, corresponding to a given features vector that represents the object, based on some *a priori* knowledge obtained through training phase.

The problem is very similar to that regression that has been previously addressed, except that in this case the output of the classifier (i.e. the output of the neural network), is the class of membership and then, differently from the regression problem, in this case the output value belongs to a discrete and countable set.

The aim of learning algorithm is to estimate the decision boundaries of the recognition system which are, in general, nonlinear functions. In other words, pattern recognition problem can be cast as a function approximation problem.

1.3.1 Simple Pattern Recognition by Adaptive Linear Combiner

Lets define a training set of N labeled patterns

$$[\mathbf{x}_i, d_i]_{i=1}^{N}, \qquad \mathbf{x}_i \in \mathbb{R}^M, \quad d_i = [-1, +1] \tag{1.33}$$

i.e. $\mathbf{x}_i$ is the set of the features vectors and d_i is the value of the binary class. According to Fig. 1.6, let $M = 2$, in the example using a simple *adaptive linear combiner* (ALC) [5], we want to estimate the line (the red line in the figure) that separates the two given classes defined by the features $[x_1 \; x_2]^T$.

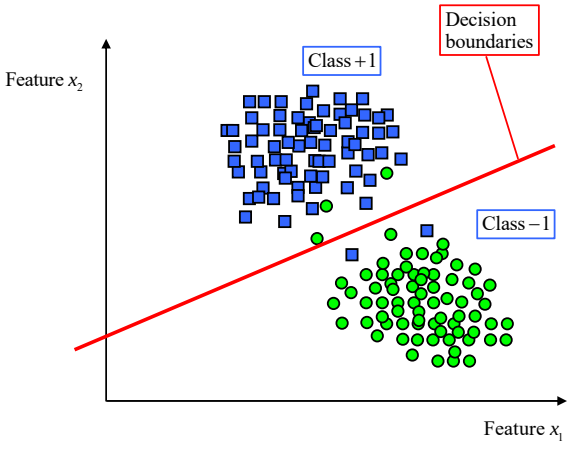

Fig. 1.6 Simple two classes pattern recognition problem. For simplicity tha classes are assumed linearly separable.

Since the output of the ALC is a real number equal to $y = w_1x_1 + w_2x_2 + \ldots + w_Mx_M$, in order to have a binary output, for example the values -1 and $+1$, it is

necessary to quantize the output value with a threshold. Thus, for the binary output we have that

$$\text{if} \quad w_1x_1 + w_2x_2 + \ldots + w_Mx_M \geq threshold, \quad \text{then} \quad y = +1, \quad \text{else} \quad y = -1.$$

For simplicity, the threshold value can be can be considered as an additional weight of the linear combiner. Denoting such weight (or bias value) as w_0, the output of this extended linear combiner can be computed as

$$\begin{aligned} \sum_{i=1}^{M} w_i x_i + w_0 \geq 0 \quad \text{then} \quad y = +1 \\ \sum_{i=1}^{M} w_i x_i + w_0 < 0 \quad \text{then} \quad y = -1. \end{aligned} \tag{1.34}$$

Defining a the function $\varphi(\cdot)$ denoted as *squashing function*, for example as

$$\varphi(s) = \begin{cases} \text{high} & s \geq 0 \\ \text{low} & s < 0 \end{cases} \tag{1.35}$$

as the hard limiter with high=1 and low=0 or low=-1, that the implements the rule (1.34), the extended ALC output can be written as

$$y = \varphi\left(\sum_{i=1}^{M} w_i x_i + w_0\right). \tag{1.36}$$

For a further simplification, defining an additional input x_0 of the ALC fixed to $x_0 = +1$, we get

$$y = \varphi\left(\sum_{i=0}^{M} w_i x_i\right) \tag{1.37}$$

that in compact form can be written as

$$y = \varphi(s) \tag{1.38}$$

where the quantity s is defined as the output of the extended linear combiner

$$s = \mathbf{w}^T\mathbf{x} \tag{1.39}$$

and the extended vectors $\mathbf{x}, \mathbf{w} \in \mathbb{R}^{(M+1)\times 1}$ are defined as

$$\begin{aligned} \mathbf{x} &= [1 \; x_1 \; x_2 \; \cdots \; x_M] \\ \mathbf{w} &= [w_0 \; w_1 \; w_2 \; \cdots \; w_M]. \end{aligned} \tag{1.40}$$

The extended ALC scheme is reported in Fig. 1.7

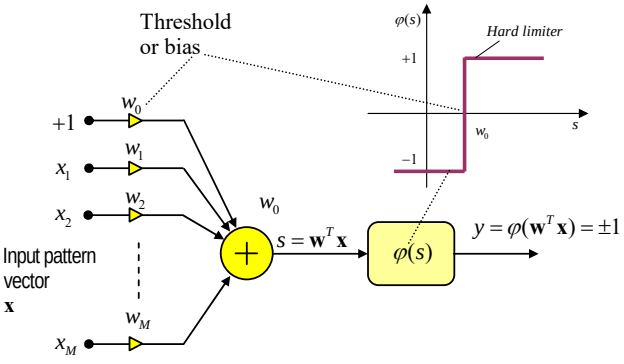

Fig. 1.7 Extended linear combiner with $M+1$ input, followed by a squashing function which ensures a binary output, with a threshold of value $-w_0$.

1.3.2 Geometrical Interpretation of EALC for Binary Classification Problems

Considering an EALC with hard limiter squashing function as Fig. 1.7 shown. For $M = 2$, it is possible to divide the input *features space*, defined by the features x_1 and x_2, in two regions. In fact, form expression (1.39) results

$$s = w_0 + x_1 w_1 + x_2 w_2 \equiv 0$$

and solving for x_2 we have the equation

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2} \tag{1.41}$$

that is a straight-line that separates the space of the features a two regions.

This geometrical interpretation can be viewed as simple classification task. If the input $[x_1 \ x_2]^T$ are such that $y = 1$, we get

$$s = w_0 + x_1 w_1 + x_2 w_2 \geq 0, \quad y = 1 \quad \Rightarrow \quad [x_1 \ x_2]^T \in \text{ class } +1$$

in the case that $y = -1$ we have

$$s = w_0 + x_1 w_1 + x_2 w_2 < 0, \quad y = -1 \quad \Rightarrow \quad [x_1 \ x_2]^T \in \text{ class } -1.$$

In Fig. 1.8 is illustrated a simple example of regions definition for EALC with two inputs, fixed weights, hard limiter squashing function and two different values of the bias w_0.

Moreover, by increasing the number of EALC inputs the dimension of the feature space increases. In Fig. 1.9 is reported an example of decision boundary plane obtained by a EALC with three inputs.

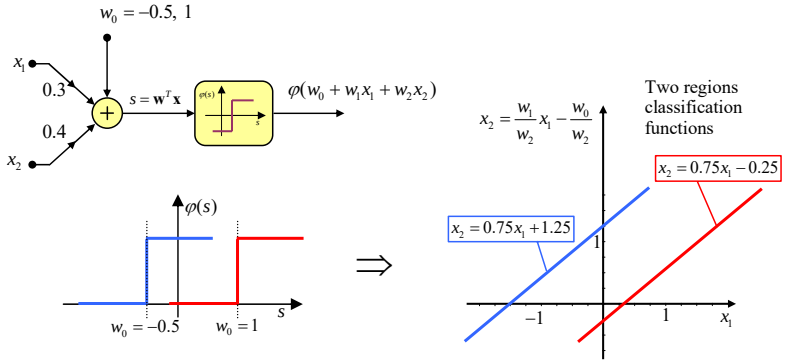

Fig. 1.8 Example of output of an EALC with fixed weights and different values of the bias. Changing bias w_0, corresponds to decision boundary translation in the plane of the features space.

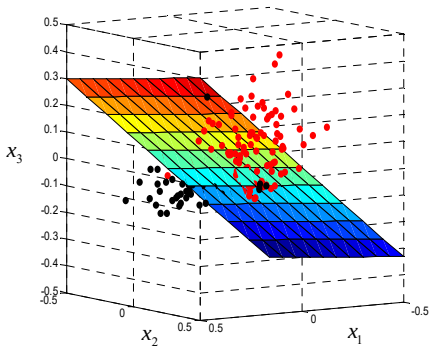

Fig. 1.9 A single three inputs EALC with hard limiter squashing function can be represented by a plane in the feature space $\{x_1,\ x_2,\ x_3\}$.

1.3.3 Learning Extended Adaptive Linear Combiner by LMS Algorithm

As observed, single EALC with M inputs (plus a bias value w_0), can divide the input space spanned by the input vector in two regions or classes that are linearly separable. In other words, we can define a M-dimension hyperplane able to divide these two classes.

The determination of the weights value can be performed by a learning phase over a predefined training set. Using a suitable learning algorithm able to estimate a 'best' (relatively to certain criteria) linear approximation of the unknown classification function.

In the experiment reported below, the training of EALC is performed by simple LMS algorithm and for the gradient descent procedures it is possible define two modality for the error calculation:

1. *Data error correction* - Use the output of the linear combiner, i.e. $e_i = d_i - \mathbf{w}^T\mathbf{x}_i$ and, subsequently the output of the linear combiner is quantized by the hard limiter as shown in the Fig. 1.10-a).
2. *Quantized error correction* - directly use the quantized output $e_i = d_i - \varphi(s)$, as illustrated in Fig. 1.10-b).

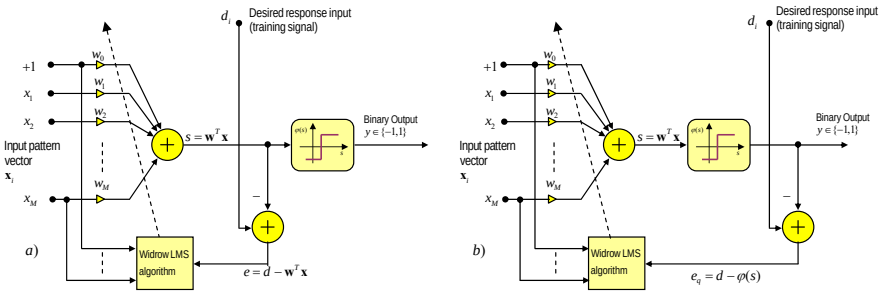

Fig. 1.10 Learning schemes of extended adaptive linear combiner: a) data error correction; b) quantized error correction.

Example 1.3. Below, it is shown a simple MATLAB program that implements an estimate of a linear classifier of two lineary separable classes. The clusters of classes ara randomly generated by a Gaussian distribution of data around a prespecified cluster center.

```
% ------------------------------------------------------------------------
% Two linearnly-separable classes problem by Extended Adaptive Linear
% Combiner  (EALC) trained with the LMS algorithm
% ------------------------------------------------------------------------
clear all; close all;
%
rng(9);
% ------------------------------------------------------------------------
% ---- Training Set Definition and generation ----------------------------
% ------------------------------------------------------------------------

x10 = [ -1.0   1.6 ];  % Cluster 1 center
x20 = [  1.5  -1.8 ];  % Cluster 2 center

% x10 = [ 8*(rand-0.5)  8*(rand-0.5) ];  % Random cluster 1 center
% x20 = [ 8*(rand-0.5)  8*(rand-0.5) ];  % Random cluster 2 center
% ---- Training Set data generation
NofSamples = 2000;
for n = 1 : NofSamples
    if rand<=0.5
       r      = 1.2*randn;
       phi    = 2*pi*rand;
       x1(n)  = x10(1) + r * cos(phi);
       x2(n)  = x10(2) + r * sin(phi);
       d(n)   = 1;  % Class + 1
       clr(n) = 'm';
    else
       r      =  1.2*randn;
       phi    =  2*pi*rand;
```

```
        x1(n)  =  x20(1) + r * cos(phi);
        x2(n)  =  x20(2) + r * sin(phi);
        d(n)   = -1;  % Class -1
        clr(n) = 'b';
    end
end

% ------------------------------------------------------------------------
% **** LMS algorithm ----------------------------------------------------
% ------------------------------------------------------------------------

mu = 1e-2;   % LMS Learning rate

% LMS 1 Quantized error correction -- Red --------------------------------
w0  = 0.2*(rand(3,1)-0.5); % Initial conditions
w  = w0;

for k=1:NofSamples
    xw = [ 1; x1(k) ; x2(k) ];
    s = w'*xw;
    if s>0, y = 1; else y = -1; end
    e =  d(k) - y ;   % on-line error computation
    w = w + mu*e*xw;  % LMS adaptation rule
end
m1 =  -w(2)/w(3);
c1 =  -w(1)/w(3);


% LMS 2 Data error correction -- Green -----------------------------------
w  = w0;
for k=1:NofSamples
    xw = [ 1; x1(k) ; x2(k) ];
    s = w'*xw;
    e = d(k) - s ;   % on-line error computation
    w = w + mu*e*xw;  % LMS adaptation rule
end
m2 =  -w(2)/w(3);
c2 =  -w(1)/w(3);
%
% Classification line
%
xmin = -5;
xmax =  5;

x  = xmin;
dx = (xmax - xmin)/NofSamples;
for i = 1 : NofSamples
  ff1(i) = m1*x + c1;
  ff2(i) = m2*x + c2;
  xx(i)  = x;
  x   = x + dx;
end

% ------------------------------------------------------------------------
% Plot results
% ------------------------------------------------------------------------
figure;
hold on; grid on; box on;
for n=1:NofSamples
   plot(x1(n), x2(n), '.','color',clr(n),'LineWidth',2);
end

ylim([ xmin xmax ]);
xlim([ xmin xmax ]);
title('Two classes problem');

plot(xx, ff1, 'color', 'r', 'LineWidth', 2);
plot(xx, ff2, 'color', 'g', 'LineWidth', 2);
xlabel('\it{x}_1');
```

```
ylabel('\it{x}_2');
% --------------------------------------------------------------------------
```

Fig. 1.11 Classification problem by EALC and LMS algorithm learning schemes. The cluster of data 'Class +1' and 'Class −1' are supposed linearly separable. The red and the green lines represent two different estimation of the classification functions. What was the 'best' estimate?

1.3.4 Classification of Non-Linearly Separable Patterns

The problem of classification of non-linearly separable pattern can be very complex and in these introductive notes, the problem is treated only for the purpose of example. As illustrated in 1.12, we consider the problem of the estimate of the classification function, or decision boundary.

Fig. 1.13 shows two possible simple schematization of classification systems for not linearly separable patterns. The first is formed by (usually) fixed nonlinear preprocessing network connected to adaptive elements (for example EALC), while in the second, on the left part of the figure, the the adaptive weigths are located before the non-linear function (for example as in multi-layer feedforward neural networks).

The architecture with the adaptive part that is downstream of the non-linearity (left part of Fig. 1.13), is typically appealed considering the type of the adopted not linear expansion. In the case that the non linear expansion is constituted by simple non-linear functions as, for example, trigonometric functions, polynomials, etc. the structure is usually referred to as *functional link network*. Moreover, if the nonlinearities represent a certain "structured" information, for example kernel functions in some way connected with the structure of the input data, in the literature there are

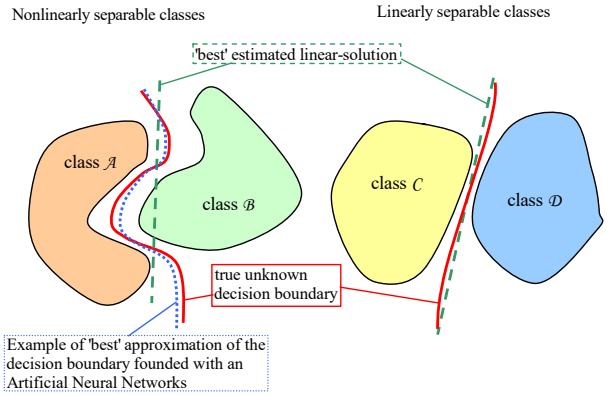

Fig. 1.12 Nonlinear vs. linear two classes separation.

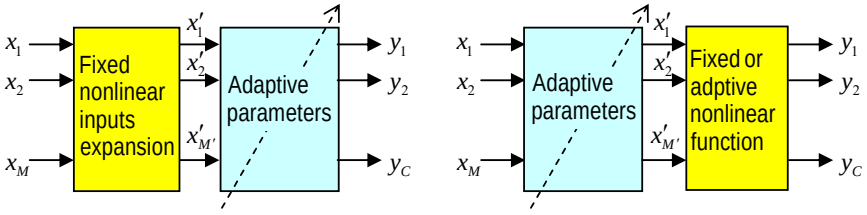

Fig. 1.13 Simple schemes for nonlinear classification problem: a) realized with nonlinear fixed expansion of the input data; b) neuron-like structure which realizing a nonlinear combination of the input data.

many different types of such networks as, for example, radial basis functions (RBF), kernel machines, etc.

1.3.4.1 The Cover Theorem

The expansion of a features vector $\mathbf{x}$ with a nonlinear function $\varphi : \mathbb{R}^{M_0} \to \mathbb{R}^M$ with $M > M_0$ increases the probability that the new feature vector $\mathbf{x}' = \varphi(\mathbf{x})$ having classes that are linearly separable. That is, in fact, the Cover Theorem [16].

Theorem 1.1. *The probability that classes are linearly separable increases when the features are nonlinearly mapped to a higher dimensional feature space. In other words, a complex pattern classification problem, cast in non-linearly expanded high-dimensional space, is more likely to be linearly separable than a low-dimensional space, provided that the space is not densely populated.*

1.3.4.2 Example of EALC with Non-Linearly Expanded Input

According to with the Cover's Theorem 1.1, a simple network for the classification of two not linearly separable patterns, consists in a simple EALC to the input of which is placed a block functional expansion as shown in Fig. 1.14.

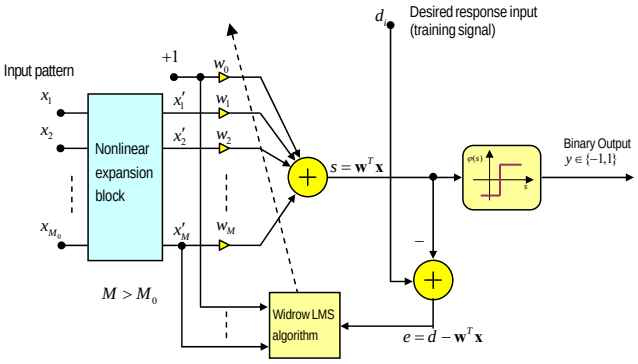

Fig. 1.14 Extended adaptive linear combiner with functional link expansion.

Example 1.4. As a first example of recognition of non-linearly separable classes, we consider the classification "toy problem" so-called *two moons* shown in Figure1.15.

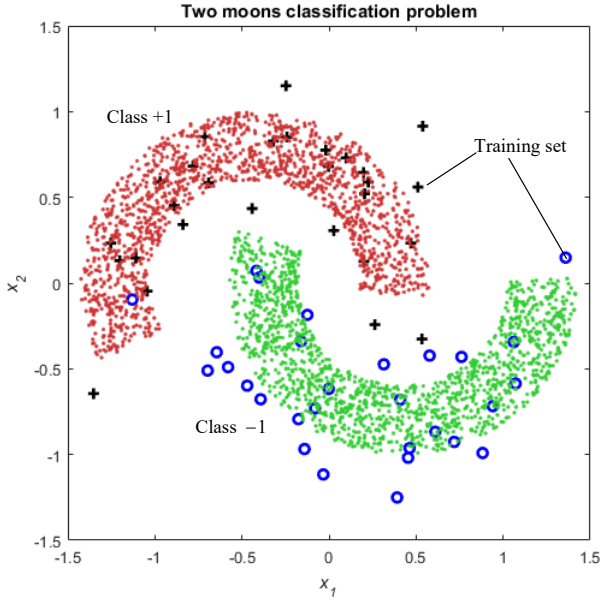

Fig. 1.15 Two moons problem. For the classification of the correct class only few input patterns are available. The training set consists in 60 noisy input/output pairs.

For simplicity hor $M_0 = 2$, considering the geometric interpretation of the EALC in §1.3.2, the input functional expansion is choice as a polynomial defined as such that we have

$$s = w_0 + w_1x_1 + w_2x_1^2 + \cdots + w_Mx_1^M + w_{M+1}x_2 \geq 0, \Rightarrow \text{ class } +1$$
$$s = w_0 + w_1x_1 + w_2x_1^2 + \cdots + w_Mx_1^M + w_{M+1}x_2 < 0, \Rightarrow \text{ class } -1.$$

With the previous positions, the estimated classification function is a polynomial defines as

$$x_2 = c_0 + c_1 x_1 + c_2 x_1^2 + \cdots + c_M x_1^M$$

where

$$\begin{aligned} c_0 &= -w_0/w_{M+1} \\ c_2 &= -w_1/w_{M+1} \\ &\cdots \\ c_M &= -w_M/w_{M+1} \end{aligned}$$

```
% ------------------------------------------------------------------------
% Two non-linearnly separable classes problem by Extended Adaptive Linear
% Combiner  (EALC) trained with the LMS algorithm
% ------------------------------------------------------------------------
clear all; close all;
%
rng(9);
% ------------------------------------------------------------------------
% ---- Training Set Definition and generation ----------------------------
% ------------------------------------------------------------------------
nSamples = 60;
data = two_moons(nSamples);
sigma = 0.25; % std dev of added Gaussina noise
x1 = data(:,1)  +  sigma*randn(nSamples,1);
x2 = data(:,2)  +  sigma*randn(nSamples,1);
d =  data(:,3);
% ------------------------------------------------------------------------
% ---- Set LS/LMS algorithms  parameters ---------------------------------
% ------------------------------------------------------------------------
P  = 7;             % Approximating polynomial order
nIteration = 1000;  % for LMS algorithm
mu = 0.001;         % LMS Learning rate
w0 = zeros(P+2,1);  % Initial conditions
w1 = zeros(P+2,1);
w2 = zeros(P+2,1);
xw = zeros(P+2,1);
xw(1) = 1;
%------------------------------------------------------------------------
% LS solution -------------------------------------------------------------
%------------------------------------------------------------------------
X = zeros(nSamples, P+2); % Data Matrix definition
% Build Data matrix X
for i = 1 : nSamples
    X(i,1) = 1;
    for p = 1 : P    % add raw to X matrix for high degree polynomial
        X(i, p+1) = x1(i)^p;
    end
    X(i, P+2) = x2(i);
end
Rxx  = eye(P+2) + X'*X;
w1   = inv(Rxx)*X'*d;
%
% ------------------------------------------------------------------------
% LMS Solution
% ------------------------------------------------------------------------
for nn = 1 : nIteration
    for k=1:nSamples
        for p = 1 : P   % add elements x^p to xx vector for high degree polynomial
            xw(p+1) = x1(k)^p;
        end
        xw(P+2) = x2(k);
        s = w2'*xw;
        e =  d(k) - s;   % on-line error computation
```

```
        w2 = w2 + mu*e*xw;  % Leaky-LMS adaptation rule
    end
end
% ------------------------------------------------------------------------
% Approximating polynomial
% ------------------------------------------------------------------------
for k=1:P+1
    cw1(k) = -w1(k)/w1(P+2);  % LS
    cw2(k) = -w2(k)/w2(P+2);  % LMS
end
%
% ------------------------------------------------------------------------
% Classification function
% ------------------------------------------------------------------------
xmin = -1.5;
xmax =  1.5;
x  = xmin;
dx = (xmax - xmin)/(nSamples-1);
xw = zeros(P+1,1);
xw(1) = 1;
for i = 1 : nSamples
    for p = 1 : P   % add elements x^p to xx vector for high degree polynomial
       xw(p+1) = x^p;
    end
  ff1(i) = cw1*xw;  % LS
  ff2(i) = cw2*xw;  % LMS
  xx(i)  = x;
  x  = x + dx;
end
% ------------------------------------------------------------------------
% Plot results
% ------------------------------------------------------------------------
figure;
hold on;  box on;

for n=1:nSamples
    if d(n)==1
        plot(x1(n), x2(n), '+', 'color','k','LineWidth',2);
    else
        plot(x1(n), x2(n), 'o', 'color','b','LineWidth',2);
    end
end
ylim([ xmin  xmax ]);
xlim([ xmin  xmax ]);
title('Two moons classification problem');
plot(xx, ff1, 'color', 'r', 'LineWidth', 2); % LS
plot(xx, ff2, 'color', 'g', 'LineWidth', 2); % LMS
xlabel('\it{x}_1');
ylabel('\it{x}_2');
% ------------------------------------------------------------------------
```

In Fig. 1.16 are reported the results of two experiments considering a different number of training patterns, and a different value of noise superimposed on the training set. Below is reported a MATLAB procedure for generation of the two moons patterns used in the above experiments.

```
function data = two_moons(N, r1, r2, r3)
if nargin < 1
    N = 3000;
end
if mod(N,4) ~= 0
    N = round(N/4) * 4;
end
if nargin < 2
```

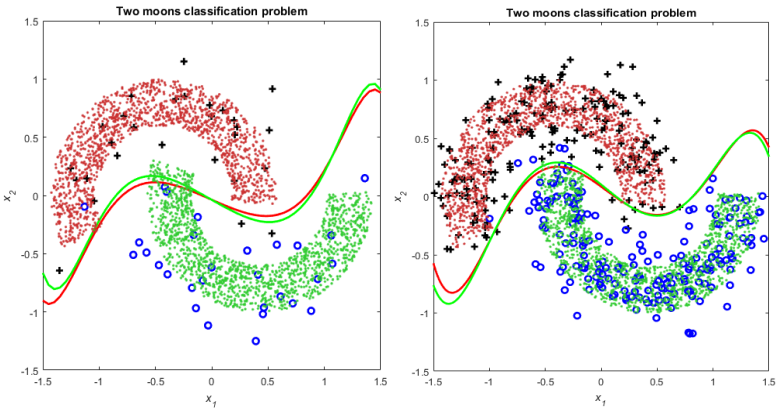

Fig. 1.16 Two moons problem. Non linear regression results. the red line is relative to the LS algorithm, while the green to LMS (see the MATLAB code below). Left) 60 training set pairs. Added noise $\sigma_\eta = 0.25$; right) 400 training set pairs. Added noise $\sigma_\eta = 0.15$.

```
    r1 = 0.3;
end
if nargin < 3
    r2 = 0.6;
end
if nargin < 4
    r3 = 1;
end
N1 = N/2;
N2 = N/2;
a = 0.85; % Def a=1
d = r3 - r2; %  half-moon width
phi2 =  a*pi + (1/a)*rand(N1,1) * pi;
R2 = (rand(N2,1));
crescent1 = [cos(phi2) .* (r2 + R2 * d) - r3/2 + r1/4   -sin(phi2) .* (r2 + R2 * d
    )    ones(N2,1)];
%
a = 0.9; % Def a=1
phi2 =  a*pi + (1/a)*rand(N2,1) * pi;
% R2 = sqrt(rand(N2,1));
 R2 = (rand(N2,1));
crescent2 = [cos(phi2) .* (r2 + R2 * d)  + r3/2 - r1/4  sin(phi2) .* (r2 + R2 * d)
        -ones(N2,1)];
data = [crescent1; crescent2];
```

1.3.4.3 Underfitting and overfitting

Given a set of data, denoted as training set, related to an unknown function $f_0(\mathbf{x})$, the task of regression (and also classification) is to find an estimate of a set of parameters $\mathbf{w}_{opt}$ of given model $f(\mathbf{w},\mathbf{x})$ (or hypothesis f), such that $||f_0(\mathbf{x}) - f(\mathbf{x},\mathbf{w})|| < \varepsilon$.

In general, there is not a general method for determining the values of the parameters $\mathbf{w}_{opt}$ that provide the "best" data interpolation. To this end, it is necessary to choose:

1. the type of model (e.g. polynomial, exponential, linear, neural networks, specific white-box, etc.),
2. the class of numerical optimization algorithms, such that

$$\mathbf{w}_{opt} \therefore \arg\min_{\mathbf{w}\in\mathbb{R}} \|f_0(\mathbf{x}) - f(\mathbf{x},\mathbf{w})\|$$

where $\|\cdot\|$ indicate some measure of distance, which starting from the initial values of $\mathbf{w}$, leads to points $\mathbf{w}_{opt}$ considered optimal.

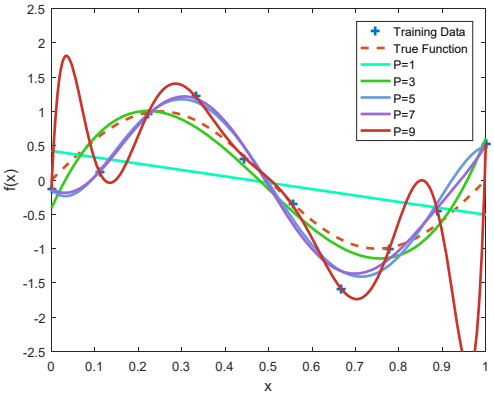

Fig. 1.17 Polynomial regression problem for different polynomial order.

One of the main problems of such *data driven* procedure are the phenomena of *underfitting* and *overfitting*.

We have overfitting if the choice of the model and/or its parameters is not appropriate, and approximating function is not the correct one but tends to perfectly fit the training data, not taking into account that these are affected by noise.

The underfitting phenomenon may occur when the model is inadequate to represent the problem, for example the order of the regression polynomial is too low or when the model is not well trained.

For example, Fig. 1.17 shows the curve regression problem, for different polynomial order given 10 training samples of function $f_0(x) = \sin(2\pi x) + noise$. For high order polynomial the the model $f(\mathbf{w},\mathbf{x})$ exactly fit the training set samples and we have *overfitting*. On the contrary, for to low order polynomial we have the *underfitting* problem.

Finally, note that if the available training set is large, relatively to the model order of the problem being treated, the overfitting can be mitigated.

1.3.5 Probabilistic Interpretation of LS methods

In the LS methods, the distribution of the RV vector $\mathbf{w}$ is unknown: the LS can be seen as a classical estimation theory problem. So, the vector $\mathbf{w}$ is cosidered as a vector of deterministic unknown parameters.

For a probabilistic interpretation of LS methods, assume that the error e_i are identical and indipendent distributed (IID) according to a zero-mean Gaussian (or Normal) distributed with variance σ_e^2 and usually indicated as $e_i \in \mathcal{N}(0, \sigma^2)$. Thus, according with the normal distribution, the pdf is

$$f(e_i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{e_i^2}{2\sigma^2}\right). \tag{1.42}$$

By substituting in the previous expression the error (1.18) ($e_i = d_i - \mathbf{w}^T\mathbf{x}$), we get

$$f(d_i|\mathbf{x};\mathbf{w}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(d_i - \mathbf{w}^T\mathbf{x})^2}{2\sigma^2}\right) \tag{1.43}$$

where the notation $f(d_i|\mathbf{x};\mathbf{w})$ means that the distribution of d_i depends on $\mathbf{x}$ (i.e. conditioned by $\mathbf{x}$), and where the symbol ";" means that the distribution of d_i is also parametrized by $\mathbf{w}$, i.e. the vector $\mathbf{w}$ is considered as a vector of unknown parameters.

Moreover, considering the vector formulation of LS, given a data matrix $\mathbf{X}$, that contains the row $\mathbf{x}_i^T$ and $\mathbf{w}$, the pdf of the data can be written as $f(\mathbf{d}|\mathbf{X};\mathbf{w})$ and the expression (1.43), can be written as

$$f(\mathbf{d}|\mathbf{X}\mathbf{w}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(\mathbf{d} - \mathbf{X}\mathbf{w})^2}{2\sigma^2}\right). \tag{1.44}$$

In regression and classification problems, the interest is in the estimation of the parameters $\mathbf{w}$ for which $f(\mathbf{d}|\mathbf{X};\mathbf{w})$ is the *likelihood function* (see §1.4.1, Eqn. (1.48)). In general, to highlight the dependence on parameters $\mathbf{w}$, the likelihood function it is written as $L(\mathbf{w};\mathbf{X})$ or by its logarithm $\mathcal{L}(\mathbf{w};\mathbf{X}) = \ln L(\mathbf{w};\mathbf{X})$ or, sometime, simply as $\mathcal{L}(\mathbf{w})$.

White previous position the likelihood function can be written as

$$\mathcal{L}(\mathbf{w}) = \frac{\ln\left(\sqrt{2\pi}\sigma\right)}{2\sigma^2} + (\mathbf{d} - \mathbf{X}\mathbf{w})^2. \tag{1.45}$$

Note that, although $\mathbf{w}$ is a deterministic parameter, the likelihood function $L(\mathbf{w})$ or $\mathcal{L}(\mathbf{w})$ has stochastic nature and is considered a RV. In this case, if the estimate solution exists, it can be founded as unique solution of the equation that determines the *maximum of the likelihood* equation that is defined as

$$\mathbf{w}_{ML} \triangleq \{\mathbf{w} \therefore \nabla\mathcal{L}(\mathbf{w};\mathbf{X}) \to \mathbf{0}\} \tag{1.46}$$

such solution is defined as *maximum likelihood estimate* (MLE). As the term $(\ln\sqrt{2\pi}\sigma)/(2\sigma^2)$ is constant, the maximization of $\mathcal{L}(\mathbf{w})$ is equivalent to maximize the term $(\mathbf{d} - \mathbf{X}\mathbf{w})^2$. The gradient of the likelihood function is

$$\nabla\mathcal{L}(\mathbf{w}) = \frac{\partial\mathcal{L}(\mathbf{w})}{\partial\mathbf{w}} = -2\mathbf{X}\mathbf{d} + 2\mathbf{X}^T\mathbf{X}\mathbf{w}$$

and the maximum is for $\nabla\mathcal{L}(\mathbf{w}) \to 0$; i.e. for

$$\mathbf{X}^T\mathbf{X}\mathbf{w} = \mathbf{X}\mathbf{d}$$

and we note that the previous expression is exactly equal to the Yule-Walker's normal equations (see Eqn. (1.30)), whit solution

$$\mathbf{w}_{ML} \equiv \mathbf{w}_{LS} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}\mathbf{d}.$$

Thus, from a statistical point of view, the LS solution is a maximum likelihood estimation solution.

1.4 Classical vs Bayesian Estimation Theory

The regression and classification with LS family methods, can be introduced considering a probabilistic approach. In particular the problem can be formalized in accordance with the *Estimation Theory* (ET) [7], [8].

We can make the following distinction within the estimation theory:

Classical estimation theory – In *classical estiamation theory*, the vector $\mathbf{w}$ is a RV vector with unknown distribution, none assumption on the behavior of the data is carried on. In this case the vector $\mathbf{w}$ is considered as a simple unknown term. Moreover, the formalism $f(\mathbf{x};\mathbf{w})$ indicates that $\mathbf{w}$ represents a vector of deterministic unknown parameters, i.e. simply indicates that the distribution of the data $\mathbf{x}$ depend to $\mathbf{w}$ as a parametric function.

Bayesian estimation theory – In *Bayesian estimation theory* $\mathbf{w}$ is as a RV vector characterized by its pdf $f_{\mathbf{w}}(\mathbf{w})$, and called as *a priori* pdf or simply *prior*, which contains all the known (or believed) information. The quantity to be estimated is interpreted as a realization of the RV $\mathbf{w}$.

The Bayesian and classical estimation philosophies determine two families of algorithms known as:

- maximum a posteriori estimation (MAP)
- maximum likelihood estimation (MLE)

1.4.1 Maximum a Posteriori Estimation (MAP)

In the *maximum a posteriori estimation* (MAP), the parameter $\mathbf{w}$ is characterized by the *prior* $f_{\mathbf{w}}(\mathbf{w})$ that is determined from knowledge known before the measure of data $\mathbf{d}$ in the absence of other information. The new knowledge obtained them from the measure of $\mathbf{x}$, determines a change in the pdf of $\mathbf{w}$ which is, therefore, *conditioned* by the new available measure. The new pdf of $\mathbf{w}$ depends from the new $\mathbf{x}$ and indicated as $f(\mathbf{w}|\mathbf{x})$, i.e. defined as conditioned pdf of $\mathbf{w}$ by measures $\mathbf{x}$ and, for this reason, is called as *a posteriori pdf* of $\mathbf{w}$ respect $\mathbf{x}$.

The estimation process is described by the *joint pdf*, through the well known Bayes' formula, defined as

$$f(\mathbf{x},\mathbf{w}) = f(\mathbf{x}|\mathbf{w})f(\mathbf{w}) = f(\mathbf{w}|\mathbf{x})f(\mathbf{x}) \tag{1.47}$$

where $f(\mathbf{x}, \mathbf{w})$ is the joint pdf, the terms $f(\mathbf{x}|\mathbf{w})$ is the *conditional pdf* that represents the knowledge carried from the data $\mathbf{x}$ conditioned by $\mathbf{w}$.

Now, if we are interested in the estimation of $\mathbf{w}$ in the presence of information provided by the new available data $\mathbf{x}$, we can write

$$f(\mathbf{w}|\mathbf{x}) = \frac{f(\mathbf{x}|\mathbf{w})f(\mathbf{w})}{f(\mathbf{x})} \tag{1.48}$$

and the Bayes' theorem plays a very important significance. In fact, the previous formula allows us to evaluate the uncertainty in $\mathbf{w}$ *after* the observation $\mathbf{x}$ and for this reason represents *a posteriori* probability.

The term on the right hand $f(\mathbf{x}|\mathbf{w})$, evaluated for the observed data $\mathbf{x}$, is a function of parameters $\mathbf{w}$ and is denoted as *likelihood function*. So, from previous definitions, from a qualitative point of view, Bayes' theorem asserts that

$$\text{posterior} \propto \text{likelihood} \times \text{prior}. \tag{1.49}$$

The denominator of Eqn. (1.48), which does not depend on $\mathbf{w}$, is a normalization constant allowing the left-hand is a valid pdf.

Definition 1.2. Assuming that a prior distribution $f_{\mathbf{w}}(\mathbf{w})$ exists, the MAP estimate consists in determining the maximum of the *a posteriori* pdf, formally:

$$\mathbf{w}_{MAP} = \arg\max_{\mathbf{w}\in\mathbb{R}^M} \{f(\mathbf{w}|\mathbf{x})\}. \tag{1.50}$$

This allows us to treat $\mathbf{w}$ as a random variable as in Bayesian statistics, and from the Bayes theorem the *a posteriori* distribution of $\mathbf{w}$ can be expressed as:

$$f(\mathbf{w}|\mathbf{x}) = \frac{f(\mathbf{x}|\mathbf{w})f(\mathbf{w})}{f(\mathbf{x})} \tag{1.51}$$

where the denominator of the previous expression, that can be written as

$$f(\mathbf{x}) = \int_w f(\mathbf{x}|\mathbf{w}) d\mathbf{w}$$

is the the *a posteriori* distribution of $\mathbf{x}$ respect $\mathbf{w}$, i.e. the so-called *partition function*, does not depend on $\mathbf{w}$.

It follows that the MAP estimator can be expresses as

$$\begin{aligned} \mathbf{w}_{MAP} &= \arg\max_{\mathbf{w}\in\mathbb{R}^M} \{f(\mathbf{w}|\mathbf{x})\} \\ &= \arg\max_{\mathbf{w}\in\mathbb{R}^M} \left\{ \frac{f(\mathbf{x}|\mathbf{w})f(\mathbf{w})}{f(\mathbf{x})} \right\} \end{aligned} \tag{1.52}$$

or

$$\mathbf{w}_{MAP} \triangleq \left\{ \mathbf{w} \therefore \nabla \left[\frac{f(\mathbf{x}|\mathbf{w})f(\mathbf{w})}{f(\mathbf{x})} \right] = \mathbf{0} \right\}. \tag{1.53}$$

Since the logarithm is a monotonically increasing function, the max value found with (1.53) is the same if we consider the logarithm of the gradient argument. In fact, considering the logarithm of both sides of (1.51) we can write

$$\ln f(\mathbf{w}|\mathbf{x}) = \ln f(\mathbf{x}|\mathbf{w}) + \ln f(\mathbf{w}) - \ln f(\mathbf{x})$$

thus, the procedure for the MAP estimate is

$$\frac{\partial}{\partial \mathbf{w}} (\ln f(\mathbf{x}|\mathbf{w}) + \ln f(\mathbf{w}) - \ln f(\mathbf{x})) = \mathbf{0} \tag{1.54}$$

and, since $\ln f(\mathbf{x})$ do not depend on $\mathbf{w}$, we can write

$$\mathbf{w}_{MAP} \triangleq \left\{ \mathbf{w} \therefore \frac{\partial}{\partial \mathbf{w}} (\ln f(\mathbf{x}|\mathbf{w}) + \ln f(\mathbf{w})) = \mathbf{0} \right\}. \tag{1.55}$$

Finally, note that is possible to determine the MAP solution equivalently through (1.53) or (1.55).

1.4.2 Maximum Likelihood Estimation

Considering a fixed set of data $\mathbf{x}$, and underlying statistical model, the method of *maximum likelihood* selects the set of values of the model parameters $\mathbf{w}$, considered as a simple deterministic unknown, that maximizes the so called *likelihood function* indicated as $L(\mathbf{w};\mathbf{x})$ or its logarithm $\mathcal{L}(\mathbf{w};\mathbf{x}) = \ln L(\mathbf{w};\mathbf{x})$.

1.4.2.1 Maximum Likelihood Principles

Suppose we have a IID random sample $\mathbf{x} = [x_1, x_2, ..., x_N]$, of N observations, coming from a pdf which depends on some unknown parameter $\mathbf{w}_0 \in \mathbb{R}^{M\times 1}$ such that $f_0(x_1, x_2, ..., x_N) = f(x_1, x_2, ..., x_N; \mathbf{w}_0)$. Our primary goal here will be to find an estimator such the $\hat{\mathbf{w}} \sim \mathbf{w}_0$.

Suppose that the parametric pdf family for the given iid sample can be written as

$$f(\mathbf{x};\mathbf{w}) = f(x_1;\mathbf{w}) \cdot f(x_2;\mathbf{w}) \cdot \ldots \cdot f(x_N;\mathbf{w}). \tag{1.56}$$

In order to define the method of maximum likelihood we consider a new function called *likelihood*, defined as

$$L(\mathbf{w};\mathbf{x}) = \prod_{i=1}^{N} f(x_i;\mathbf{w}) \tag{1.57}$$

where the observed values $x_1, x_2, ..., x_N$, are considered fixed parameters of this function, while $\mathbf{w}$ is a function's variable allowed to vary freely.

In practice, in order to avoid multiplications, it is often more convenient to work with the logarithm of the likelihood function or *log-likelihood* defined as:

$$\mathcal{L}(\mathbf{w};\mathbf{x}) = \ln L(\mathbf{w};\mathbf{x}) = \sum_{i=1}^{N} \ln f(x_i;\mathbf{w}) \tag{1.58}$$

or considering the estimated *average log-likelihood* defined as

$$\hat{\mathcal{L}}(\mathbf{w};\mathbf{x}) = \frac{1}{N} \ln \mathcal{L}(\mathbf{w};\mathbf{x}). \tag{1.59}$$

Note that, although $\mathbf{w}$ is a deterministic parameter, the likelihood function $L(\mathbf{w};\mathbf{x})$ or $\mathcal{L}(\mathbf{w};\mathbf{x})$ has stochastic nature and is considered a RV. In this case, if the estimate solution exists, it can be founded as unique solution of the equation that determines the maximum of the *likelihood equation* that is defined as

$$\mathbf{w}_{ML} \triangleq \{\mathbf{w} \therefore \nabla \mathcal{L}(\mathbf{w};\mathbf{x}) = \mathbf{0}\} \tag{1.60}$$

such solution is defined as *maximum likelihood estimate* (MLE).

In other words, the ML methods search for the *most likely* value of $\mathbf{w}$. Namely, research within the space $\mathbf{w}$ of all possible parameters $\mathbf{w}$ values, the value of the parameter that maximizes the probability that $\mathbf{w}_{ML}$ is the more plausible sample. From an optimization point of view we have

$$\mathbf{w}_{ML} = \arg\max_{\mathbf{w}\in\mathbb{R}^M} \{\mathcal{L}(\mathbf{w};\mathbf{x})\}. \tag{1.61}$$

1.5 Probabilistic View of Classification

The classification task, as seen in the previous examples, is similar to the regression problme except for the fact that the output takes discrete values, for example, 0 and 1 (or -1 and 1).

Sometimes, instead of having the value of the class, for example considering only two classes $y \in \{0,1\}$, we are interested in knowing the probability (or better its estimated) that the input pattern belongs to that class. In these cases, we have that the output of the classifier, i.e. the hypotheses $f(\mathbf{x},\mathbf{w})$ that here can written as $\varphi(\mathbf{w}^T\mathbf{x})$, is defined in the continuous interval $y \in [0,1]$.

1.5.1 EALC with Smooth Squashing Function: the Formal Neuron

A very simple and natural choice to determine this probability is to consider a smooth squashing function defined as

$$\varphi(s) = \left(\frac{1}{1+e^{-s}}\right) \tag{1.62}$$

that is the well known *sigmoid* or logistic *function*. In fact, it is easy to show, see Fig. 1.18, that the previous expression is equivalent to

$$\varphi(s) = \lim_{\alpha \to \infty} \left(\frac{1}{1 + e^{-\alpha s}} \right) \tag{1.63}$$

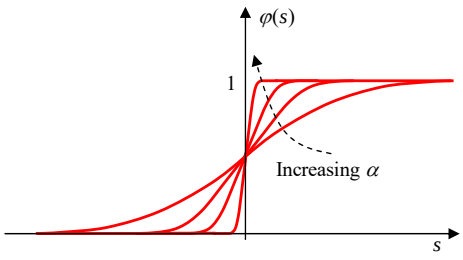

Fig. 1.18 Sigmoid or logistic function. As $\alpha \to \infty$ the sigmoid function bceome an hard limiter.

The sigmoid function is continuous and differentiable and, also, exhibits smoothness and the desired asymptotic properties. It is a continuous approximation of the step function.

The derivative of Eqn. (1.62) is

$$\begin{aligned} \frac{d\varphi(s)}{ds} &= \frac{e^{-s}}{(1+e^{-s})^2} \\ &= \frac{1}{(1+e^{-s})} \cdot \left(1 - \frac{1}{(1+e^{-s})}\right) \\ &= \varphi(s)[1-\varphi(s)]. \end{aligned} \tag{1.64}$$

The resulting EALC structure illustrated in Fig. 1.19 is defined as the formal neuron model of McCulloch and Pitts (MP-neuron) [10], which forms the basis for designing artificial neural networks (ANN)s. Here we identify three basic elements of the neuronal model:

- a set of synapses or connecting links, each of which is characterized by a synaptic weight o weight value;
- an adder for summing the input signals, weighted by the respective synapses of the neuron: the operations described here, sometime called activation potential, constitute a simple linear combiner;
- an *activation function* (AF) or squashing function for limiting the amplitude of the output of a neuron.

The continuous nature of activation function permits the calculus to be used to derive learning rules for perceptrons and multilayer perceptrons (MLP)s [9] (Rumelhart, Hinton, Williams, 1986).

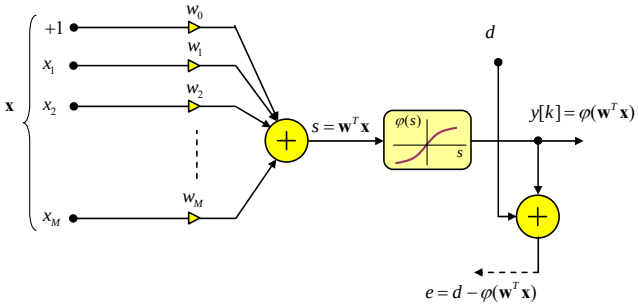

Fig. 1.19 The McCulloch and Pitts's neuron model. The parameters $\mathbf{w}$ are denoted as *synapses* or *synaptic weight*, while the squashing function is denoted as *activation function* (AF).

1.5.2 The Generalized Delta Rule

The Widrow's LMS is the basis algorithm for developing the so called gradient-based error correction learning rules which represents the most used learning methods in ANNs. The LMS, in fact, can be easily extended for EALC in presence of differentiable activation function as the MP-neuron architecture in Fig. 1.19.

Considering the adaptation rule of the form

$$\mathbf{w}_{new} = \mathbf{w}_{old} + \tfrac{1}{2}\mu[-\nabla J(\mathbf{w})] \tag{1.65}$$

where, as in the LMS and the CF $J(\mathbf{w}) = e^2$, and the term $\frac{1}{2}$ is inserted only for formal simplification.

Let d the desired output, *a priori* error is defined as $e = d - \varphi(s)$ where $s = \mathbf{w}^T\mathbf{x}$. Taking into account the presence of nonlinear function $\varphi(s)$, the gradient $\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}$ using the chain derivative rule, can be rewritten as

$$\begin{aligned}\nabla J(\mathbf{w}) &= \frac{\partial J(\mathbf{w})}{\partial e}\cdot\frac{\partial e}{\partial s}\cdot\frac{\partial s}{\partial \mathbf{w}} \\ &= \frac{\partial e^2}{\partial e}\cdot\frac{\partial [d-\varphi(s)]}{\partial s}\cdot\frac{\partial \mathbf{w}^T\mathbf{x}}{\partial \mathbf{x}} \\ &= -2e\cdot\frac{\partial \varphi(s)}{\partial s}\cdot\mathbf{x}\end{aligned}$$

and for (1.64), for the k-th iteration, let $y[k] = \varphi(s)$, we can write

$$\nabla J(\mathbf{w}) = -2e\cdot y[k](1-y[k])\cdot\mathbf{x} \tag{1.66}$$

Thus, substituting the previous expression in (1.65), the *generalized delta-rule* (GDR) for MP-neuron, for the k-th adaptation step, can be written as

$$\begin{aligned}\mathbf{w}_k &= \mathbf{w}_{k-1} + \mu e[k]\cdot y[k](1-y[k])\cdot\mathbf{x}_k \\ &= \mathbf{w}_{k-1} + \mu e[k]\cdot\delta_k\cdot\mathbf{x}_k\end{aligned} \tag{1.67}$$

where the quantity "delta" is defined as $\delta_k = e[k]\cdot y[k](1-y[k])$ from which the name generalized delta-rule.

The resulting structure, considering also the adaptation scheme, is shown in Fig. 1.20

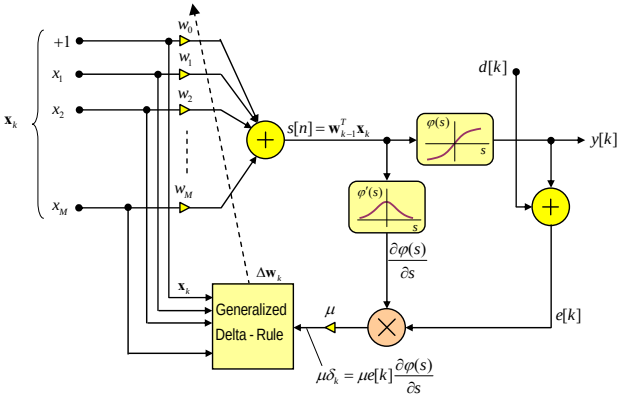

Fig. 1.20 The generalized delta-rule (GDR) for the McCulloch and Pitts's neuron model at the k-th adaptation step. The squashing function is the neuron activation function.

Remark 1.6. In some classification and regression problems, you must have a positive and negative output. In this case the activation function, can be defined as

$$\varphi(s) = \left(\frac{2}{1+e^{-s}} \right) - 1 \tag{1.68}$$

while (see also Exercise 1.9), its derivative as

$$\varphi'(s) = 1 - \varphi^2(s) \tag{1.69}$$

The Fig. 1.21 shows the trends of the unipolar and bipolar activation functions along with their derivatives.

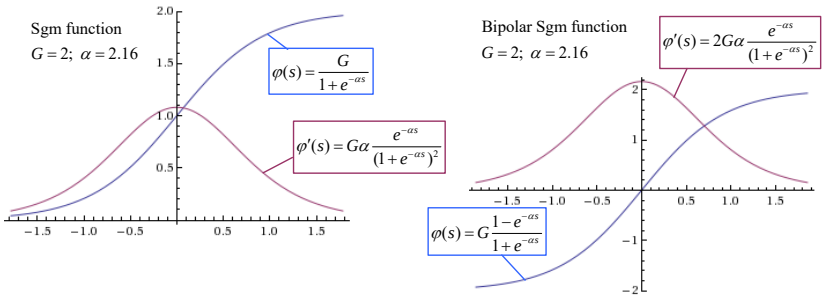

Fig. 1.21 Example of unsigned and signed sigmoid functions considering a gain G and slope α, and they derivative for $G = 2$ and $\alpha = 2.16$ (which corresponds to a slope of about 45° around the inflection point).

1.5.3 Probabilistic Interpretation of MP-Neuron

The expression (1.64) shows that the derivative of logistic function can be expressed in term of itself. Moreover, that is the most important, under a set of probabilistic assumptions, the sigmoid function can fit the parameters via maximum likelihood approach.

Let assume that

$$\begin{aligned} p(y=1|\mathbf{x};\mathbf{w}) &= f_{\mathbf{w}}(\mathbf{x}) \\ p(y=0|\mathbf{x};\mathbf{w}) &= 1-f_{\mathbf{w}}(\mathbf{x}) \end{aligned} \tag{1.70}$$

that in compact form can be written as

$$p(y|\mathbf{x};\mathbf{w}) = (f_{\mathbf{w}}(\mathbf{x}))^{y} \cdot (1-f_{\mathbf{w}}(\mathbf{x}))^{1-y} \tag{1.71}$$

Now, considering that we have N training examples $[\mathbf{x}_i, d_i]_{i=1}^{N}$ were generated independently. For 1.57 we can then write down the likelihood of the parameters as

$$\begin{aligned} L(\mathbf{w}) &= \prod_{i=1}^{N} p(d_i|\mathbf{x}_i;\mathbf{w}) \\ &= \prod_{i=1}^{N} (f_{\mathbf{w}}(\mathbf{x}_i))^{d_i} (1-f_{\mathbf{w}}(\mathbf{x}_i))^{1-d_i} \end{aligned} \tag{1.72}$$

and considering the log likelihood

$$\begin{aligned} \mathcal{L}(\mathbf{w}) &= \ln L(\mathbf{w}) \\ &= \sum_{i=1}^{N} \Big(d_i \ln f_{\mathbf{w}}(\mathbf{x}_i) + (1-d_i) \ln[1-f_{\mathbf{w}}(\mathbf{x}_i)] \Big) \end{aligned} \tag{1.73}$$

By writing explicitly the hypothesis $f_{\mathbf{w}}(\mathbf{x})$ considering a sigmoidal squashing function for the i-th input pattern, we have that $f_{\mathbf{w}}(\mathbf{x}) = \varphi(s)$, where $s = \mathbf{w}^T\mathbf{x}_i$, we can determine an gradient adaptation rule which maximizes the log-likelihood

$$\mathbf{w}_{new} = \mathbf{w}_{old} + \mu \nabla \mathcal{L}(\mathbf{w}).$$

The gradient $\nabla\mathcal{L}(\mathbf{w})$ can be determined as

$$\begin{aligned} \frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} &= \left(d_i \frac{1}{\varphi(s)} - (1-d_i) \frac{1}{1-\varphi(s)} \right) \frac{\partial \varphi(s)}{\partial \mathbf{w}} \\ &= \left(d_i \frac{1}{\varphi(s)} - (1-d_i) \frac{1}{1-\varphi(s)} \right) \varphi(s)\,(1-\varphi(s)) \frac{\partial \mathbf{w}^T\mathbf{x}_i}{\partial \mathbf{w}} \\ &= (d_i\,(1-\varphi(s)) - (1-d_i)\varphi(s))\,\mathbf{x}_i \\ &= (d_i - \varphi(s))\mathbf{x}_i \end{aligned}$$

Note that in the above expression we used the rule (1.64), i.e. $\varphi'(s) = \varphi(s)(1-\varphi(s))$. Thus the adaptation rule can be written as

$$\mathbf{w}_i = \mathbf{w}_{i-1} + \mu(d_i - \varphi(\mathbf{w}^T\mathbf{x}_i))\mathbf{x}_i \tag{1.74}$$

The previous rule is formally similar, but not exactly identical, to the LMS (see §1.2.1, Eqn. (2.2)).

1.6 Regularized LS

According to the french mathematician Jacques Hadamard [11], a *well-posed* problem is a problem for which the solution exists, is unique and there is continuous dependence of the solution from the data.

Definition 1.3. - One problem is indicated as *well-posed* if, and only if, it meets the following three requirements:

- a solution existence,
- the solution is unique,
- the behavior of the solution has continuity with the initial conditions.

On the contrary, if one of those three conditions is not satisfied the problem is called *ill-posed*. In this case, the numerical solution can not exist or that found can have a strong deviation from the true theoretical solution.

The Russian mathematician Tikhonov (or Tychonoff) was perhaps the first to study the problem of deviation from the true solution in terms of regularization [12]. The problem is posed in the definition of a criterion for the selection of an approximate solution among a set of *feasible solutions*.

The basic idea of the *Tikhonov's regularization theory*, consists in the determination of a compromise between a solution faithful to the noisy data and a solution based on *a priori* information available about the nature of the data (for example, knowledge of the model, the order of generation of the data, the statistics of the noise, etc.). In other words, the regularization imposes a smoothness constraint on the set of possible solutions.

In other words, to bypass the difficulty of finding "good" solution for ill-problems, Tikhonov suggested to use the *a priori* knowledge that, in real cases, are almost always available.

1.6.1 Penalized vs Constrained Regularization Forms

Many optimization problems are characterized by an hill-posed cost function $\hat{J}(\mathbf{w})$, a simple method to incorporate some *a priori* knowledge consists in defining a new function as the sum of two terms: the first term is the standard cost function $\hat{J}(\mathbf{w})$; while the second term is *regularization term* or *penalty function* , that is indicated as $J_r(\mathbf{w})$. In the so called *penalized* (or Lagrangian) forms, the new cost function is defined as

$$J(\mathbf{w}) \triangleq \hat{J}(\mathbf{w}) + \delta J_r(\mathbf{w}) \tag{1.75}$$

where δ is defined as *regularization parameters*.

The problem described by Eqn. (1.75) may also be described in terms of constrained optimization: it is necessary to find the minimum of a cost function that is subject to a constraint on the solution, derived from certain a *priori* knowledge.

In this case, the constraint over the weights is described in term of *feasible solution region*, that is delimited by a set of equality and/or inequality constraints on the parameters. The problem described by Eqn. (1.75) can be written in *constrained* forms as

$$\mathbf{w}_* \therefore \arg\min_{\mathbf{w}} \hat{J}(\mathbf{w}) \quad \text{subject to} \quad \mathbf{w} \in B \tag{1.76}$$

where B describe the region, denoted as a open or closed *ball*, that indicate the region of the feasible solution.

The standard LS problem is characterized by a cost defined as the sum of squared error $\mathbf{e}^T\mathbf{e}$ (see Eqn. (1.28)), i.e. the square of the L_2 norm of the vector $\mathbf{e}$ that is indicated as $||\mathbf{e}||_2^2$. So the regularized LS can be written as

$$\mathbf{w}_* \therefore \arg\min_{\mathbf{w}} \|\mathbf{d} - \mathbf{X}\mathbf{w}\|_2^2 \quad \text{subject to} \quad \mathbf{w} \in B \tag{1.77}$$

In regularized LS the ball B is usually chosen as a L_2 or L_1 norm of weights[13]. For these cases we can define two different regularization approaches so solve the constrained LS problem: the *ridge regression* and *Lasso regression* (least absolute shrinkage and selection operator) defined, respectively, as:

$$\mathbf{w}_* \therefore \arg\min_{\mathbf{w}} \|\mathbf{d} - \mathbf{X}\mathbf{w}\|_2^2 \qquad \text{subject to} \qquad \|\mathbf{w}\|_2 \leq \rho \tag{1.78}$$

$$\mathbf{w}_* \therefore \arg\min_{\mathbf{w}} \|\mathbf{d} - \mathbf{X}\mathbf{w}\|_2^2 \qquad \text{subject to} \qquad \|\mathbf{w}\|_1 \leq \rho \tag{1.79}$$

The ridge regression and lasso can be expressed as

$$\mathbf{w}_* \therefore \arg\min_{\mathbf{w}} \|\mathbf{d} - \mathbf{X}\mathbf{w}\|_2^2 + \delta \|\mathbf{w}\|_2 \tag{1.80}$$

$$\mathbf{w}_* \therefore \arg\min_{\mathbf{w}} \|\mathbf{d} - \mathbf{X}\mathbf{w}\|_2^2 + \delta \|\mathbf{w}\|_1 \tag{1.81}$$

These versions are called the penalized (or Lagrange) forms of the problems, whereas the ones above are called the constrained forms and where the regularization or tuning parameter δ is the Lagrange multiplier. Moreover, note for each of the constrained problems above, and every value of $\rho \leq 0$, there is a corresponding value of $\delta \leq 0$ such that the analogous penalized problem has the same solutions.

Remark 1.7. In machine learning, the regularization function and regularization parameter, are especially important, often the success of a method rather than another, depends on the choice of the function $J_r(\mathbf{w})$ and of the parameter δ.

1.6.2 Regularized L_2 LS: the Ridge Regression

In the case of rigde regression (1.78), the regularized LS cost function has the form

$$J(\mathbf{w}) = \|\mathbf{d} - \mathbf{X}\mathbf{w}\|_2^2 + \delta \|\mathbf{w}\|_2^2, \qquad \delta > 0. \tag{1.82}$$

The regularized solution $\mathbf{W}$ can be defined by placing its gradient to zero

$$\nabla J(\mathbf{w}) = \mathbf{X}^T (\mathbf{d} - \mathbf{X}\mathbf{w}) - \delta \mathbf{w} \equiv \mathbf{0}. \tag{1.83}$$

From the above may be derived the normal equations in the form

$$(\mathbf{X}^T\mathbf{X} + \delta \mathbf{I})\mathbf{w} = \mathbf{X}^T\mathbf{d} \tag{1.84}$$

with solution

$$\mathbf{w}_* = (\mathbf{X}^T\mathbf{X} + \delta \mathbf{I})^{-1}\mathbf{X}^T\mathbf{d}. \tag{1.85}$$

Note that the rigde LS solution coincides with the Levenberg-Marquardt solution to solve ill-posed linear inverse problems [14], [15].

1.6.2.1 Revisiting the polynomial regression problem: the overfitting problem

To better understand the problem of regularization, we consider the problem of the polynomial regression described in paragraph §1.2.2, for a non-linear function. In particular, the proposed problem can be formalized as: knowing some noisy samples of the function $f_0(\mathbf{x}) = \sin(2\pi x)$, in the interval $x \in [0\ 1]$, find and the polynomial coefficients that better approximated the original function (see Fig. 1.22).

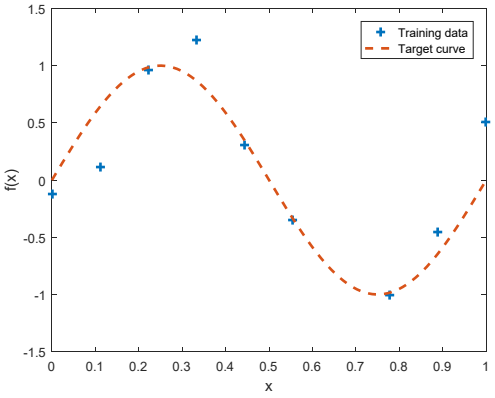

Fig. 1.22 Training data and target curve for revisited polynomial regression problem. The available data are obtained uniformly sampling the function $\sin(2\pi x)$, for $0 \le x \le 1$, ten samples in our case, and adding Gaussian noise with with standard deviation $\sigma_\eta = 0.3$.

Let $y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_P x^P = \mathbf{w}^T\mathbf{x}$ the given polynomial function, the available noisy samples represent our training set and the with LS method we minimize the square error, defined as

$$e_{LS} = \frac{1}{N}\sum_{n=1}^{N}\left\{d_n - \mathbf{w}^T\mathbf{x}\right\}^2$$

only over the training set.

Choosing an appropriate polynomial order, by the LS method we can estimate polynomial coefficients such that the $e_{LS} \to 0$. Indeed, considering 10 available training set points, by choosing a polynomial of order 9, (i.e. with 10 degree of freedom), the fitted curve passes exactly through all the 10 available points. This is illustrated in Fig. 1.23 where you can see that the approximating curve, passes exactly in the training points and the error is almost zero. However, the fitted polynomial curve oscillates wildly and gives a very poor representation of the target function $\sin(2\pi x)$.

This situation is denoted as *overfitting* (see §1.3.4.3), and it is said that the approximating curve "learn the training set". In other words, if the model is not adequate, for example the polynomial has a too high degree and there not any regularization remedy, we have that we learn the noisy training set, but not the underlying model, i.e. the generative model of the data.

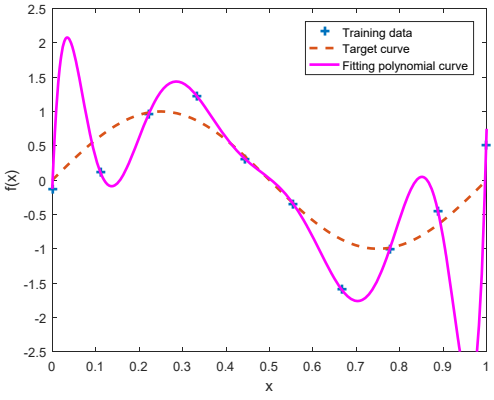

Fig. 1.23 If the polynomial has high degree the polynomial fitted curve can pass through the training set points. In these case we are in presence of the *over fitting problem* and the generalization performance are very poor.

Example 1.5. As a further example, Fig. 1.24 shows the results of the polynomial regression for broader training sets (20, 50 and 100 samples), also in presence of more noisy data.

Below, it shows the MATLAB code to generate the Fig. 1.24

```
%--------------------------------------------------------------------------
%  Naive Introduction of Least Squares Method
%  Polynomial fitting data by LS algorithm
%--------------------------------------------------------------------------
clear all;   close all;
 rng(11);
%
% Function generation ----------------------------------------------------
nPoints = 1000;
dt = 1/(nPoints - 1);
t = 0;
```

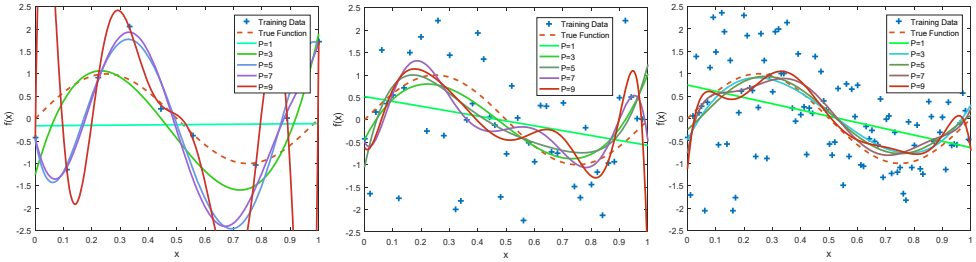

Fig. 1.24 Polynomial regression problem for different polynomial order and different size of training set. Left 20, center 50 and right 100 samples. In these experiments the added Gaussian noise with unitary standard deviation $\sigma_\eta = 1$.

```
for i = 1 : nPoints
    ft(i) = sin(2*pi*t);
    ts(i) = t;
    t = t + dt;
end

% ---- 10 noisy samples of sin function Oct. 2016 -------------------------
nSamples = 100;
d = zeros(nSamples, 1);
x = zeros(nSamples, 1);
dn = fix(nPoints / (nSamples - 1) );
n = 1;
for j = 1 : nSamples
    d(j) = ft(n) +  randn(1,1);
    x(j) = ts(n);
    n = n + dn;
end
%
% Polynomial Regression order and Learing Rate ----------------------------
P0 = 1;
DP = 2;
P = P0;
nRun = 5;
MinGenErrQ = 10;
delta = 1e-14;
for kk = 1 : nRun
    %-------------------------------------------------------------------------
    % LS solution -------------------------------------------------------------
    %-------------------------------------------------------------------------
    X = zeros(nSamples, P+1); % Data Matrix definition
    xx = zeros(P+1,1);  % vector for the calculation of the dot product  w'*xx
    % Build Data matrix X
    for i = 1 : nSamples
        X(i,1) = 1;
        for p = 1 : P    % add raw to X matrix for high degree polynomial
            X(i, p+1) = x(i)^p;
        end
    end
    Rxx = delta*eye(P+1) + X'*X;
    PP(kk).wLS = inv(Rxx)*X'*d;
    P = P + DP;
end
% -------------------------------------------------------------------------
% Plot regression curve  --------------------------------------------------
% -------------------------------------------------------------------------
figure;
hold on; box on;
xlim([0 1]);
```

```
ylim([-2.5 2.5]);
plot(x,  d,'+','LineWidth',2);
plot(ts, ft,'--','LineWidth',2);
P = P0;
for kk = 1 : nRun
    t = 0;
    xx = zeros(P+1,1);  % vector for the calculation of the dot product  w'*xx
    xx(1) = 1;
    for n = 1 : nPoints
        for p = 1 : P   % add elements x^p to xx vector for high degree polynomial
            xx(p+1) = t^p;
        end
        yLS(n)  =  PP(kk).wLS'*xx;
        t = t + dt;
    end
    P = P + DP;
    kk/nRun
    plot(ts, yLS, 'color',[ (kk-1)/nRun , 1 - (kk-1)/nRun, rand ], 'LineWidth',2);
end
% title(sprintf('Noisy polynomial regression,   \\delta =%.1e',  delta ));
xlabel('x');
ylabel('f(x)');
legend('Training Data', 'True Function', 'P=1', 'P=3',  'P=5',  'P=7', 'P=9');
% --------------------------------------------------------------------------
```

Remark 1.8. The overfitting/underfitting is one of the central problems in learning of neural networks and other machine learning and can be reduced if the choice of the model is adequate to the given problem, or using regularized learning strategies.

Because very often in real problems, both regression and classification, the most appropriate model order is almost unknown, the regularized learning approach are of central importance in all automatic learning systems.

1.6.2.2 Revisiting the polynomial regression problem: the regularized LS solution

Considering the problem of polynomial approximation described in Fig. 1.22 for a fixed order P of the fitted polynomial. We consider the regularized solution in (1.85), varying the regularization parameter δ. The problem is to determine the regularization parameter that minimizes the generalization error.

Example 1.6. In Fig. 1.25, are reported the trends of the the squared error e_{LS} on the training points, and the generalization error e_{GE} as a function δ, over the entire target function $\sin(2\pi x)$. Increasing δ, the generalization error tends to decrease until it reaches a minimum which determines the optimum value of the regularization parameter.

Fig. 1.26, shows the curve fitted computed with the optimization parameter equal to $\delta = 1e^{-3}$, which corresponds to the minimum generalization error in Fig. 1.25.

```
%--------------------------------------------------------------------------
%  Naive Introduction of Least Squares Methods
%  Polynomial fitting data by LS algorithm
%--------------------------------------------------------------------------
clear all;  close all;
```

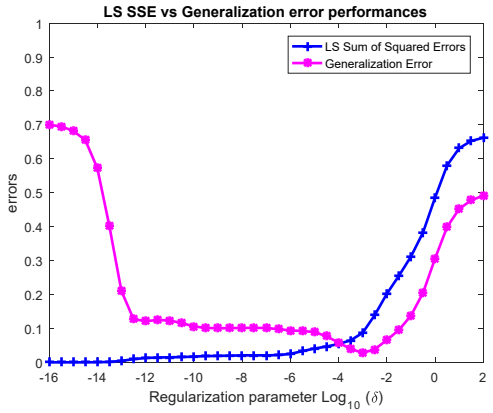

Fig. 1.25 LS error vs generalization error. In the case of over fitting the LS error in low while the generalization error high.

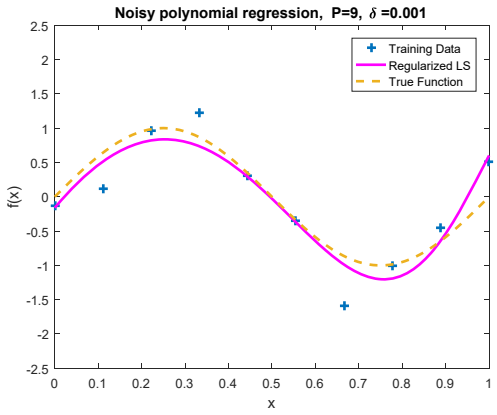

Fig. 1.26 Polynomial fitted curve, for $P = 9$, computed using LS method with an optimal regularization parameter $\delta = 1e^{-3}$.

```
rng(11);
% Function generation -----------------------------------------------------
nPoints = 1000;
dt = 1/nPoints;
t = 0;
for i = 1 : nPoints
    ft(i) = sin(2*pi*t);
    ts(i) = t;
    t = t + dt;
end
% ---- 10 noisy samples of sin function Oct. 2016 -------------------------
nSamples = 10;
d = zeros(nSamples, 1);
x = zeros(nSamples, 1);
dn = fix(nPoints / (nSamples - 1) );
n = 1;
for j = 1 : nSamples
    d(j) = ft(n) + 0.3*randn(1,1);
    x(j) = ts(n);
    n = n + dn;
end
%
% Polynomial Regression order and Learing Rate ----------------------------
P  = 9;
ln_init = -16.5
```

```
ln_delta(1) = ln_init;
nRun = 40;
MinGenErrQ = 10;
for kk = 1 : nRun
    ln_delta =  ln_delta + 0.5;
    ln_kk(kk) =  ln_delta; % only for plotting
    %-------------------------------------------------------------------------
    % LS solution ------------------------------------------------------------
    %-------------------------------------------------------------------------
    X = zeros(nSamples, P+1); % Data Matrix definition
    xx = zeros(P+1,1);  % vector for the calculation of the dot product  w'*xx
    % Build Data matrix X
    for i = 1 : nSamples
        X(i,1) = 1;
        for p = 1 : P    % add raw to X matrix for high degree polynomial
            X(i, p+1) = x(i)^p;
        end
    end
    delta(kk) = 10^ln_delta;
    Rxx    = delta(kk)*eye(P+1) + X'*X;
    wLS    = inv(Rxx)*X'*d;
    %-------------------------------------------------------------------------
    % Training set SSE or ----------------------------------------------------
    %-------------------------------------------------------------------------
    e = (d - X*wLS); % errors in training samples
    eq_LS(kk)  = sum(e.^2) / nSamples;
    % -----------------------------------------------------------------------
    % Compute the generalizaion error
    % -----------------------------------------------------------------------
    t = 0;
    xx(1) = 1;
    for n = 1 : nPoints
        for p = 1 : P   % add elements x^p to xx vector for high degree polynomial
            xx(p+1) = t^p;
        end
        yLS(n)  =  wLS'*xx;
        e(n) = ft(n) - yLS(n);
        t = t + dt; % x-axis for plot
    end
    eq_GN(kk)  = sum(e.^2) / nPoints;
    fprintf('Delta = %5.2e   ErrQ_LS = %5.2e   ErrQ_GEN = %5.2e \n', ...
        delta(kk),   eq_LS(kk), eq_GN(kk) );
    % -----------------------------------------------------------------------
    %  Optimal delta value
    % -----------------------------------------------------------------------
    if MinGenErrQ > eq_GN(kk)
        MinGenErrQ = eq_GN(kk);
        ik = kk;
    end
end
% -------------------------------------------------------------------------
% LS Sum of Squared Errors vs Generalization Error Plot -------------------
% -------------------------------------------------------------------------
figure;
hold on; box on;
xlim([ln_kk(1)   2 ]);
ylim([0 1 ]);
plot( ln_kk, eq_LS, '+-', 'color','b', 'LineWidth',2);
plot( ln_kk, eq_GN, '*-', 'color','m', 'LineWidth',2);
title('LS SSE vs Generalization error performances');
xlabel('Regularization parameter Log_1_0 (\delta)');
ylabel('errors');
legend('LS Sum of Squared Errors', 'Generalization Error');
% -------------------------------------------------------------------------
% Scttar plot and regression curve  ---------------------------------------
% -------------------------------------------------------------------------
Rxx    = delta(1)*eye(P+1) + X'*X;
wLS1   = inv(Rxx)*X'*d;
```

```
Rxx     = delta(ik)*eye(P+1) + X'*X;
wLS2    = inv(Rxx)*X'*d;
 t = 0;
 xx(1) = 1;
 for n = 1 : nPoints
        for p = 1 : P   % add elements x^p to xx vector for high degree polynomial
            xx(p+1) = t^p;
        end
        yLS1(n)  =  wLS1'*xx;
        yLS2(n)  =  wLS2'*xx;
        x1(n)  = t;
        t = t + dt; % x-axis for plot
end
figure;
hold on;
box on;
xlim([0 1]);
ylim([-2.5 2.5]);
plot(x,  d,'+','LineWidth',2);
% plot(x1, yLS1,'-', 'color','b', 'LineWidth',2);
plot(x1, yLS2,'-', 'color','m', 'LineWidth',2);
plot(ts, ft,'--','LineWidth',2);
title(sprintf('Noisy polynomial regression,  P=%1d, \\delta =%5.3f',P, delta(ik) )
    );
xlabel('x');
ylabel('f(x)');
%legend('Training Data', 'LS', 'Regularized LS', 'True Function');
legend('Training Data',  'Regularized LS', 'True Function');
% ------------------------------------------------------------------------
```

1.6.2.3 Regularized LMS algorithm: the leaky LMS

The algorithm *Leaky LMS* (LLMS), has an LF characterized by the sum of two contributions [17, 18]. At the standard LF of the LMS (1.19), it is added a penalty term proportional to the internal product $\mathbf{w}^T$, for which in Lagrange forme the LF is redefined as

$$J(\mathbf{w}) = |e[n]|^2 + \delta \mathbf{w}_{n-1}^T \mathbf{w}_{n-1} \tag{1.86}$$

where δ is denoted as the "*leak*". The square error minimization, along with the penalty function, limits the L_2 norm of weights during the adaptation process. Therefore, the expression 1.86 is a regularized LF.

For algorithm development, differentiating the (1.86), we get

$$\begin{aligned} \nabla J(\mathbf{w}) &= \frac{\partial(|e[n]|^2 + \delta \mathbf{w}_{n-1}^T \mathbf{w}_{n-1})}{\partial \mathbf{w}_n} \\ &= -2e[n]\mathbf{x} + 2\delta \mathbf{w}_{n-1} \end{aligned} \tag{1.87}$$

for which, the law adaptation is

$$\begin{aligned} \mathbf{w}_n &= \mathbf{w}_{n-1} - \mu(-e[n]\mathbf{x} + \delta \mathbf{w}_{n-1}) \\ &= (1 - \mu\delta)\mathbf{w}_{n-1} + \mu e[n]\mathbf{x}. \end{aligned} \tag{1.88}$$

Moreover, the regularized form of the LMS algorithm can be considered also for EALC with soft or smooth squashing functions.

Example 1.7. Comparison LMS and Leaky LMS adaptation algorithm for EALC with bipolar sigmoid squashing-function. The training set consists in 60 samples of the two moons problem, to which have been added Gaussian noise with standard deviation $\sigma_\eta = 0.25$. Fig. 1.27 shows the comparison results.

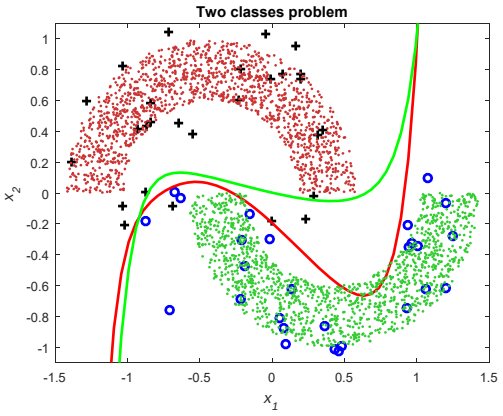

Fig. 1.27 Comparison results between LMS (red curve) and leaky LMS (green curve) for EALC with bipolar sigmoid squashing-function. Polynomial order $P = 13$, nr of iteration 2000, learning rate $\mu = 1e^{-1}$, for LLMS $\delta = 1e^{-2}$, 60 training set pairs the added noise $\sigma_\eta = 0.25$.

Note that, despite the high polynomial order $P = 13$, from the figure you can observe as the regularized solution (green line) is less loyal to the training data and more inclined to identify the generation model. Moreover, it is also noted, that the sigmoid function acts as a limiter of the output and therefore also of the error. This has a twofold advantage in that it allows to have a more stable adaptation and with

It is also noted that the sigmoid function of EALC acts as a limiter of the output and therefore of the error. This has a twofold advantage in that it allows to have a more stable adaptation, with an higher value of the learning rate parameter, and then a smaller number of iterations learning.

Below the MATLAB program

```
% --------------------------------------------------------------------------
clear all; close all;
%
rng(19);
% --------------------------------------------------------------------------
% ---- Training Set Definition and generation ------------------------------
% --------------------------------------------------------------------------
nSamples = 60;
data = two_moons(nSamples);
sigma = 0.25;
x1 = data(:,1)  +  sigma*randn(nSamples,1);
x2 = data(:,2)  +  sigma*randn(nSamples,1);
d =  data(:,3);
% --------------------------------------------------------------------------
% **** LMS/Leaky-LMS algorithms ---------------------------------------------
% --------------------------------------------------------------------------
nIteration = 2000   ;
mu = 1e-1;      % LMS Learning rate
delta = 1e-2;  % Leaky LMS regulariation parameter
% --------------------------------------------------------------------------
% LMS  error correction -- Red ---------------------------------
```

```
% ------------------------------------------------------------------------
P  = 13;
w1 = zeros(P+2,1);
w2 = zeros(P+2,1);
xw = zeros(P+2,1);
xw(1) = 1;
 for nn = 1 : nIteration
    for k=1:nSamples
        for p = 1 : P   % add elements x^p to xx vector for high degree polynomial
            xw(p+1) = x1(k)^p;
        end
        xw(P+2) = x2(k);
        s = w1'*xw;
        y =  2 /( 1 + exp(-s) ) - 1;  % bipolar sigmoid squashing function
        e =  d(k) - y ;   % on-line  w = w + mu*e*y*(1 -y)*xw;  % LMS adaptation
     rule
        w1 =  w1 + mu*e*xw;  % Leaky-LMS adaptation rule error computation
       % w1 =  w1 + mu*e*(1 - y^2)*xw;  % Leaky-LMS GDR error computation
    end
end
% ------------------------------------------------------------------------
% Leaky-LMS Solution
% ------------------------------------------------------------------------
gamma = mu*delta;
for nn = 1 : nIteration
    for k=1:nSamples
        for p = 1 : P   % add elements x^p to xx vector for high degree polynomial
            xw(p+1) = x1(k)^p;
        end
        xw(P+2) = x2(k);
        s = w2'*xw;
        y =  2 /( 1 + exp(-s) ) - 1; % bipolar sigmoid squashing function
        e =  d(k) - y ; %+ randn;    % on-line error computation
        w2 = (1 - gamma)*w2 + mu*e*xw;  % Leaky-LMS adaptation rule
        %w2 =(1 - dd)*w2 + mu*e*(1 - y^2)*xw; % Leaky-LMS GDR
    end
end
for k=1:P+1
    cw1(k) = -w1(k)/w1(P+2);  % LMS
    cw2(k) = -w2(k)/w2(P+2);  % L-LMS
end
%
% ------------------------------------------------------------------------
% Classification function
% ------------------------------------------------------------------------
xmin = -1.5;
xmax =  1.5;
x  = xmin;
dx = (xmax - xmin)/(nSamples-1);
xw = zeros(P+1,1);
xw(1) = 1;
for i = 1 : nSamples
    for p = 1 : P   % add elements x^p to xx vector for high degree polynomial
       xw(p+1) = x^p;
    end
  ff1(i) = cw1*xw;  % LMS
  ff2(i) = cw2*xw;  % LLMS
  xx(i)  = x;
  x  = x + dx;
end
%
% ------------------------------------------------------------------------
% Plot results
% ------------------------------------------------------------------------
figure;
hold on;  box on;
 for n=1:nSamples
    if d(n)==1
        plot(x1(n), x2(n), '+', 'color','k','LineWidth',2);
```

```
    else
        plot(x1(n), x2(n), 'o', 'color','b','LineWidth',2);
    end
end
ylim([ -1.1  1.1 ]);
xlim([ xmin  xmax ]);
title('Two classes problem');
plot(xx, ff1, 'color', 'r', 'LineWidth', 2); %LLMS
plot(xx, ff2, 'color', 'g', 'LineWidth', 2); % LLMS
xlabel('\it{x}_1');
ylabel('\it{x}_2');
w1'
w2'
% --------------------------------------------------------------------------
```

The program output is:

```
Estimated Polynomial coefficients LMS w1
2.9938 13.1240 5.2600 -8.9167 -0.3081 -3.6188 -3.6926 -1.4006 -4.1651 -1.5446
-3.1730 -2.8977 -1.6734 -4.8115 15.3733

Estimated Polynomial coefficients Leaky LMS w2
-0.0093 0.8289 -0.4319 -0.5456 -0.2919 -0.6617 -0.1614 -0.6762 -0.0704
-0.6734 -0.0009 -0.6674 0.0577 -0.6629 3.8734
```

Finally, we can notice how the regularized solution limits the value of the polynomial coefficients.

1.6.2.4 Regularization by early stopping criteria

Given noisy training set to training an adequate model $f(\mathbf{x}, \mathbf{w})$, to obtain a regularized solution, a simple approach is the so called *early stopping* criteria

The early stopping criteria are based on analysis of upper bounds on the generalization error as a function of the iteration number. For this reason, we split the available data into training set and validation set (for example in a 2-to-1 proportion). The learning procedure is performed on the training set, while the error is measured on the validation set. As can be seen in example of Fig. 1.28, while the error on the training set tends to decrease with increase of the number of iterations, the error on the test set has a minimum.

More sophisticated forms of early stopping use *cross-validation* criteria using more sophisticate multiple partitions of the data into training set and validation set. Although conceptually simple, the early stopping criteria can be, in practice, complicated by the fact that the validation error can oscillate during the learning producing more local minima.

Example 1.8. Consider the two moons classification experiment with a training set consisting of 60 samples and a test set of 40 samples. The standard deviation noise added is $\sigma_\eta = 0.25$.

Fig. 1.28 shows the typical trends of the training set error vs error test set.

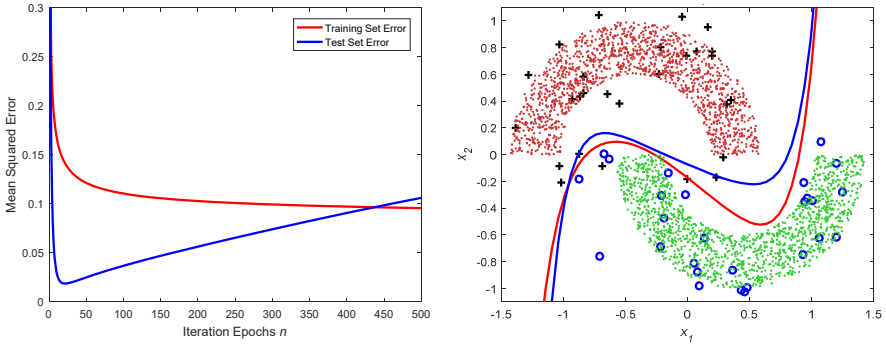

Fig. 1.28 Example of early stopping criteria. Left) Training set error vs test set error for the two moons problem; right) estimate classification curve at early stop (blue) and after 500 iteration (red).

```
% ------------------------------------------------------------------------
% Two moons classificatin problem
% ------------------------------------------------------------------------
clear all; close all;
%
rng(19);
% ------------------------------------------------------------------------
% ---- Training Set Definition and generation ----------------------------
% ------------------------------------------------------------------------
nSamples = 60;
data = two_moons(nSamples);
sigma = 0.25 ;
x1 = data(:,1)  +  sigma*randn(nSamples,1);
x2 = data(:,2)  +  sigma*randn(nSamples,1);
d =  data(:,3);
% ------------------------------------------------------------------------
% ---- Test Set Definition and generation --------------------------
% ------------------------------------------------------------------------
nSamplesTest = 40;
data = two_moons(60); % min 60 samples
xt1 = data(1:nSamplesTest,1)  +  sigma*randn(nSamplesTest,1);
xt2 = data(1:nSamplesTest,2)  +  sigma*randn(nSamplesTest,1);
dt =  data(1:nSamplesTest,3);
% ------------------------------------------------------------------------
% **** LMS algorithms -----------------------------------------
% ------------------------------------------------------------------------
nIteration = 500;
StopCrit = 20;
mu =0.2;     % LMS Learning rate
% ------------------------------------------------------------------------
% LMS  error correction
% ------------------------------------------------------------------------
P  = 9;
w1 = zeros(P+2,1);
w2 = zeros(P+2,1);
xw = zeros(P+2,1);
xw(1) = 1;
for nn = 1 : nIteration
    % Adaptation on training set ---------------------------------------
    for k=1:nSamples
        for p = 1 : P   % add elements x^p to xx vector for high degree polynomial
            xw(p+1) = x1(k)^p;
        end
        xw(P+2) = x2(k);
```

```
        s = w1'*xw;
        y =  2 /( 1 + exp(-s) ) - 1;  % bipolar sigmoid squashing function
        e =  d(k) - y ;   % on-line  w = w + mu*e*y*(1 -y)*xw;  % LMS adaptation
     rule
        w1 =  w1 + mu*e*xw;  % Leaky-LMS adaptation rule error computation
       % w1 =  w1 + mu*e*(1 - y^2)*xw;  % Leaky-LMS GDR error computation
    end
    if nn == StopCrit,
        w_opt =  w1;
    end
    % Global rrror computation on training set  ---------------------------
    tr_qerr(nn) = 0;
    for k=1:nSamples
        for p = 1 : P   % add elements x^p to xx vector for high degree polynomial
            xw(p+1) = x1(k)^p;
        end
        xw(P+2) = x2(k);
        s = w1'*xw;
        y =  2 /( 1 + exp(-s) ) - 1;  % bipolar sigmoid squashing function
        e =  d(k) - y ;     % LMS adaptation rule
        tr_qerr(nn) = tr_qerr(nn) + e^2;
    end
    tr_qerr(nn) = tr_qerr(nn) ./ nSamples;
    % Global rrror computation on test set  ---------------------------
    ts_qerr(nn) = 0;
    for k=1:nSamplesTest
        for p = 1 : P   % add elements x^p to xx vector for high degree polynomial
            xw(p+1) = xt1(k)^p;
        end
        xw(P+2) = xt2(k);
        s = w1'*xw;
        y =  2 /( 1 + exp(-s) ) - 1;  % bipolar sigmoid squashing function
        e =  dt(k) - y ;    % LMS adaptation rule
        ts_qerr(nn) = ts_qerr(nn) + e^2;
    end
    ts_qerr(nn) = ts_qerr(nn) ./ nSamplesTest;
end
for k=1:P+1
    cw1(k) = -w1(k)/w1(P+2);        % LMS
    cw2(k) = -w_opt(k)/w_opt(P+2);  % Stop Criteria
end
% ------------------------------------------------------------------------
% Classification function
% ------------------------------------------------------------------------
xmin = -1.5;
xmax =  1.5;
x  = xmin;
dx = (xmax - xmin)/(nSamples-1);
xw = zeros(P+1,1);
xw(1) = 1;
for i = 1 : nSamples
    for p = 1 : P   % add elements x^p to xx vector for high degree polynomial
       xw(p+1) = x^p;
    end
  ff1(i) = cw1*xw;  % LMS
  ff2(i) = cw2*xw;  %Stop Criteria
  xx(i)  = x;
  x  = x + dx;
end
% ------------------------------------------------------------------------
% Plot results
% ------------------------------------------------------------------------
figure;
hold on;  box on;
 for n=1:nSamples
    if d(n)==1
        plot(x1(n), x2(n), '+', 'color','k','LineWidth',2);
    else
        plot(x1(n), x2(n), 'o', 'color','b','LineWidth',2);
```

```
    end
end
ylim([ -1.1   1.1 ]);
xlim([ xmin   xmax ]);
title('Two classes problem');
plot(xx, ff1, 'color', 'r', 'LineWidth', 2); %LLMS
  plot(xx, ff2, 'color', 'b', 'LineWidth', 2); % LLMS
xlabel('\it{x}_1');
ylabel('\it{x}_2');
% --------------------------------------------------------------------
% Plot results
% --------------------------------------------------------------------
figure;
hold on;  box on;
ylim([ 0   0.3 ]);
% plot(10*log10(tr_qerr), 'color', 'r', 'LineWidth', 2); % LLMS
% plot(10*log10(ts_qerr), 'color', 'b', 'LineWidth', 2); % LLMS
plot(tr_qerr, 'color', 'r', 'LineWidth', 2); % LLMS
plot(ts_qerr, 'color', 'b', 'LineWidth', 2); % LLMS
xlabel('Iteration Epochs \it{n}');
%ylabel('10log({\itJ}({\bfw}))');
ylabel('Mean Squared Error');
legend('Training Set Error','Test Set Error');
% --------------------------------------------------------------------
```

1.6.3 Regularized L_1 LS: the LASSO T.B.C.

TBC

1.6.4 On the Norm Order p of the Penalty Term

Given a vector $\mathbf{w} \in \mathbb{R}^N$, its norm refers to its length relative to a vector space. In the case of a normed-space of order p, said L_p space, the norm is indicated as $||\mathbf{w}||_p$, and is defined a

Definition 1.4. (L_p Vector Norm) - A generic norm of a vector is defined as

$$\|\mathbf{w}\|_p \triangleq \left[\sum_{i=1}^{N} |w_i|^p\right]^{1/p}, \quad p \geq 1. \tag{1.89}$$

The choice of the norm of the constraint is strictly related to the problem to be solved. The L_2 norm allows to obtain a smooth weights solutionand is widely used because it can be implemented with simple linear algorithms. With the norm L_2 you get a vector solution with all the non-zero elements.

The L_1 norm is used when the solution to be obtained is sparse, i.e. with many zero entries. Sparse solution is of great importance in the case where the unknown vector has large size (as in Big-Data) in which only a sub-set of solutions is important.

For some problems we defines a norm with $p < 1$

Definition 1.5. (L_0 Vector Norm) - The expression (1.89) is valid even when $0 < p < 1$; however, the result is not exactly a norm. For $p = 0$, the (1.89) becomes

$$\|\mathbf{w}\|_0 \triangleq \lim_{p\to 0} \|\mathbf{w}\|_p = \sum_{i=1}^{N} |w_i|^0 \tag{1.90}$$

that is the *numerosity* of non-zero elements; i.e. the number of nonzero entries of the vector $\mathbf{w}$.

1.7 Proposed Problems

Exercise 1.1. Modify the MATLAB procedure of the Example 1.2 in order to have a quadratic (or higher degree) regression. In particular determine, some empirical data normalization, in order to have a 'consistent' (with no numerical errors) results. For example: normalize the available data, find the values of the learning rate and the regularization parameter (for the batch procedure), search for more robust algorithms, etc.

Exercise 1.2. Write a MATLAB procedure for the Example 1.2 in order to plot the quadratic loss function surface $J(\mathbf{w})$ (see Eqn. (1.29)).

Exercise 1.3. The matrix $\mathbf{X}^T\mathbf{X}$ that appears in the normal equations of Yule-Walker (see Eqn. 1.30) represents an intrinsic estimation of the *covariance matrix* of the input data $\mathbf{x}$.

1. Justify the above statement (search on the internet, for example using Wikipedia);
2. demonstrate that $\mathbf{X}^T\mathbf{X}$ is a *normal matrix*;
3. list and discuss the salient properties of normal matrices.

Exercise 1.4. Write a batch MATLAB procedure for the problem of classification of two classes described in Example 1.3.

Exercise 1.5. Extend the procedure of Example 1.3, for a features vector of dimension M. Plot the separation surface for the case $M = 3$.

Exercise 1.6. Write a program in order to evaluate the classification error for the comparison of the procedure 'data error correction' and 'quantized error correction', used for the estimation of the classification function.

Exercise 1.7. In Fig. 1.29 is reported the scheme of the polynomial linear combiner, denoted *Widrow's Padaline*.

Write a MATLAB program for the adaptation of Padaline parameters, with $M \geq 2$ inputs, by LMS like algorithms. Furthermore, demonstrate that the performance surfaces is convex, therefore, with a single global minimum. Find that the optimal solution is thus guaranteed.

Exercise 1.8. In more general form the sigmoid squashing function can be written as:

$$\varphi(s) = \frac{G}{1 + e^{-\alpha s}} \tag{1.91}$$

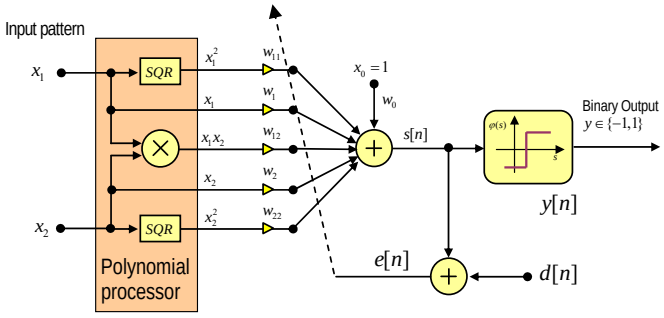

Fig. 1.29 The Widrow's polynomial linear combiner *Padaline*.

where the parameters α and G represents, respectively, the slope and gain of the curve. The sigmoid function can assume each value form 0 to G and is a differentiable function.

For the expression (1.91) find:

- the expression of the derivative $\frac{\partial \varphi(s)}{\partial s}$;
- the value of α such that for $s = 0$ the slope of the curve is equal to 1.

Exercise 1.9. A more general form of $\varphi(\cdot)$ is a *sigmoide bipolar function*, define as :

$$\varphi(s) = G\left(\frac{2}{1+e^{-\alpha s}} - 1\right) \tag{1.92}$$

For the above expression demonstrate that:

- the expression (1.92) is equivalent to $\varphi(s) = G\tanh(\alpha s/2)$;
- the derivative of (1.92) is

$$\frac{\partial \varphi(s)}{\partial s} = \alpha G[1 - \varphi^2(s/2)].$$

Exercise 1.10. In probabilistic interpretation of MP-Neuron, it was obtained the MLE learning algorithm in the case of unipolar-sigmoidal activation function. Determine the estimator MLE in the case of bipolar activation function 1.92.

Chapter 2
Introduction to Adaptive Filtering

2.1 Introduction to Linear Adaptive Filtering

In the curve fitting and pattern recognition problems discussed above, the data are processed in no particular order. The problem in this case is said *static* or *memory less* and the succession in which the data are presented to the learing machine, regressor or pattern recongition system, is of not relevance. In fact, the data acquired from *Porta Portese* are 'as is' a and may be presented to the learning machine with any other order without modify the performance of the regression.

In other kind of problems, such as the weather forecast or the study of the performance of a certain economic index, the data show a sequential structure, i.e. what happens now depends on what has happened before; or in other cases, there may be a certain spatial relationship: what happens here may depend on what happened in the neighborhood; or, in yet other cases, there may be simultaneously spatial, temporal and other domains dependencies. For example, in speech analysis, the probability of a certain phoneme depends on the previous ones. In image processing, the luminosity intensity of a pixel depends in some way from its neighborhood, while in a video the luminosity intensity of a pixel at time n depends on its neighborhood at time $n,\ n-1\ n-2,\ ....$

In these cases the problem is said *dynamic* or *with memory* or *not memory less* (in the sense of time, patial or/and other domains dependencies), and the succession in which the data are presented to the learing machine following the causality of the problem and must not be changed.

In the latter case the length of the memory of the system is of great importance. The input that feeds the network is a sequence of ordered data for the whole length of the needed memory.

An *linear adaptive filter* or simply *adaptive filter* (AF) represents one of the simplest models of adaptive system, usually implemented by a supervised learning methods, that is able to learn temporal (or other domains) dependencs of an ordere sequence of data.

The AF can be seen as a computational device that can be realized as a set of program instruction or on an hardware device, in which the free parameters $\mathbf{w}$, are

continuously changed according to *a priori* defined criteria at the variation of the statistic of signals at its input.

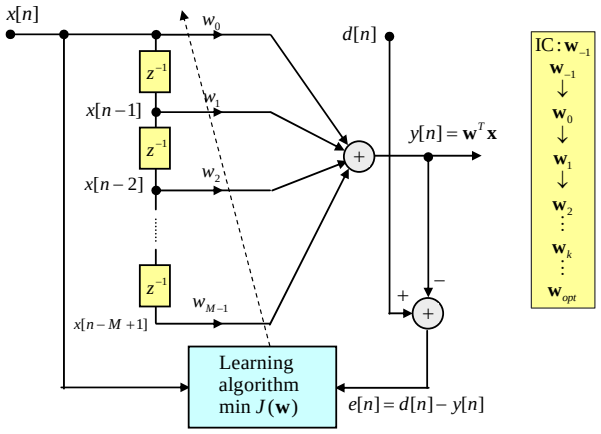

Fig. 2.1 Signal flow graph of a linear adaptive filter usually called *transversal adaptive filter*. The filter is composed of delay line mechanism followed by an ALC. The output is a linear combination between the inputs $x[n]$, $x[n-1]$, $\cdots$, $x[n-M+1]$, and the filter coefficients $w[0]$, $w[1]$, $\cdots$, $w[M]$.

The simplest AF is that shown in Fig. 2.1 and as you can see from the figure, the AF is simply an ALC (see Fig. 1.5 whose input is placed a *delay line* mechanism that allows an orderly and sequential input to the ALC.

Let $x[n]$ the n-th sample of a sequence, the input of the M-length ALC is fed by the samples $x[n]$, $x[n-1]$, ... $x[n-M+1]$, the relationship between the input $x[n]$ and the output $y[n]$ is linear and defined by the following scalar product

$$y[n] = \sum_{k=0}^{M-1} w[k]x[n-k] = \mathbf{w}^T\mathbf{x} \tag{2.1}$$

which represents the convolution sum, between the input $x[n]$ and the filter parameters $w[n]$ that, as we shall see in the next paragraphs, represents impulse response of the filter.

In more formal terms an AF can be defined by the presence of two distinct parts:

1. an algorithm that processes an input sequence and produces a certain output, such as in the linear case, that implements the relation (2.1);
2. an algorithm of the type $\mathbf{w}_{new} = \mathbf{w}_{old} + \Delta\mathbf{w}$, indicated as adaptation or learning algorithm, that computes the filter parameters $\mathbf{w}_n$, with a predetermined criterion of the type $\Delta\mathbf{w} = f(\mu, e[n], \mathbf{x})$.

2.1.1 LMS Learning Rule for Adaptive Filtering

In order to derive a simple rule of adaptation, we proceed in a similar way as previously reported in §1.2.3 where the loss function to be minimized is exactly the *instantaneous square error* (ISE) at time n, that is $J(\mathbf{w}) = e^2[n]$. So, considering a stochastic gradient

approach, developed in Eqn.s (1.19)-(1.22), the learning rule is LMS that can be formulated as

$$\mathbf{w}_n = \mathbf{w}_{n-1} + \mu e_n \mathbf{x}_n, \quad \text{for} \quad i = 1, \ ,2 \ \ldots \ N. \tag{2.2}$$

where M is the length of the transversal filter, $\mathbf{x} = [x[n] \ x[n-1] \ \cdots \ x[n-M+1]]$ is the signal that flow in the filter delay-line, and μ is the *step-size* or *learning rate* parameter.

2.1.1.1 Delay line

Nelle problematiche che necessitiano una soluzione real-time, come per esempio nel DSP, la implementazione della delay-line riveste una certa importanza e non può essere trascurata. Il modo più semplice per realizzare una linea di ritardo, consiste nell'aggiornare una lista First Input Last Output (FILO). La implementazione della linea di ritardo in modo software può essere fatta con un vettore in cui vengono fatti scorrere i campioni come illustrato in Fig. 2.2.

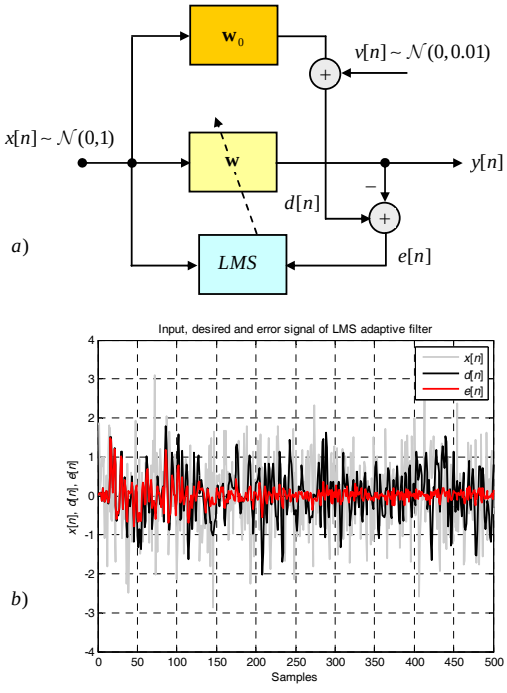

Fig. 2.2 Schema del buffer che realizza una DL e una possibile function (in "C") che la implementa.

Una implementazione efficiente della linea di ritardo è quello che evita la operazione di scorrimento secondo la tecnica denominata indirizzamento circolare. Infatti, è possibile pensare al buffer che implementa la DL come se fosse disposto circolarmente. Invece di effettuare lo scorrimento dei campioni lungo la linea, viene incrementato il valore dell'indice (puntatore) che indirizza la posizione dove viene immesso il campi-

one del segnale di ingresso Il primo campione di segnale viene immesso nella posizione zero, il secondo nella posizione uno, e così via. Quando la lunghezza della DL viene superata, la prima posizione del buffer viene sovrascritta e così via (wrap). Nel buffer sono quindi sempre presenti gli ultimi D, campioni del segnale.

2.1.2 Main Application of AF

Principal applications

1. Direct modeling
2. Inverse modeling
3. Prediction

2.1.2.1 Direct Model Identification

In order to verify the LMS learning rule for adaptive filtering problems, consider a simple example of model identification where the input signal is a WGN $x[n] \sim \mathcal{N}(0,1)$ and the derided signal is defined as $d[n] = \mathbf{w}_0^T \mathbf{x} + \mathcal{N}(0, 0.01)$. The adaptation scheme is reported in Fig. 2.3-a).

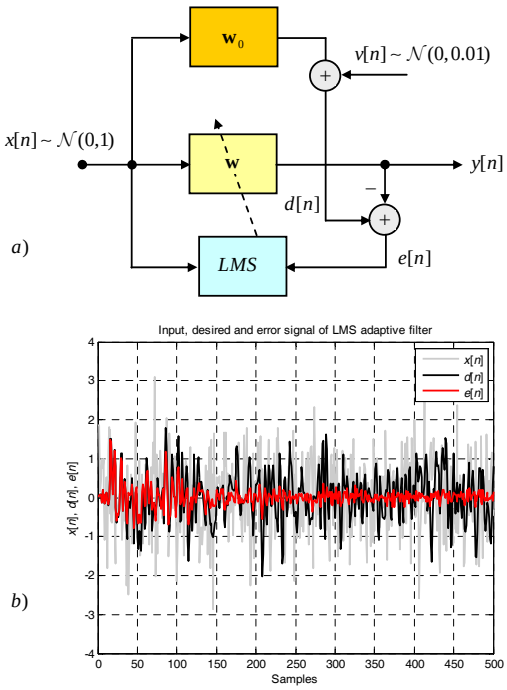

Fig. 2.3 Adaptive filtering by LMS learning rule for identification problems. a) Adaptation scheme by LMS algorithm. b) Input signal $x[n]$, desired signal $d[n]$ and error signal $e[n]$, during the adaptation problem.

In Fig. 2.3-b) are plotted the signal $x[n]$, $d[n]$ and $e[n]$, during the adaptation for a simple identification problem. In particular, for the identification of a filter impulse

response $\mathbf{w}_0$ of length $M = 31$ for $\mu = 0.01$ (see the code below). Note that the impulse response $\mathbf{w}_0$ is generated by a MATLAB function `w0 = fir1(M-1,0.5)`, that designs an $(M-1)$-th order FIR filter using the so called "window method", (see the MATLAB `help fir1` for details).

From the figure we can observe how the error tends to decrease with the number of iterations of the algorithm. As we shall see in the next chapters, at convergence the error variance is identical to the noise variance superimposed on the desired output signal.

Example 2.1. In the following, is reported the MATLAB code for generated the plot in Fig. 2.3-b).

```
%--------------------------------------------------------------------------
%  Naive Introduction of LMS algorithm:
%  Filtering a noisy signal by LMS and LS algorihm
%
%   Copyright 2016 - A. Uncini
%   DIET Dpt - University of Rome 'La Sapienza' - Italy
%   $Revision: 1.0$  $Date: 2016/03/13$
%--------------------------------------------------------------------------
clear all; close all;
s1=0.1;s2=0.2;
N = 500;   M = 31;
SNR_dB   = 20; % Noise level in dB
noise_std_dev = 10^(-SNR_dB /20)  % Convert [dB] in natural value
x   = randn(N,1);     % Input to the filter
w0  = fir1(M-1,0.5)';     % FIR system to be identified
d   = filter(w0,1,x) + noise_std_dev*randn(N, 1);  % Desired signal
mu  = 0.01;            % LMS step size
%
% ************************************************************************
% LMS solution
% ************************************************************************
xn  = zeros(M, 1);  % delay-line input buffer
w   = zeros(M, 1);  % filter coeff.s
e   = zeros(N, 1);  % error  vector (needs for plot)
y   = zeros(N, 1);  % output vector (needs for plot)
for n = 1 : N
    for i = M : -1 : 2 xn(i) = xn(i-1); end %  delay-line up-date
    xn(1) = x(n);           % load new input sample into filter delay-line
    y(n)  = w'*xn;          % output filter computation
    e(n)  = d(n) - y(n);    % error computation
    w = w + mu*e(n)*xn ;    % LMS adaptation
end

% --- Signals plot ------------------------------------------------------
figure;
grid on;
hold on; box on;
plot(x,  'color', [0.8  0.8  0.8], 'LineWidth',2);
plot(d,  'color', [0  0  0], 'LineWidth',2);
plot(e,  'color', 'r', 'LineWidth',2);
ylim([-4 4]);
title('Input, desired and error signal of LMS adaptive filter');
xlabel('Samples');
ylabel('{\itx}[{\itn}], {\itd}[{\itn}], {\ite}[{\itn}]');
legend('{\itx}[{\itn}]', '{\itd}[{\itn}]', '{\ite}[{\itn}]');
```

2.1.2.2 Inverse modeling

2.1.2.3 Prediction

2.1.3 A Bit of Wiener's Adaptive Filter Theory

The vector of free parameters $\mathbf{w}$ calculation of an AF is usually computed according optimization criterion that minimizes a predefined $J(\mathbf{w})$. The criterion is usually chosen depending on the input signal characteristics.

If the nature of the input signal is stochastic, the LF $J(\mathbf{w})$ is a function of some statistic of the error signal. In these cases, it is usual to consider the statistical expectation (1.5) (or expected value or ensemble-average value), indicated by the operator $E\{\cdot\}$, of the square of error signal. Such quantity is indicate as *mean-square error* (MSE) and is defined as

$$J(\mathbf{w}) = E\{|e[n]|^2\}. \tag{2.3}$$

The minimization of Eqn. (2.3) is performed by a stochastic optimization criterion called *minimum mean square error* (MMSE).

Thus, for the determination of the minimum of the function (2.3), as for LMS, we proceed by calculating the gradient of $J(\mathbf{w})$ and putting the result to zero: $\nabla J(\mathbf{w}) \to 0$. For the derivative computation we consider the explicit error formulation $e[n] = (d[n] - \mathbf{w}^T\mathbf{x})^2$, and developing the square for Eqn.s (2.3), we have that $J(\mathbf{w}) = E\{d^2[n] - 2\mathbf{x}^T\mathbf{w}d[n] + \mathbf{w}^T\mathbf{x}\mathbf{x}^T\mathbf{w}\}$. For the linearity of the expectation operator and since $\mathbf{w}$ is a constant term, that can be brought out of the expectation, we can write

$$J(\mathbf{w}) = E\{d^2[n]\} - 2\mathbf{w}E\{\mathbf{x}d[n]\} + \mathbf{w}^T E\{\mathbf{x}\mathbf{x}^T\}\mathbf{w}. \tag{2.4}$$

The terms of the expectations in the previous expression have an important statistical significance. In particular we have that:

- the first term on the right side is the variance of the signal $d[n]$ usually indicated as $\sigma_d^2 = E\{d^2[n]\}$;
- the second term represents the *cross-correlation vector* among the input $\mathbf{x}$ and the desired signal $d[n]$ usually indicated as

$$\mathbf{g} = E\{\mathbf{x}d[n]\} \tag{2.5}$$

- the last term is the *autocorrelation matrix* of the input sequence usually indicated as

$$\mathbf{R} = E\{\mathbf{x}\mathbf{x}^T\}. \tag{2.6}$$

Thus, from the previous definitions the expression (2.7) can be reduced to the following quadratic form

$$J(\mathbf{w}) = \sigma_d^2 - 2\mathbf{w}^T\mathbf{g} + \mathbf{w}^T\mathbf{R}\mathbf{w} \tag{2.7}$$

with gradient defined as

$$\begin{aligned} \nabla J(\mathbf{w}) &= \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} \\ &= \frac{\partial(\sigma_d^2 - \mathbf{w}^T\mathbf{g} - \mathbf{g}^T\mathbf{w} + \mathbf{w}^T\mathbf{R}\mathbf{w})}{\partial \mathbf{w}} \\ &= 2\left(\mathbf{R}\mathbf{w} - \mathbf{g}\right). \end{aligned} \tag{2.8}$$

From the previous expression, you can write the following system of linear equations

$$\mathbf{R}\mathbf{w} = \mathbf{g} \tag{2.9}$$

known as *normal equations* in the Wiener-Hopf notation [20]. The solution of the system (2.9), also known as the Widrow-Hoff Widrow equations can be written as

$$\mathbf{w} = \mathbf{R}^{-1}\mathbf{g} \tag{2.10}$$

Remark 2.1. In the Wiener's optimal filtering theory, the filter's inputs are considered as a stochastic process described in terms of their *a priori* known second-order statistics. Note that many authors (see e.g. [2], [23]) define the adaptive filter, the filter whose parameters are iteratively adjusted based on the new signal samples that gradually flows to its input. In fact, the determination of the coefficients $\mathbf{w}_{opt}$, it is not a direct function of the signal flow input samples, but the filter is designed on the base on *a priori* knowledge of second order moments of the input sequence.

2.1.3.1 On the Correlations Matrix Estimation

Note the Wiener's expression (2.9) is formally similar to the LS solution (see Eqn. (1.30)). From the definitions of the the cross-correlation vector $\mathbf{g} = E\{\mathbf{x}d[n]\}$ and correlation matrix $\mathbf{R} = E\{\mathbf{x}\mathbf{x}^T\}$, we have that the terms that appears in Yule-Walker's normal equations are strictly related to these quantity.

In fact, the correlation matrix can be estimated as

$$\mathbf{R}_{xx} = \tfrac{1}{N}\mathbf{X}^T\mathbf{X}, \qquad \textit{correlation matrix estimate} \tag{2.11}$$

while the cross-corralation vector, as

$$\mathbf{R}_{xd} = \tfrac{1}{N}\mathbf{X}^T\mathbf{d}, \qquad \textit{croos-correlation vector estimate.} \tag{2.12}$$

2.1.4 Performance of LMS algorithm

2.1.4.1 Learning Curve of LMS algorithm

2.1.4.2 eigenvalue and the performance

2.1.4.3 On-line vs Batch Algorithms

2.1.5 Revisiting the Algorithm of Mean and Variance Estimation

In this section, are presented in a revised form, the algorithms used in the previous sections.

Example 2.2. In §1.1 we introduced a simple program for the on-line estimation of the mean and variance of a sequence. in this example, is given a MATLAB function for on-line calculation of the mean and variance of an M-dimensional sequence of length equal to `BlkSize`.

```
function  [F, w, s] = MEAN_VAR_STRC_F_B(F, BlkSize, x)
```

```
%-------------------------------------------------------------------------
% On-line mean value and variance estimation
%
% Call as:
%     [F] = MEAN_VAR_STRC_F_B(F, BlkSize, x)
%
% The [F] STRUCT, that contains the status variables, can be created as:
%     [F] = create_online_mean_var(M)
%           F.M = M;
%           F.k = 0;
%           F.w   = zeros(M,1);
%           F.s   = zeros(M,1);
%
% Input Arguments:
%     F : FILTER STRUCT
%     x : x((k : k + BlkSize-1),:) % window of input sequence
%
% Output Arguments:
%     F :  Alg. STRUCT and results
%-------------------------------------------------------------------------
% [BlkSize, M] = size(x)  % The BlkSize and M parameters are superfluous
     for k=1:BlkSize
        F.k = F.k + 1;
        for i=1 : F.M
            dltW   = (x(k,i) - F.w(i))/F.k;
            % Run-time mean value estimation
            F.w(i) = F.w(i) + dltW;
            % Run-time variance estimation
            F.s(i) = (F.k-1)*F.s(i)/F.k + dltW*(x(k,i) - F.w(i));
        end
    end
    s=F.s; % this can be ommitted
    w=F.w; % this can be ommitted
end
% -------------------------------------------------------------------------

function  [F] = create_online_mean_var(M)
%-------------------------------------------------------------------------
% Create [F] STRUCT for on-line mean value and variance estimation
% [F] = MEAN_VAR_STRC_F_B(F, BlkSize, x)
%-------------------------------------------------------------------------
    F.M = M; % nr. of populations
    F.k = 0; % Internal samples counter
    F.w = zeros(M,1); % Run-time mean value
    F.s = zeros(M,1); % Run-time var value
end
% -------------------------------------------------------------------------
```

Example 2.3. Below, a demo porgram to verify the function `MEAN_VAR_STRC_F_B(F, BkSz, x);`

```
%-------------------------------------------------------------------------
% Naive Introduction of Adaptive Algorithms
% Demo program for on-line mean value and variance estimation
%
%-------------------------------------------------------------------------
clear all; close all;
N = 100000; % samples population
M = 5; % nr. of populations
% Random mean value and variane target --------------------------------
for i=1 : M
    w0(i) = rand(1,1); % Target random mean value [0,1]
    s2(i) = 1+rand(1,1) -0.5; % Target random variance value [0.5, 1.5]
end
% Create the structure for the on-line mean and variance estimation
[F] = create_online_mean_var(M);
```

```
% ---- Run-time stat.s estimation -------------------------------------
BkSz = 10000;
NrBlk = N/BkSz;
x = zeros(BkSz,M);
for k=1:NrBlk
    % Generate sequence
    for i=1 : M
        x(1:BkSz,i) = w0(i) + sqrt(s2(i))*randn(BkSz ,1);
    end
    % Run-time statistic computation
    [F] = MEAN_VAR_STRC_F_B(F, BkSz, x);
end
% ---- Print results -------------------------------------------------
fprintf('Estimated mean value and variance of a given data samples\n');
for i=1:M
    fprintf('Mean w0 = %f est.(w=%f) - Var s2 = %f est.(s=%f)\n',...
        w0(i),F.w(i),s2(i),F.s(i));
end
%-------------------------------------------------------------------------
```

Here an example of the output of the above demo program.

```
Estimated mean value and variance of a given data samples
Mean w0 = 0.440171 est.(w=0.435450) - Var s2 = 1.253000 est.(s=1.248357)
Mean w0 = 0.234791 est.(w=0.232685) - Var s2 = 1.279303 est.(s=1.282530)
Mean w0 = 0.077070 est.(w=0.077356) - Var s2 = 0.551211 est.(s=0.547946)
Mean w0 = 0.103048 est.(w=0.104123) - Var s2 = 0.616592 est.(s=0.617379)
Mean w0 = 0.296285 est.(w=0.296215) - Var s2 = 1.165584 est.(s=1.172470)
»
```

2.1.6 Revisiting the Least Mean Squares Algorithm

The least squares methods are the foundation of most of the data-driven techniques for data mining streams. In Sections §1.2.3 and §1.2.4, we introduced intuitively the LS basic method for batch and on-line approach to solve a simple linear (or polynomial) regression problem. While in Section §2.1.1 has been introduced the problem of adaptive filtering with LMS learing algorithm.

Example 2.4. In this example is presented a MATLAB function that implements the LMS algorithm, and the relative procedure in order to create the internal struct `[F]`.

```
function  [F, y, e] =  AF_STRC_LMS_F_B(F, BlkSz, x, d)
% ************************************************************************
% StandardLMS algorithm
%    AF_STRC_LMS_F.m
%       Implements the Complex LMS algorithm for signal sequence ()
%
%  Call as:
%  [F, y, e] =  AF_STRC_LMS_F_B(F, BlockLength, x, d)
%
% Input Arguments:
%   F     : ADAPTIVE FILTER STRUCT
%   BlkSz : Length of input signal
%   x     : x[n] input signal sample
%   d     : d[n] desired signal sample
%
```

```
% Output Arguments:
%   F : ADAPTIVE FILTER STRUCT
%   y : y[n] output signal sample
%   e : e[n] error signal sample
%
% --------------------------------------------------------------------------
% A. Uncini,"Fundamentals of Adaptive Signal Processing",
% Springer Editor, ISBN 978-3-319-02806-4, 2014.
%
%   Copyright 2013 - Aurelio Uncini
%   DIET Dept. - 'Sapienza' University of Rome
%   $Revision: 1.0$  $Date: 2013/11/25$
% --------------------------------------------------------------------------
y = zeros(BlkSz ,1);
e = zeros(BlkSz ,1);
    for n = 1 : BlkSz
        F.xw =  [x(n);  F.xw(1 : F.M-1)];  % shift input filter delay-line
        y(n)  = F.w'*F.xw;          % output by convolution
        e(n)  = d(n) - y(n);        % error
        F.w   = F.w + F.mu*conj(e(n))*F.xw; % cpx LMS eqn. (5.104)
    end
end
% --------------------------------------------------------------------------
% A. Uncini, Fundamentals of Adaptive Signal Processing
% --------------------------------------------------------------------------

function  [F] =  create_STRC_LMS_AF(M, mu)
% ************************************************************************
% function  [F] =  create_STRC_LMS_AF(M, mu )
%
% Input parameters
%      M      : filter length
%      mu     : LMS learning rate
%
%--------------------------------------------------------------------------
% A. Uncini ,"Fundamentals of Adaptive Signal Processing",
% Springer Ed.,  ISBN 978-3-319-02806-4, 2015.
%
%   Copyright 2011 - A. Uncini,
%   DIET - University of Rome 'La Sapienza' Italy
%   $Revision: 1.0$  $Date: 2011/10/16$
%--------------------------------------------------------------------------
w  = zeros(M,1);    % w AF taps
xw = zeros(M,1);    % buffer for filter status delay-line
AF = 'Direct-Form FIR Adaptive Filter struct for LMS algorithm';
F = struct('AF',AF, 'M',M, 'mu',mu, 'w',w, 'xw',xw);
%--------------------------------------------------------------------------




function  [F, y, e] = AF_STRC_LS_F_B(F, BlkSz, x, d)
% ************************************************************************
%  Demo LS algorithm (--- Naive LS filter (no MIL) ----)
%    AF_STRC_LS_F.m
%
%  Call as:
% function  [F, y, e] =  AF_STRC_LS_F_B(F, BlkSz, x, d)
%
% Input Arguments:
%   F     : ADAPTIVE FILTER STRUCT
%   BlkSz : Length of input signal
%   x     : x[n] input signal sample
%   d     : d[n] desired signal sample
%
% Output Arguments:
%   F : ADAPTIVE FILTER STRUCT
%   y : y[n] output signal sample
```

```
%   e : e[n] error signal sample
%
% ------------------------------------------------------------------------
% A. Uncini,"Fundamentals of Adaptive Signal Processing",
% Springer Editor, ISBN 978-3-319-02806-4, 2014.
%
%   Copyright 2013 - Aurelio Uncini
%   DIET Dept. - 'Sapienza' University of Rome
%   $Revision: 1.0$  $Date: 2013/11/25$
% ------------------------------------------------------------------------
    if nargin==0, help AF_STRC_LS_F_B; return; end
    y = zeros(BlkSz ,1);
    e = zeros(BlkSz ,1);
    % fill the data matrix X ------------------------------------------------
    for n = 1 : BlkSz
        F.xw =[x(n) ; F.xw(1 : F.M-1)]; % shift input filter delay -line
        F.X(n,:) = F.xw ; % fill data matrix with new input samples
    end
    F.Rxx = F.X'*F.X/BlkSz;
    F.Rxd = F.X'*d/BlkSz;
    F.w = inv(F.Rxx + F.dI)*F.Rxd;
    y = F.X*F.w;
    e = d - y;
return
% ------------------------------------------------------------------------

function  [F] = create_STRC_LS_AF(M, N, mu, delta)
% ************************************************************************
% Single channel structure for Adapive Filters AF
%   Call as:
%      [F] =  create_STRC_LS_AF(M, mu, delta)
%
% LS adaptive filter definition and initialization ----------
%    M  :     Filter length
%    mu :     StepSize; (for Steepest Descent Algorithm)
%    delta  : regularization parameter
%
% ------------------------------------------------------------------------
% A. Uncini ,"Fundamentals of Adaptive Signal Processing",
% Springer Ed.,  ISBN 978-3-319-02806-4, 2015.
%
%   Copyright 2011 - A. Uncini,
%   DIET - University of Rome 'La Sapienza' Italy
%   $Revision: 1.0$  $Date: 2011/10/16$
% ------------------------------------------------------------------------
if nargin==0, help create_STRC_LS_AF; return; end
% Algorithm matricies and buffers
dI = delta*eye(M);  % Regularizing diagonal matrix
AF = 'AF struct for LS algorithms';
w  =  zeros(M,1);   % w AF taps
xw =  zeros(M,1);   % buffer for filter status
Rxx = zeros(M, M);  % estimated correlation matrix
Rxd = zeros(M, 1);  % estimated cross-correlation vector
X = zeros(N, M);    % data matrix
F = struct('AF',AF,'M',M,'mu',mu,'dI',dI, 'w',w, 'xw',xw, ...
           'X',X, 'Rxx',Rxx,'Rxd',Rxd,'delta',delta);
% ------------------------------------------------------------------------
```

2.2 Discrete Time Linear Dynamical System Fundamentals

In general term *dynamic system*, we intend a mathematical model, associated a real physical system, that determines the relationship between a certain cause and the effect that follows from it. Considering for simplicity systems defined in discrete-time (DT), lets define X and Y metric vector spaces, denoting with $x[n] \in X$, the cause (i.e. the input of the system) at time n, and denoting with $y[n] \in Y$ the effect (i.e. the output of the system), the input/output relationship can be expressed as

$$y[n] = T\{x[n]\} \tag{2.13}$$

where T is an mathematical operator that maps elements of the input space X to elements of the output space Y, formally $T : X \to Y$. In other words, knowing the mathematical model of the physical system, we can predict its behavior (i.e. output of the system), for any stress to which it is subjected (i.e. for each input). Moreover, with the formalism (2.13), the mathematical properties of the operator T are related to the properties of the real physical system and vice-versa.

Property 2.1. An operator T, is said to be *linear* if it is worth the *superposition effects principle*, or simply *superposition principle*, defined as

$$y[n] = T\{c_1 x_1[n] + c_2 x_2[n]\} = c_1 T\{x_1[n]\} + c_2 T\{x_2[n]\} \tag{2.14}$$

in this case the cause (or output of the system), is always proportional to the effect that generated it.

Property 2.2. The operator T is *time-invariant* if the translation properties of causes and effects is true. Formally a time-invariant system is defined as

$$y[n] = T\{x[n]\} \quad \Rightarrow \quad y[n - n_0] = T\{x[n - n_0]\}. \tag{2.15}$$

Definition 2.1. A system that satisfies the two previous properties, it is said to be linear and time-invariant (LTI).

2.2.1 Impulse Response and Convolution

The impulse response, is defined as the system response when at its input is applied the unit impulse $\delta[n]$ (that is zero valued sequence except that for $n = 0$ in which it is equal to 1). This response is indicated $h[n] = T\{\delta[n]\}$.

Note that, if T is LTI, the system is fully characterized by its impulse response. For "fully characterized" means the property that if we known the input $x[n]$ and the impulse response $h[n]$, it is always possible to calculate the response $y[n]$.

In fact, from the time-invariant property if $h[n] = T\{\delta[n]\}$ then $h[n - n_0] = T\{\delta[n - n_0]\}$, it also appears that, the sequence $x[n]$ can be described as a sum shifted impulses, i.e. applying the principle of superposition, we can write that $x[n] = \sum_{k=-\infty}^{\infty} x[k]\delta[n - k]$. So, it is $y[n] = T\left\{\sum_{k=-\infty}^{\infty} x[k]\delta[n - k]\right\}$. Now, for the linearity property, it is possible to switch the T operator with the summation $y[n] = \sum_{k=-\infty}^{\infty} x[k]T\{\delta[n - k]\}$ and, finally, we can write

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k], \qquad \text{for} -\infty < n < \infty \tag{2.16}$$

The previous expression shows that, for a DT LTI system, known the impulse response and the input, the output computability is defined by the Eqn. (2.16).

The expression (2.16) is defined as *convolution sum*. This operation, very important in DT system, also from software/hardware implementation point of view, and usually is indicated as, $y[n] = x[n] * h[n]$ or , where the symbol $*$ denotes the DT convolution sum.

Remark 2.2. The discrete-time convolution is the fundamental operation of the digital filtering and, more generally, in the digital signal processing (DSP). So, without loss of generality, a discrete-time system, it is also said numeric or digital filter.

Remark 2.3. A DT LTI dynamical system may have an impulse response of finite or infinite duration. If the impulse response has a finite duration, the circuit is called *finite impulse response* (FIR) filter. If the impulse response has instead an unlimited duration the circuit is called *infinite impulse response* (IIR) filter.

2.2.2 The z-Domain Transfer Function of DT Circuits and Systems

Lets define $z \in \mathbb{C}$ a complex variable, the Z-transform of an impulse response $h[n]$ (or of a generic sequence), for is defined by the following equations pair

$$H(z) = Z\{h[n]\} \triangleq \sum_{n=-\infty}^{\infty} h[n]z^{-n} \qquad \textit{direct Z-transform} \tag{2.17}$$

$$h[n] = Z^{-1}\{H(Z)\} \triangleq \frac{1}{2\pi j}\oint_C H(z)z^{n-1}dz \qquad \textit{inverse Z-transform} \tag{2.18}$$

Thus, you can see that the $H(z)$ is an infinite power series in the complex z variable, where the sequence $x[n]$ plays the role of the series coefficients.

The periodic frequency response of $h[n]$ can be found in the region defined by by $z = e^{i\omega}$, $-\pi \leq \omega \leq \pi$, and $z = e^{i\omega}$, $-\pi \leq \omega \leq \pi$, which is the *unit circle* of the z-plane.

The z-transfer functions are often used to verify the frequency response and other properties of dynamical systems like, for example, the stability of IIR filter.

Definition 2.2. For a DT LTI system the *transfer function* (TF) $H(z)$, is defined as the z-domain ratio between the output and input, i.e.,

$$H(z) \triangleq \frac{Y(z)}{X(z)} \tag{2.19}$$

For the previous definition we have that the output of a DT LTI system can be computed as $Y(z) = H(z)X(z)$. This fundamental result for linear systems is known as the convolution theorem: for a DT LTI system with impulse response $h[n]$, in which input is present in a sequence $x[n]$, is subject to the relations

$$y[n] = h[n] * x[n] \quad \Leftrightarrow \quad Y(z) = H(z)X(z)$$

and

$$y[n] = h[n]x[n] \quad \Leftrightarrow \quad Y(z) = H(z) * X(z)$$

that is, the convolution in the time domain is equivalent to multiplication in the frequency domain and vice-versa. For the proof and more details see, for example, [2], [3].

2.2.3 LTI Dynamic Systems with Finite Impulse Response

The dynamic systems that have impulse response of finite duration, usually referred to as FIR filters, are very important in many relevant applications and technologies.

An FIR filter is characterized by a finite length impulse response defined as

$$\mathbf{h} \triangleq \big[\, h[0]\ h[1] \cdots h[M-1] \,\big]^T$$

the convolution operation (2.16), can be written as

$$y[n] = \sum_{k=0}^{M-1} x[k]h[n-k], \qquad \text{for } n = 0, 1, \ldots. \tag{2.20}$$

The z-transform of a FIR filter is defined as

$$H(z) = h[0] + h[1]z^{-1} + ... + h[M-1]z^{-M-1}$$

2.2.3.1 On-line Convolution as Inner Product Vectors

Let's define the vector

$$\mathbf{x} \in \mathbb{R}^{M\times 1} = \big[\, x[n]\ x[n-1] \cdots x[n-M+1] \,\big]^T$$

representing the M-length window input signal at time n, $n-1$, ..., $n-M+1$, the convolution can be written as

$$y[n] = \mathbf{h}^T\mathbf{x} = \mathbf{x}^T\mathbf{h}, \qquad \text{for } n = 0, 1, \ldots\ . \tag{2.21}$$

Thus the convolution can be seen as the dot product between the impulse response vector and the M-length window sequence over the input signal.

The expression (2.21) the describes a FIR filter can be implements as *delay-line* which implements the selection of M-length window sequence over the input signal, that is followd by a linear combiner that implements the scalar product. The related circuit, said numerical time-domain convolver, is illustrated in Fig. 2.4.

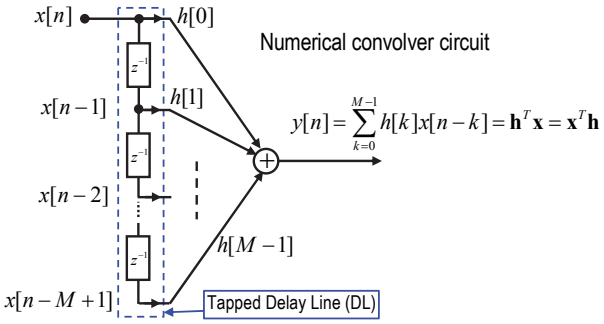

Fig. 2.4 An example of *convolver circuit*, that implements a $(M-1)$-order FIR filter. It is composed by a delay-line mechanism followed by a linear combiner. (Figure from [2]).

2.2.3.2 Convolution as a Product, Data-Matrix Impulse-Response Vector

In the case that the impulse response and the input signal are both sequences of finite duration the expression (2.16) can be interpreted as a matrix vector product. Let $x[n]$, $0 \leq n \leq N-1$, and $h[n]$, $0 \leq n \leq M-1$ with $M < N$, it follows that the output $y[n]$ has length equal to $L = N + M - 1$. Arranging the samples of the impulse response and the output in column vectors, the system output, reinterpreting (2.16), can be written as

$$\begin{bmatrix} y[0] \\ \vdots \\ \vdots \\ y[M-1] \\ \vdots \\ \vdots \\ y[N-1] \\ \vdots \\ \vdots \\ y[L-1] \end{bmatrix} = \begin{bmatrix} x[0] & 0 & \cdots & 0 \\ x[1] & x[0] & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ x[M-1] & x[M-2] & \cdots & x[0] \\ x[M] & x[M-1] & \cdots & x[1] \\ \vdots & \vdots & \ddots & \\ x[N-1] & x[N-2] & \cdots & x[N-M] \\ 0 & x[N-1] & \cdots & x[N-M+1] \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & x[N-1] \end{bmatrix} \begin{bmatrix} h[M-1] \\ h[M-0] \\ \vdots \\ h[0] \end{bmatrix} \tag{2.22}$$

or, in a more compact way, as

$$\mathbf{y} = \mathbf{X}\mathbf{h} \tag{2.23}$$

where the $(L \times M)$ is defiend as *data-matrix* $\mathbf{X}$ contains the samples of the input signal arranged in columns, gradually shifted down of a one sample.

Note that the first and the last $M-1$ rows of the matrix contain zeros due to the signal transient, by consequence, the first and the last $M-1$ output samples are characterized by a so-called transient effect.

2.2.3.3 Convolution as a Product, Convolution-Operator-Matrix Input Sequence

The relation (2.16) can also be written in the notation

$$\mathbf{y} = \mathbf{H}\mathbf{x} \tag{2.24}$$

where $\mathbf{H} \in \mathbb{R}^{(M+N-1)\times N}$ is defined *convolution operator matrix*, the matrix containing the shifted impulse response $\mathbf{h}$ replicas, filled with zeros, as indicated in (2.24), such that the vectors $\mathbf{x}$ and $\mathbf{y}$ contain, respectively, the input and output window samples

$$\begin{bmatrix} y[0] \\ \vdots \\ \vdots \\ y[M-1] \\ \vdots \\ \vdots \\ y[N-1] \\ \vdots \\ \vdots \\ y[L-1] \end{bmatrix} = \begin{bmatrix} 0 & 0 & \cdots & \cdots & \cdots & 0 & h[M-1] \\ \vdots & \cdots & \cdots & \cdots & 0 & h[M-1] & h[M-2] \\ \vdots & \iddots & \iddots & \iddots & \iddots & \iddots & \iddots \\ \vdots & \cdots & \cdots & 0 & h[M-1] & \cdots & h[0] \\ \vdots & \cdots & 0 & h[M-1] & \cdots & h[0] & 0 \\ \vdots & 0 & \iddots & \iddots & \iddots & \iddots & \vdots \\ 0 & h[M-1] & \cdots & \cdots & \cdots & \cdots & 0 \\ h[M-1] & h[M-2] & \cdots & h[1] & h[0] & 0 & 0 \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ \vdots \\ x[N-1] \end{bmatrix} \tag{2.25}$$

Note that the matrices $\mathbf{X}$ and $\mathbf{H}$, that appear in the convolution expressions (2.24) and (2.25), have identical elements on the diagonals i.e. are diagonal-constant matrices. More formally, let $[a_{i,j}]$ the elements of the matrix $\mathbf{A}$, we have that $[a_{i,j}] = [a_{i+1,j+1}]$, this particular matrix is defined as Toeplitz matrix and is very important for developing fast algorithms.

2.2.3.4 Software design of FIR filtering procedure

According to the general scheme of Fig. 2.4, it is important to observe that the delay-line, and its relative update mechanism, should be within the circuit structure. In other words the circuit (i.e. the algorithm) is an *object* that contains all is needed to implement the filtering operation. So, this object must be created before to call the filtering operation.

Example 2.5. Below, it is shown a simple MATLAB function that implements an FIR filter with on-line, mini-batch and batch processing capabilities.

```
function  [F, y] =  FIR_STRC_F_B(F, BlkSize, x)
% ************************************************************************
% On-line FIR filter implementation algorithm
%
%  Call as:
%      [F, y] =  FIR_STRC_F_B(F, BlkSize, x)
%
%   Equivalent to Matlab  function:
%      [y, F.xh(1 : F.M-1)] = filter(F.h, 1, x, F.xh(1 : F.M-1));
%
% ------------------------------------------------------------------------
```

```
    y = zeros(1,BlkSize);
    for n=1 : BlkSize
        F.xh = [x(n); F.xh(1 : F.M-1)]; % shift input delay-line and load
                                        % new input into filter
        y(n)  = F.xh'*F.h;              % output by linear combination
    end
return
% --------------------------------------------------------------------------
```

The `struct` (or record) `[F]`, contains all is needed to implements the algorithm that in this case are: 1) the filter impulse response; 2) the delay-line vector. Below, is reported a simple function `create(h)` that can be used in order to create the struct `[F]`.

```
% Create a FIR filter structure for on-line filtering
function   [F] =  create_FIR_F(h)
h = h(:);
M = length(h); % ORD - 1
F = struct( 'M',M, 'h',h, 'xh',zeros(M,1) );
%--------------------------------------------------------------------------
```

2.2.4 LTI Dynamic Systems with Infinite Impulse Response

In the class of causal DT LTI system, great practical importance have the circuit that satisfies the *finite difference equation* (FDE) of order p, i.e. of the type

$$\sum_{k=0}^{p} a_k y[n-k] = \sum_{k=0}^{q} b_k x[n-k], \qquad a_k, b_k \in \mathbb{R}, \quad p \geq q \tag{2.26}$$

For $a_0 = 1$, the above expression can be written in the normalized form

$$y[n] = \sum_{k=0}^{q} b_k x[n-k] - \sum_{k=1}^{p} a_k y[n-k] \tag{2.27}$$

that appears to be characterized by a useful circuit representation shown in Fig. 2.5. From the definition (2.19), the TF of (2.26), appears to be a rational function of the type

$$H(z) = \frac{\sum\limits_{k=0}^{q} b_k z^{-k}}{\sum\limits_{k=0}^{p} a_k z^{-k}} = \left(\frac{b_0}{a_0}\right) \frac{\prod\limits_{k=1}^{q} \left(1 - c_k z^{-1}\right)}{\prod\limits_{k=1}^{p} \left(1 - d_k z^{-1}\right)} \tag{2.28}$$

Therefore, the indices (p,q) represent, respectively, the maximum degree of the polynomial in the TF's numerator and denominator. Please note that it can sometimes be convenient to indicate the summation of such as (M,N) representing the delay-lines length (see Fig. 2.5) and it is obvious that in this case is $M = q+1$ and $N = p+1$.

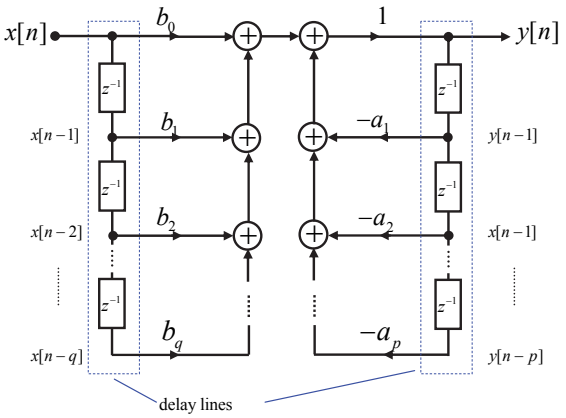

Fig. 2.5 An example of circuit representation of finite difference equation for $a_0 = 1$. (Figure from [2]).

As for the physical realizability the FDE order is expressed by the degree of the denominator of the TF (2.28).

2.2.4.1 Software design of IIR filtering procedure

In the following, a simple MATLAB function that implements a IIR filtering operation designed according to the scheme of Fig. 2.5. The transfer function of the filter is defined as in Eqn. (2.28), which corresponds to a finite difference equation of the type $a_0y[n] = b_0x[n] + b_1x[n-1] + \ldots + b_qx[n-q] - a_1y[n-1] - a_2y[n-2] - \ldots - a_py[n-p]$ and where usually $a_0 = 1$.

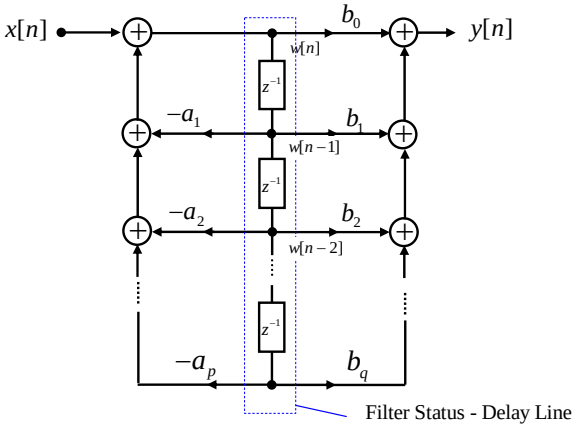

Fig. 2.6 Signal flow graph of a Direct-Form-II implementation of a p=q-order IIR digital filter. The single delay-line $w[n]$, $w[n-1]$, $\cdots$, elements represent the *filter status* which are are shared with the p-poles and q-zeros sections.

In our procedure the filter is implemented in the so called Direct-Form-II (DF-II), and signal flow graph for DF-II realization of the IIR filter is shown in Fig. 2.6. The corresponding difference equation for the DF-II structure can be written in a two-pass

procedure as

$$w[n] = x[n] - a_1 w[n-1] - a_2 w[n-2] - \ldots - a_p w[n-p]$$
$$y[n] = b_0 w[n] + b_1 w[n-1] + \ldots + b_q w[n-q]$$

from the above expression we have that at index time n, the filter output can be calculated by considering the input $x[n]$ and the filter status $w[n]$.

Example 2.6. The resulting procedure must be contains the values of transfer function parameter **a** and **b**. Using MATLAB language, all filter parameters can be memorized a structure (or record) `[F]` that must be created before the use of the filtering procedure. The record `[F]` contains, also, the hidden status of the filter **w** and this procedure can be use in batch, mini-batch and on-line mode, and can be create using the following function.

```
% Create a IIR filter structure for on-line filtering
function   [F] =  create_IIR_F(b, a)
P = length(a); % P = p + 1
Q = length(b); % Q = q + 1
M   = max(P,Q);
w   = zeros(M,1);
if (a(1) ~= 1) % coeff.s normalization
    b = b./a(1);
    a = a./a(1);
end
F = struct('Q',Q, 'P',P, 'M',M, 'w',w, 'a',a, 'b',b);
%-------------------------------------------------------------------------
```

In the following, is reported the MATLAB function `IIR_STRC_F_B(F,BLKSIZE,x)`, that implements the IIR shown in Fig. 2.6 in on-line, batch and mini batch mode.

```
function [F, y] = IIR_STRC_F_B(F, BlkSize, x)
% ************************************************************************
% On-line, mini-batch and batch IIR filter in "Direct Form II"
% implementation of the standard difference equation:
%
%    a(1)*y(n) = b(1)*x(n) + b(2)*x(n-1) + ... + b(Q)*x(n-Q+1)
%                          - a(2)*y(n-1) - ... - a(P)*y(n-P+1)
%
% If a(1) is not equal to 1, the filter coefficietns are normalized by a(1)
%
%    Equivalent to Matlab function:
%          [y, F.z] = filter(F.b, F.a, x, F.z );
% -------------------------------------------------------------------------
 %  [y, F.z] = filter(F.b, F.a, x, F.z );
    y = zeros(1,BlkSize);
    for n = 1 : BlkSize
        F.w(1) = x(n); % load input signal into delay-line (F.a(1)=1)
        for i = 2 : F.P
            F.w(1) = F.w(1) - F.a(i)* F.w(i); % input MAC
        end
        y(n) = F.b(1)*F.w(1);
        for i = 2 : F.Q
            y(n) = y(n) + F.b(i)* F.w(i); % output MAC
        end
        F.w(2:F.M) = F.w(1:F.M-1); % status delay-line shift
    end
return
% -------------------------------------------------------------------------
```

In the following, is reported a test program of the abowe IIR filtering function, in comparison with the standard MATLAB routine `filter(a,b,x)`.

```
%  Testing IIR_STRC_F_B(F, BlkSize , x) vs filter(a,b,x) ------------------
N = 20;
x = randn(N,1); % generate random signal
a = [1.9   -0.1 0.3 ]';  % A(z) filter coefficients
b = [0.8    1    0.8 ]';  % B(z) filter coefficientts
F = create_IIR_F(b, a); % create the filter struct H(z) = B(z)/A(z)
%
% --- batch IIR filtering ------------------------------------------------
[F, y0] = IIR_STRC_F_B(F, N, x);

% --- on-line IIR filtering ----------------------------------------------
F.w(:) = 0; % reset filter status
for n=1 : N
    [F, y1(n)] = IIR_STRC_F_B(F, 1 , x(n));
end

% --- mini-batch IIR  filtering ------------------------------------------
F.w(:) = 0; % reset filter status
BlkSize = 5;
for n=1: BlkSize : N
    rw_ind = n : n + BlkSize - 1; % running window start/end indices
    [F, y2(rw_ind)] = IIR_STRC_F_B(F, BlkSize , x(rw_ind));
end
%
% *** Test standard Matlab IIR filtering routines -------------------------
%
% --- batch filter standard matlab routine -------------------------------
y3 = filter(b,a, x);
%
% --- filter standard matlab routine  on line ----------------------------
w = zeros( max(length(a),length(b))-1,  1); % filter status
for n=1 : N
    [y4(n), w] =  filter(b, a, x(n), w);
end

% --- mini-batch IIR  filtering ------------------------------------------
w(:) = 0; % reset filter status
BlkSize = 5;
for n=1: BlkSize : N
    rw_ind = n : n + BlkSize - 1; % running window start/end indices
    [y5(rw_ind), w] = filter(b, a, x(rw_ind), w);
end
%---- Print results -------------------------------------------------------
for n=1 :  N
   fprintf('y[%2d]=%6.3f  %6.3f  %6.3f  %6.3f  %6.3f  %6.3f\n', ...
       n, y0(n), y1(n), y2(n), y3(n), y4(n), y5(n) );
 end
%--------------------------------------------------------------------------
```

Here an example of the output of the above demo program.

```
 » ES_TEST_IIR_filters
y[ 1]= 0.127 0.127 0.127 0.127 0.127 0.127
y[ 2]=-0.258 -0.258 -0.258 -0.258 -0.258 -0.258
y[ 3]=-0.178 -0.178 -0.178 -0.178 -0.178 -0.178
y[ 4]=-0.059 -0.059 -0.059 -0.059 -0.059 -0.059
y[ 5]= 0.103 0.103 0.103 0.103 0.103 0.103
....
```

2.3 Proposed Problems

Exercise 2.1. In Digital Signal Processing (DSP), for the determination of the frequency response of a filter with impulse response $h[n]$, we use the discrete-time Fourier transform (DTFT) defined as:

$$H(e^{j\omega}) = \sum_{k=-\infty}^{\infty} h[k]e^{-j\omega k}.$$

1. Prove that the DTFT, is a special case of the z-transform. (search on the internet, for example using Wikipedia);
2. determine the expression of the inverse DTFT;
3. determine the expression of Discrete Fourier Transform (DFT);
4. find a MATLAB procedure for fast computation of DFT (fft algorithm).

Exercise 2.2. Write a MATLAB program to generalize the function `FIR_STRC_F_B(F, BLKSIZE, x)`; for filtering multidimensional signal. Suggestion: before you write a multi-channel DSP filtering procedure, write a function that creates the structure `[F]` in which the filters' impulse responses are given as a matrix $\mathbf{h} \in \mathbb{R}^{P\times M}$ where P represents the number of channels and M their length supposed identical for all filters.

Exercise 2.3. Write a MATLAB program to generalize the function `FIR_STRC_F_B(F, BLKSIZE, x)`; for filter bank of P channel. Suggestion: 1) the impulse responses of the filters bank given as a vector $\mathbf{h} \in \mathbb{R}^{P\cdot M}$, where M is the filter length supposed identical for all filters of the bank.

Exercise 2.4. Write a MATLAB program to generalize the function `FIR_STRC_F_B(F, BLKSIZE, x)`; for Multi Input an Multi Output filtering. Suggestion: 1) impulse responses of the MIMO system are given as a matrix $\mathbf{h} \in \mathbb{R}^{Q\times P}$ where P and Q represent, respectively, the number of input and output channels, and M is the filter length supposed identical for all filters. 2) Consider the MIMO system as a Q filters bank with P inputs.

Exercise 2.5. Write a MATLAB program to generalize the function `IIR_STRC_F_B(F, BLKSIZE, x)`; for filtering multidimensional signal.

Exercise 2.6. Modify the MATLAB adaptive filtering program proposed in Section §2.1.1 using the function `function [F, y, e] = AF_STRC_LMS_F_B(F, BlkSz, x, d)`.

Chapter 3
A Naïve Approach to Time Series Analysis and Adaptive Filters Applications

The growing availability of technologies to capture and store information from disparate sources, makes more and more relevant the need to have algorithms able to extract information from such *historical data*. In this direction, we can define the *time series analysis*, as the set of theoretical and computational methods with the aim to understand the dynamics of the observed phenomenon known in terms of time series.

3.1 Introduction of Time Series Analysis

The analysis of time series are used in signal processing, in pattern recognition, econometrics, mathematical finance, weather, intelligent transport systems, prediction of trajectories, earthquake prediction, analysis of biomedical signals, control engineering, communications engineering, social networks traffic information, big-data analytics, content analysis and retrieval, and largely in any field of applied science and engineering that involves temporal measurements [19]-[22].

The main objectives of the analysis of time series are

1. Time series prediction and forecasting.
2. Signal detection and estimation.
3. Regression analysis.

The time series prediction and forecasting is principally used in the context of statistics, econometrics, quantitative finance, seismology, meteorology, and geophysics. The primary goal of time series analysis is forecasting, i.e. the determination of a model which is fed with the past and noisy data, that is able to predict a future outcome. For signal detection and estimation, we intend the recovery of signal that is corrupted by noise, and it is suitable in the context of signal processing, control engineering and communication engineering. The regression analysis is often used to test whether the current values of a time series are influenced by another time series

Note that in the context of data mining, pattern recognition and machine learning, the time series analysis can be used for clustering, classification, query by content, anomaly detection as well as forecasting.

3.1.1 Autoregressive and Moving Average Time Series Model

In modern study of stochastic systems, a very powerful paradigm, useful for characterization of certain types of time series, is to consider the time series as output of a linear time invariant filter, the input of which is fed by white noise.

Given a sequence of zero-mean and σ_η^2 variance white Gaussian noise (WGN) indicated as $\eta[n] \in \mathcal{N}(0, \sigma_\eta^2)$, considered as input of a linear time invariant IIR filter whit output $x[n]$, we have that the transfer function is

$$H(z) = \frac{X(z)}{\eta(z)} = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + \cdots b_q z^{-q}}{1 + a_1 z^{-1} + \cdots a_p z^{-p}} \tag{3.1}$$

the general scheme for the generation of linear random sequence is shown in Fig. 3.1. Thus, in discrete time domain, it is defined as the output of a model characterized by the following finite difference equation

$$x[n] = -\sum_{k=1}^{p} a_k x[n-k] + \sum_{k=0}^{q} b_k \eta[n-k]. \tag{3.2}$$

where, depending on the context, the filter coefficients are indicated as a_k, b_k.

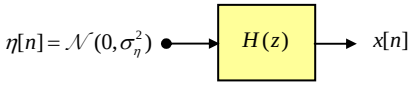

Fig. 3.1 Block diagram for generating a linear random discrete time stochastic process $x[n]$.

Notet that, in vector form the previous expression can be written as $x[n] = \mathbf{a}^T \mathbf{x}_{n-1} + \mathbf{b}^T \boldsymbol{\eta}$, where $\mathbf{a} = [\, a_1 \, \cdots \, a_p \,]^T$, $\mathbf{b} = [\, b_0 \, \cdots \, b_q \,]^T$, and $\boldsymbol{\eta} = [\, \eta[n] \, \cdots \, \eta[n-q+1] \,]^T$.

If $H(z)$ has poles and zeros the generation filter is denoted as autoregressive moving average (ARMA) model. Denoting by q and p, respectively, the degree of the polynomial at numerator and at the denominator of the transfer function (3.1), the model is indicated as ARMA(p, q). In addition, we can consider the following cases:

1. $p > 0$, $q > 0$, autoregressive moving-average ARMA(p,q) model;
2. $p > 0$, $q = 0$, autoregressive AR(p) model;
3. $p = 0$, $q > 0$, moving-average MA(q) model.

For $z = e^{j\omega}$, the power spectral density (PSD), is defiend as

$$\begin{aligned} R_{xx}(e^{j\omega}) &= \sigma_\eta^2 \left| H(e^{j\omega}) \right|^2 \\ &= \sigma_\eta^2 \frac{\left| b_0 + b_1 e^{-j\omega} + b_2 e^{-j2\omega} + \cdots + b_q e^{-jq\omega} \right|^2}{\left| 1 + a_1 e^{-j\omega} + a_2 e^{-j2\omega} + \cdots + a_p e^{-jp\omega} \right|^2}. \end{aligned}$$

So, using this parametric representation, the PSD of the ARMA(p,q) sequence is entirely determined by the transfer function polynomial coefficients. Thus, the deter-

mination of such coefficients is equivalent to the PSD estimation. The models AR, MA or ARMA are widely used in time-series analysis, such as in many contexts: the analysis and synthesis of signals, signals compression, time-series classification, econometrics, etc.

Remark 3.1. The finite difference equation (3.2) can be implemented as presented in Section §2.2.4.1. In addition, in MATLAB the Eqn. (3.2) corresponds to the function `y = filter(b,a,x)`, that filters the data in vector `x` with the filter described by vectors *and* to create the filtered data `y`. The `filter` is a "Direct Form II Transposed" implementation of the standard difference equation (3.2).

3.1.2 Autoregressive Time-Series Generation

For $q = 0$ the Eqn. (3.2) can be written as the following difference equation

$$x[n] = -\sum_{k=1}^{p} a[k]x[n-k] + b\eta[n] \tag{3.3}$$

The previous expression defines a p-order autoregressive time-series model AR(p) with a frequency response equal to

$$H(e^{j\omega}) = \frac{1}{A(z)} = \frac{1}{1+\sum_{k=1}^{p} a_k e^{-j\omega k}}.$$

In practice, the filter has p poles, and a multiple zero of order p at the origin. The power spectral density (PSD) of the process is equal to

$$R_{xx}(e^{j\omega}) = \frac{\sigma_\eta^2}{\left|1+\sum_{k=1}^{p} a[k]e^{-j\omega k}\right|^2}. \tag{3.4}$$

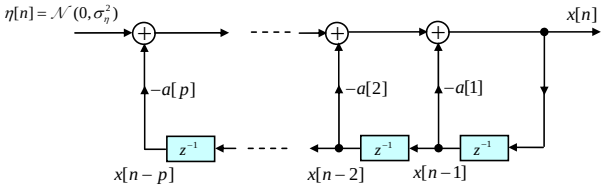

Fig. 3.2 Discrete-time circuit for the generation of a linear autoregressive random sequence.

3.1.2.1 Generating a First Order Autoregressive Process with a given variance

Consider a first-order AR process in which, for simplicity assume $a = -a[1]$. The corresponding equation finite difference is defined as

$$x[n] = ax[n-1] + b\eta[n], \quad n \geq 0, \quad x[-1] = 0. \tag{3.5}$$

Taking the expectation of the square of both sides we obtain

$$\begin{aligned} E\{x^2[n]\} &= E\{(ax[n-1] + b\eta[n])^2\} \\ &= E\{a^2x^2[n-1] + 2b\eta[n]ax[n-1] + b^2\eta^2[n]\} \\ &= E\{a^2x^2[n-1]\} + 2E\{ax[n-1]b\eta[n]\} + E\{b^2\eta^2[n]\}. \end{aligned}$$

For the white Gaussian noise properties, we have that $E\{x[n-1]\eta[n]\} = 0$. So, the previous expression can be written as

$$E\{x^2[n]\} = a^2E\{x^2[n-1]\} + b^2E\{\eta^2[n]\}.$$

Moreover we have that $\sigma_x^2 = E\{x^2[n]\} = E\{x^2[n-1]\}$, and we can write

$$b^2\sigma_\eta^2 = (1-a^2)\sigma_x^2$$

Thus, to generate a signal $x[n]$ with variance equal to that of the input noise, it is sufficient to have $b^2 = 1-a^2$, i.e.

$$b = \sqrt{1-a^2}. \tag{3.6}$$

Remark 3.2. For performance evaluation of adaptive algorithms, usually, are used narrow-band and unit variance stochastic processes. Very often, the test sequences are created with (3.5) for *values* very close to one i.e. $0 \ll a < 1$ and $b = \sqrt{1-a^2}$.

Example 3.1. In this example, we want to write a program in MATLAB language, for the generation of a narrow band, first-order Markov process with unitary variance. From Eqn. (3.6), we have that

$$x[n] = ax[n-1] + \sqrt{1-a^2}\eta[n], \quad n \geq 0, \quad x[-1] = 0. \tag{3.7}$$

Here's a simple MATLAB code that implements the previous equation

```
%--------------------------------------------------------------------------
%  Generation a AR stochastic process
%--------------------------------------------------------------------------
clear all;  close all;
% IIR filter coefficients ------------------------------------------------
a = 0.95;
%
% Generation of N samples of WGN sequence --------------------------------
N = 10000;
eta = randn(N,1);
% Generation of N samples of MA sequence ---------------------------------
x = filter( sqrt(1 - a^2),[1 -a], eta);
fprintf('Results:\n');
fprintf('Input var = %f,    Output var = %f \n', var(eta), var(x) );
%--------------------------------------------------------------------------
```

Below, we can find the output of the program run.

```
Results:
Input var = 1.017054, Output var = 1.038046
```

Remark 3.3. The acf of the sequence generated by the (3.7) is equal to $r[k] = \sigma_\eta^2 a^k$. for $k = 0,\ 1,\ ...,\ M$. The autocorrelation matrix is defined as

$$\mathbf{R}_{xx} = \sigma_\eta^2 \begin{bmatrix} 1 & a & a^2 & \cdots & a^{M-1} \\ a & 1 & a & \cdots & a^{M-2} \\ a^2 & a & 1 & \cdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & a \\ a^{M-1} & a^{M-2} & \cdots & a & 1 \end{bmatrix}.$$

For the case $M = 2$, the condition number of $\mathbf{R}_{xx}$, that is defined as the ratio of maximum and minimum eigenvalue, can be determined as

$$p(\lambda) = \det \begin{bmatrix} 1-\lambda & a \\ a & 1-\lambda \end{bmatrix} = \lambda^2 - 2\lambda + (1-a^2).$$

So,

$$\lambda_{1,2} = 1 \pm a.$$

Thus, the condition number of the correlatiobn matrix, that is defines as the ratio $\chi(\mathbf{R}_{xx}) = \frac{\lambda_{\max}}{\lambda_{\min}}$, is given by

$$\chi(\mathbf{R}_{xx}) = \frac{1+a}{1-a}. \tag{3.8}$$

To analyze the behavior of the adaptive algorithms in extreme conditions, it is possible to generate a process with a predetermined value of the condition number. In fact, solving the latter for a, we get

$$a = \frac{\chi(\mathbf{R}_{xx}) - 1}{\chi(\mathbf{R}_{xx}) + 1}. \tag{3.9}$$

3.1.3 Moving Average Time Series Generation

For $p = 0$ the Eqn. (3.2), can be written as the following difference equation

$$x[n] = \sum_{k=0}^{q} b[k]\eta[n-k] \tag{3.10}$$

with transfer function

$$H(z) = B(z) = b_0 + b_1 z^{-1} + \cdots b_q z^{-q}.$$

Usually, to indicate the coefficients of the filter, instead of the symbols $b[n]$, it is used the symbol $h[n]$, more familiar to indicate the finite-length impulse response of the filter. The coefficients of the filter $h[n]$ are called moving average parameters.

In the analysis of time series, this model is used when it is assumed the output signal $x[n]$ depends on the current and past input samples (the model has memory) with a linear relationship. Moreover, note that the expression is a discrete-time convolution. Therefore, MA time-series is a discrete-time stochastic process which consists of WGN filtered by a FIR filter with, usually unknown, impulse response **h**. The random

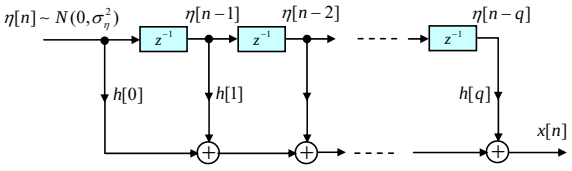

Fig. 3.3 Signal flow graph of the discrete-time circuit generating a linear random discrete time stochastic process $x[n]$.

sequence generated in this way, is defined as *moving average linear stochastic process* or simply *MA process*.

Remark 3.4. For the *Wold representation*or *decomposition* theorem (see [2], pag. 682), says that every stationary time series $x[n]$ can be written as the sum of two time series, one deterministic and one stochastic. For simplicity, in Eqn. (3.1.3) the deterministic term of the sequence has been considered zero.

3.1.3.1 Generating a Moving Average process with a given frequency-band and variance

As a first example, suppose we want generate a MA stochastic process with the same variance of the white noise η at the input of the linear filter (see Fig. 3.3.

$$\begin{aligned} E\{x^2[n]\} &= E\left\{\left(\sum_{k=0}^{q} h[k]\eta[n-k]\right)^2\right\} \\ &= E\left\{\sum_{k=0}^{q} h^2[k]\eta^2[n-k] + \text{ cross-products }\right\}. \end{aligned} \tag{3.11}$$

Now, we have that $E\{\eta[n]\eta[n-k]\} = 0$, for all $k \neq 0$; thus the expectation of the all cross-product terms is zero. In addition, we have that $E\{\eta^2[n]\} = E\{\eta^2[n-k]\}$. So, the previous expression can be written as

$$E\{x^2[n]\} = \sum_{k=0}^{q} h^2[k]E\{\eta^2[n-k]\}$$

or, in other terms, as

$$\sigma_x^2 = \sigma_\eta^2 \cdot \sum_{k=0}^{q} h^2[k] = \sigma_\eta^2 \cdot \|\mathbf{h}\|_2^2 \tag{3.12}$$

for which, in order that the filter has a unitary gain, it is sufficient to normalize its coefficients as $\mathbf{h} \leftarrow \mathbf{h}/\|\mathbf{h}\|$.

Example 3.2. In this example we wont to generate a MA process with a frequency band specified by a given FIR filter. The impulse response of the filter is defined as: $\mathbf{h} = [-0.0712\ 0.3080\ -0.3571\ 0\ 0.3571\ -0.3080\ 0.0712]$, which corresponds to the frequency response in the Fig. 3.4.

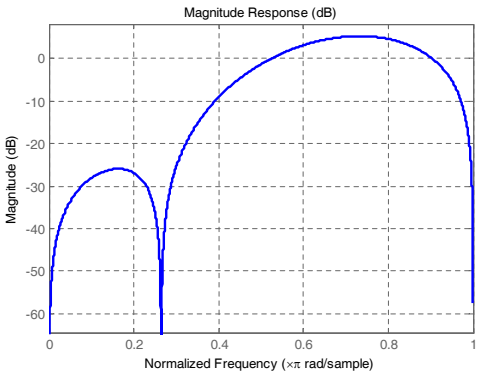

Fig. 3.4 Frequency response of the FIR filter $\mathbf{h}$ plotted with the MATLAB program `fvtool(h)`.

Considering a WGN input $\eta \in \mathcal{N}(0,1)$, i.e. with unit variance, in order to have a MA process $x[n]$ with unitary variance, for Eqn. (3.12), it is sufficient to impose a filter gain equal to $G = 1/\sqrt{\mathbf{h}^T\mathbf{h}}$.

Here's a simple MATLAB code.

```
%--------------------------------------------------------------------------
%  Generation a MA stochastic process with a given frequency-band
%  and variance
%
%   Copyright 2016 - A. Uncini
%   DIET Dpt - University of Rome 'La Sapienza' - Italy
%   $Revision: 1.0$  $Date: 2015/03/11$
%--------------------------------------------------------------------------
clear all;  close all;
% FIR filter  generation with a given frequency response ------------------
h0 = firls(6,[0 .4 .5 1],[0 0 1 1],'h')';
G = 1/norm(h0);

% Impulse response normalization ------------------------------------------
h =  G*h0;

% Generation of N samples of WGN sequence ---------------------------------
N = 10000;
eta = randn(N,1);

% Generation of N samples of MA sequence ----------------------------------
x = filter (h,1,eta);
fprintf( ...
'--------------------------------------------------------------------------\n');
```

```
fprintf('Results:\n');
fprintf('Filter gain  G= %f \n', G );
fprintf('Input var = %f,    Output var = %f \n', var(eta), var(x) );
fprintf( ...
'--------------------------------------------------------------------------\n');
%--------------------------------------------------------------------------
```

Below, we can find the output of the program run.

```
Results:
Filter gain G= 1.482551
Input var = 1.019199, Output var = 0.989201
```

3.2 Moving-Average System Identification

As indicated in the previous session, a generic time-series can be thought as the output of a dynamic system when its input is fed with white Gaussian noise. If the dynamic system is linear, time-series generative model is described by the finite-difference equation (3.2).

In time-series analysis, one of the most interesting issues is to estimate its generative model. The problem of the generation-model identification is very important in many areas of great strategic interest.

In this section, it addresses the problem of identification of the generative model of given time-series in the case where: 1) it is assumed known that the model is of type MA; 2) the model order is known.

3.2.1 Least Squares Parameters Estimation of a MA(q) Time-Series

To define the data-driven identification method, we consider the scheme of Fig. 3.5, where the $M=(q+1)$-length vector $\mathbf{w}_0 \in \mathbb{R}^{M\times 1}$ is the target, i.e. the system impulse response to be identified, $x[n]$ represents the input signal that usually is WGN with unitary variance, and $\upsilon[n]$ is the measurements or observation noise. Usually the measurements noise is zero-mean WGN $\upsilon[n]=\mathcal{N}(0,\sigma_\upsilon^2)$, and its variance is set in order to have a prescribed signal-to-noise ratio (SNR). Proceeding intuitively, the identification problem can be addressed as outlined in the polynomial regression (see 1.2.2). Assuming that the length of the sequence is equal to N, the observations can indicated by the pairs that defines our *training set*,

$$\mathbf{x}_d = \big[x[n], d[n]\big]_{n=0}^{N-1}$$

and where the sequence $d[n]$, denoted as *desired signal*, is the output of the target system $\mathbf{w}_0$

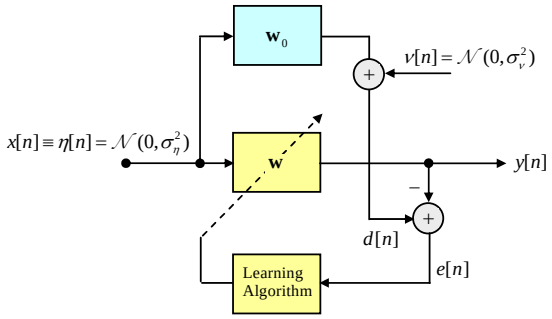

Fig. 3.5 Block diagram to describe the data-driven approach for identification of the generative model of linear time-series $y[n]$.

$$d[n] = \mathbf{w}_0^T \mathbf{x} + \upsilon[n]$$

where $\mathbf{x}$ indicated the vector of a M-length window on the input signal defined as

$$\mathbf{x} \in \mathbb{R}^{M\times 1} = [x[n]\ x[n-1]\ \cdots\ x[n-M+1]]^T.$$

In order to determine the coefficients $\mathbf{w} \in \mathbb{R}^{M\times 1}$ we minimize the residual error between the desrired signal $d[n]$ and the output of the filter $\mathbf{w}$. So, lets define $y[n]$ the output of the filter at time n as

$$y[n] = \mathbf{w}^T \mathbf{x}$$

the error is defined as

$$e[n] = d[n] - y[n].$$

The square error over each input pairs can be written as

$$\begin{aligned} J(\mathbf{w}) = e^2[n] &= \left(d[n] - \mathbf{w}^T\mathbf{x}\right)^2 \\ &= d^2[n] - 2\mathbf{w}^T\mathbf{x} + \mathbf{w}^T\mathbf{x}\mathbf{x}^T\mathbf{w} \end{aligned} \tag{3.13}$$

which define the loss function $J(\mathbf{w})$ as in Eqn. (1.19), i.e. as the square error over each input pairs.

Example 3.3. Consider the problem of MA time series parameters estimation as shown in Fig. 3.5. Lets $\mathbf{w}_0 = [1\ -2]^T$ be, the model to be identified is

$$d[n] = w_0[0]x[n] + w_0[1]x[n-1] + \upsilon[n]$$

where the input signal $x[n]$ is zero-mean Gaussian white noise (WGN) with variance $\sigma_x^2 = 1$ and $\upsilon[n]$ is zero-mean WGN that characterizes the measurement uncertainty that for this experiment has a variance $\sigma_\upsilon^2 = 0.1$ (i.e. noise level equal to -10 [dB]).

Example 3.4. Here's a simple MATLAB code for LMS and LS moving average parameters estimation.

```
%--------------------------------------------------------------------------
%  Naive Introduction of Least Squares Method: Identification of
%  Moving Average (MA) stochastic process
%
%   Copyright 2013 - A. Uncini
```

```
%   DIET Dpt - University of Rome 'La Sapienza' - Italy
%   $Revision: 1.0$  $Date: 2013/03/09$
%-------------------------------------------------------------------------
clear all; close all;
%
% Moving Average N samples time-series generation -------------------------
N  = 10000; % number of samples ( Fs = 1.0 Hz )
w0 = [1 -2]';  x = randn(N,1);
% --- Desired data generation + Additive WGN (AWGN) ------------------------
Snr_dB  = 10; % Noise level in dB
noise_std_dev = 10^(-Snr_dB /20); % Convert [dB] in natural value
d = filter( w0, 1, x ) + noise_std_dev*randn(size(x));
%
fprintf('Estimated variance of input signal,   var(x) = %f\n', var(x));
fprintf('Estimated variance of desired signal, var(d) = %f\n', var(d));
%
M   = length(w0);   % filter length
mu = 0.001;
[LMS] =  create_STRC_LMS_AF(M, mu);
BkSz = 10;
% ************************************************************************
% on-line vs mini-batch LMS solution
% ************************************************************************
for n=1 : BkSz : N
    nn = n : n + BkSz-1;
    [LMS, y(nn), e(nn)] = AF_STRC_LMS_F_B(LMS, BkSz, x(nn), d(nn));
    w1(nn)= LMS.w(1); % memorize weight for weight's trajectory plot
    w2(nn)= LMS.w(2); %
end

% --- Error plot -------------------------------------------------------
figure;
plot(e);
grid on; box on;
ylim([-8 8]);
title('Error of LMS algorithm');
xlabel('Samples');
ylabel('{\ite}[{\itn}]');

% --- Weights' trajectories plot ---------------------------------------
figure;
grid on;
hold on; box on;
plot(w1,'color','b', 'LineWidth',2);
plot(w2,'color','r', 'LineWidth',2);
title('Weights trajectory');
xlabel('Samples');
ylabel('{\itw}_0[{\itn}], {\itw}_1[{\itn}]');

% --- Mean Squares Error plot in [dB], Learning Curve plot ---------------
figure;
LineWidth = 2;
MSE_Bound(1:N) = -Snr_dB;
MSE_dB = 20*log10(2*abs(e));
hold on;
grid on; box on;
[b,a]  = butter(2,0.01);  % butterworth smooth filter
Smooth_MSE_dB = filtfilt(b,a, MSE_dB);
plot(Smooth_MSE_dB,  'LineWidth',LineWidth);
plot(MSE_Bound, '--', 'color','k', 'LineWidth',LineWidth);

title('Mean Squares Error ');
xlabel('Samples');
ylabel('20*log_1_0(abs({\ite}))');
legend('MSE','MSE bound');


% ************************************************************************
% LS solution
```

```
% ************************************************************************
[LS]  =  create_STRC_LS_AF(M, N, 1, 1e-5);
[LS,  y2, e2] = AF_STRC_LS_F_B(LS, N, x, d);

fprintf('Estimated correlation matrix:');
LS.Rxx
fprintf('Estimated cross-orrelation vector');
LS.Rxd

fprintf('Estimated LMS solution');
LMS.w

fprintf('Estimated LS solution');
LS.w
% LMS energy error
fprintf(...
'Variance of the residual error: %f, variance of added noise: %f\n', ...
var(e(5000:N)), 10^(-Snr_dB/10) );
% Plot the estimated performance surface
plot_Jw( [-2 4 ], [-5 1] , 50,  var(d), LS.Rxd, LS.Rxx);
% ------------------------------------------------------------------------
```

Below, the program exit

```
Input var = 1.017054, Output var = 1.038046
Estimated variance of input signal, var(x) = 0.984905
Estimated variance of desired signal, var(d) = 4.984923
Estimated correlation matrix:
Rxx =
0.9848 0.0077
0.0077 0.9848

Estimated cross-orrelation vector
Rxd =
0.9672
-1.9603

Estimated LMS solution
w =
1.0027
-1.9901

Estimated LS solution
wopt =
0.9976
-1.9983

Variance of the residual error: 0.102138, variance of added
noise: 0.100000
```

Finally, Fig. 3.6 shows the behavior of the error, while the Fig. 3.7 the trend of the trajectories of the weights and the performance surface.

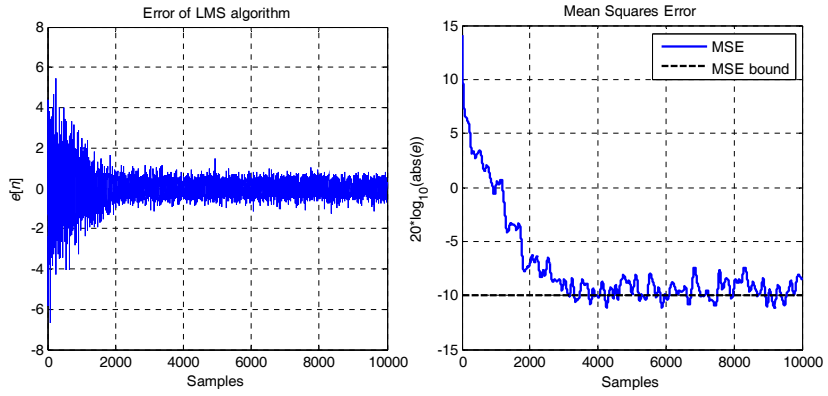

Fig. 3.6 Performance of LMS algorithm: left) plot of error $e[n]$; right) plot of MSE error in [dB].

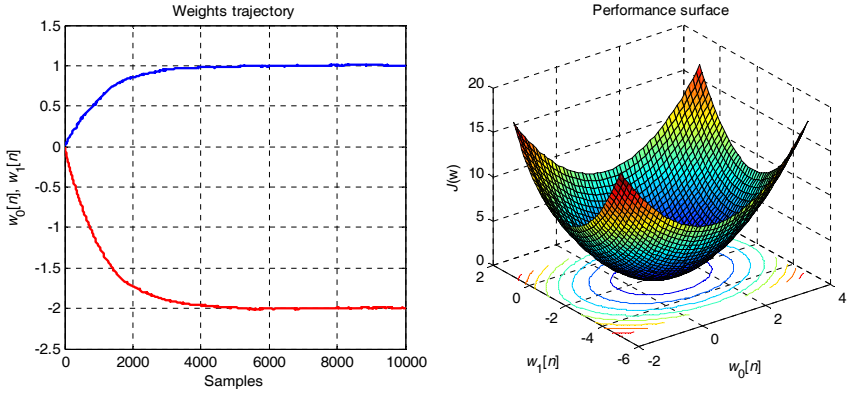

Fig. 3.7 Performance of LMS algorithm: left) weights $w_0[n]$ and $w_1[n]$ trajectories; right) Performance surface.

3.3 Forward Linear Prediction

The estimation process of a certain quantity related to future time (or otherwise in a different domain) is a problem that has always affected philosophers and scientists. With the term *prediction* is indicated the estimate, considering a possible accepted error, of an event starting from the knowledge of the some observations related to it.

When the prediction relates to the estimation of a future event, when they are known some past events, the problem is called *time series forecasting*. In the following the term "prediction", it is used to indicate also to indicate the estimate of future samples of a time series.

The *prediction system* or *predictor* is a *parametric operator* which at its input accepts a certain number of past samples, and at its output produces an estimate of one or more future samples. More formally (see [2], pag. 68), denoting as $\hat{x}[n]$ the estimated predicted value, we have

$$\hat{x}[n] = T\left\{x[n-1],\ x[n-2],\ ...,\ x[n-M], \mathbf{w}^f\right\} \tag{3.14}$$

where the superscript "^{f}" stands for *forward* prediction.

In the case that the operator $T\{\cdot\}$ is linear, it follows that the predicted sample is a linear combination of the past samples. Lets define $\mathbf{x}_{n-1} \in \mathbb{R}^{M\times 1} = [x[n-1]\ x[n-2]\ \cdots\ x[n-M]\ ]^T$, the expression (3.14) can be written as

$$\hat{x}[n] = w^f[1]x[n-1] + w^f[2]x[n-2] + \ \cdots\ + w^f[M]x[n-M]$$

and the above equation indicates that the predicted value depends linearly from past M samples of the sequence $x[n]$.

Now considering the presence of a certain *prediction error*, that can be defined as the difference between the 'true' value $x[n]$ and the predicted value $\hat{x}[n]$, form the above expression we can write

$$\begin{aligned} e[n] &= x[n] - \hat{x}[n] \\ &= x[n] - \mathbf{w}^{fT}\mathbf{x}_{n-1}. \end{aligned} \tag{3.15}$$

The last expression is equivalent to the scheme reported in (3.8).

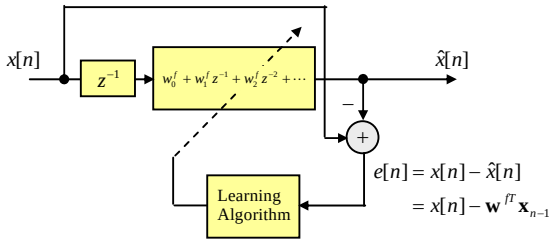

Fig. 3.8 One-step forward linear prediction scheme.

In addition, lets define the vector $\mathbf{a} = [1\ \ -w^f[1]\ \ -w^f[2]\ \cdots\ -w^f[M]]^T$ and a vector $\mathbf{x} = [\hat{x}[n]\ \ \mathbf{x}_{n-1}]^T$, we have that the Eqn. (3.15) can be rewritten as

$$e[n] = \mathbf{a}^T\mathbf{x}. \tag{3.16}$$

3.3.1 Linear Prediction: Theoretical Wiener Analysis

For the theoretical analysis of the predictor we proceed to the MSE minimization as

$$J(\mathbf{w}^f) = E\left\{|e[n]|^2\right\}$$

so, from Eqn. (3.15), we can write

$$J(\mathbf{w}^f) = E\left\{\left(x[n] - \mathbf{w}^{fT}\mathbf{x}_{n-1}\right)\left(x[n] - \mathbf{w}^{fT}\mathbf{x}_{n-1}\right)\right\}$$

developing the square, we get

$$\begin{aligned} J(\mathbf{w}^f) &= E\left\{x^2[n] - 2\mathbf{w}^{fT}\mathbf{x}_{n-1}x[n] + \mathbf{w}^{fT}\mathbf{x}_{n-1}\mathbf{x}_{n-1}^T\mathbf{w}^f\right\} \\ &= E\left\{x^2[n]\right\} - 2\mathbf{w}^{fT}E\left\{\mathbf{x}_{n-1}x[n]\right\} + \mathbf{w}^{fT}E\left\{\mathbf{x}_{n-1}\mathbf{x}_{n-1}^T\right\}\mathbf{w}^f. \end{aligned}$$

Now remember that $\sigma_x^2 = E\{x^2[n]\}$, denoting as $\mathbf{g} = E\{\mathbf{x}_{n-1}x[n]\}$, and remember that by definition $\mathbf{R} = E\{\mathbf{x}_{n-1}\mathbf{x}_{n-1}^T\}$, we have

$$J(\mathbf{w}^f) = \sigma_x^2 - 2\mathbf{w}^{fT}\mathbf{g} + \mathbf{w}^{fT}\mathbf{R}\mathbf{w}^f \tag{3.17}$$

that represents the quadratic performance surface of the linear predictor of order M.

The gradient of Eqn. (3.17) is $\nabla J(\mathbf{w}^f) = -2\mathbf{g} + 2\mathbf{R}\mathbf{w}^f$ and, setting $\nabla J(\mathbf{w}^f) \to 0$, the optimal Wiener solution is $\mathbf{w}_{opt}^f = \mathbf{R}^{-1}\mathbf{g}$.

3.3.2 Linear Prediction of Autoregressive Stochastic Process: The Prediction Error Filter

As an first example of prediction suppose that the observed stochastic process $x[n]$ is a p-order autoregressive process

$$x[n] = -a_1x[n-1] - a_2x[n-1] - \cdots - a_px[n-p] - +\eta[n]$$

where $\eta[n]$ is a zero-mean WGN whit a given variance. The AR(p) transfer function is then

$$H(z) = \frac{X(z)}{\eta(z)} = \frac{1}{A(z)} = \frac{1}{1 + a_1z^{-1} + \cdots + a_pz^{-p}}.$$

In Fig. 3.9-a), as example, is reported an AR process generation, followed by the prediction scheme. The the output of the predictor can be written as

$$\hat{x}[n] = w[0]x[n-1] + w[1]x[n-2] + \ \cdots \ + w[p-1]x[n-p]$$

now considering the scheme of Fig. 3.9-b) the relationship between the input $x[n]$ and the prediction error $e[n]$, can be written according to the equation

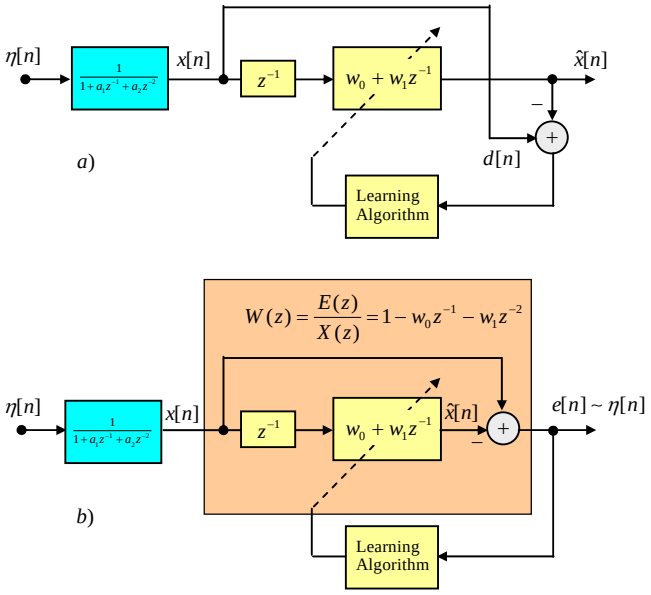

Fig. 3.9 Prediction of autoregressive AR(2) process (the generalization to AR(p) is straightforward). a) One-step forward predictor scheme; b) Equivalent configuration but as error prediction scheme.

$$e[n] = x[n] - w[0]x[n-1] - w[1]x[n-2] - \cdots - w[p-1]x[n-p] \tag{3.18}$$

then, the transfer function between the input $x[n]$ and the error $e[n]$ is defined as

$$W(z) = \frac{E(z)}{X(z)} = 1 - w[0]z^{-1} - \cdots - w[p-1]z^{-p}.$$

still considering the Fig. 3.9-b), the overall transfer function between the white noise input $\eta[n]$ and the output error $e[n]$, is equal to

$$\begin{aligned} H(z) &= \frac{E(z)}{X(z)} \cdot \frac{X(z)}{\eta(z)} = W(z) \cdot \frac{1}{A(z)} \\ &= \frac{1 - w[0]z^{-1} - w[1]z^{-2} - \cdots - w[p-1]z^{-p}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_p z^{-p}}. \end{aligned}$$

Thus, the 'best predictor', such that $\hat{x}[n] \sim x[n]$, is the transfer function $W(z)$ such that $W(z)/A(z) = 1$, i.e. $e[n] \sim \eta[n]$ and, with a simple visual inspection, we have that the optimal predictor filter coefficients are $w_{opt}[i-1] = -a_i$, for i=1, 2, … p.

Note that for the above reasoning, the transfer function $W(z)$ is denoted as *prediction error filter*.

Example 3.5. Below a possible modification of the program shown in Example 3.4, for the study of the linear prediction problem of a given autoregressive sequence.

```
. . .
% Autoregressive N-samples time-series generation -------------------------
N  = 10000; % number of samples ( Fs = 1.0 Hz )
```

```
% --- Input Signal Generation WGN ----------------------------------------
NoiseLev_dB    =  -10; % Noise level in dB
noise_std_dev =  10^(NoiseLev_dB /20); % Convert [dB] in natural value
% AR(2) coefficients -------------------------------------------------------
a1 = 0.8;
a2 = 0.1;
% --- Implements the AR Model --------------------------------------------
x = zeros(N+1, 1);
x = filter(1,[1 -a1 -a2],noise_std_dev*randn(N+1, 1));
% --- Desired data generation  -------------------------
d = [x( 2 :(N+1) )];
%
. . .
```

In Fig. 3.10 and Fig. 3.11, are reported the results of a linear prediction experiment of an AR(2) stoschastic process with $a_1 = 0.8$ and $a_2 = 0.1$ for the other quantity see the MATLAB code.

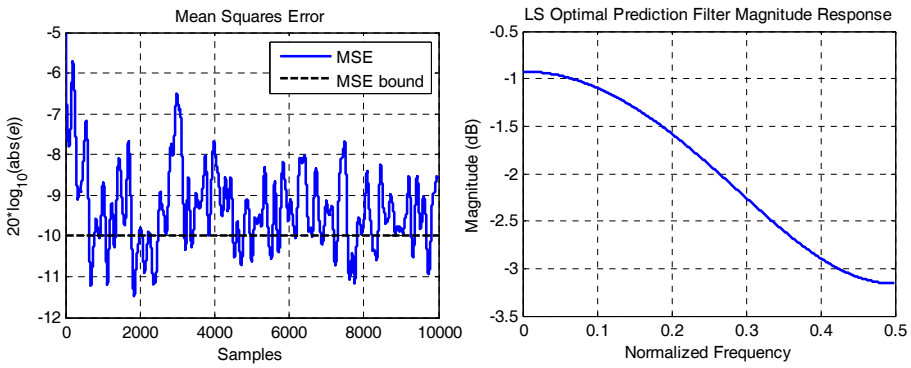

Fig. 3.10 Prediction of an AR(2) stochastic process, LS approach with a prediction filter length $M = 1$. Left) Learning curve of the LMS algorithm; Right) Frequency response of the LS prediction error filter.

Below, the program exit.

```
Estimated correlation matrix:
Rxx =
0.4722 0.4191
0.4191 0.4721

Estimated cross-orrelation vector
Rxd =
0.4191
0.3822
```

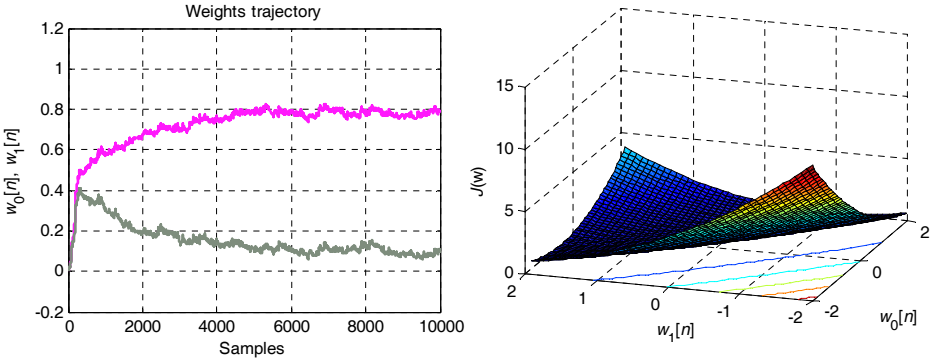

Fig. 3.11 Prediction of an AR(2) stochastic process, LS approach with a prediction filter length $M = 1$. Left) weights $w_0[n]$ and $w_1[n]$ trajectories. Right) Performance surface.

```
Estimated LMS solution
w =
0.7885
0.1015

Estimated LS solution
wopt =
0.7971
0.1020

Variance of the residual error: 0.099248, variance of the input
noise: 0.100000
```

3.3.3 Linear Prediction of Sinusoidal Sequence with a Superimposed White Gaussian Noise

As a second example of prediction suppose that the observed stochastic process $x[n]$ is a sum of N_s sinusoidal sequence (with unknown frequency) with a superimposed Gaussian white noise. Therefore, it follows that the stochastic process model to predict, is equal to

$$x[n] = \sum_{i=1}^{N_s} A_i \cos(2\pi f_i n + \varphi_i) + \eta[n], \quad n = 1,\ 2,\ ...,\ N$$

where A_i indicates the amplitudee, φ_i the phases, and where the signal frequencies is related to a unitary sampling frequency (i.e. for the sampling theorem the maximum

frequency contained in the signal is equal to 0.5 [Hz]), for which $f_i \in (0, 0.5)$. The term $\eta[n]$, is a zero-mean WGN whit a given variance.

To designate the predictor must then determines:

- the type of algorithm (online or batch);
- determine the order of the prediction filter.

The selection of the predictor order, is a not easy problem. In general, we proceed by taking advantage of the available a priori knowledge (if any) on the observed stochastic process. For example, some physical law underlying of the observations, or (as in our case) we known that the signal of interest in a sum of sinusoids.

In our case, we known that the signal of interest is sinusoidal, and as suggested in [19], the the minimum length FIR predictor is given by $M = 2N_s - 1$.

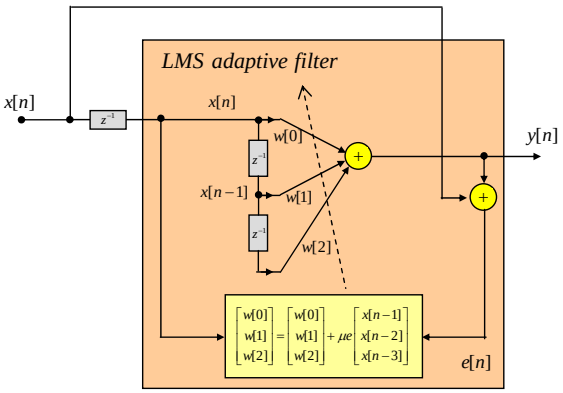

Fig. 3.12 On-line prediction implemented by the LMS algorithm.

A possible scheme of a predictor, implemented with the LMS algorithm, is shown in Fig. 3.12.

Example 3.6. Below a possible modification of the program shown in Example 3.4, for the study the linear prediction problem of sinusoidal signals.

```
 . . .
%  Sum of Sinusoid + WGN Signal Generation --------------------------------
N  = 10000; % number of samples ( Fs = 1.0 Hz )
A = 1.41; phi = pi/3; omega = 2*pi*0.1;
% --- Additive WGN std_dev ----------------------------------------------
NoiseLev_dB    = -10; % Noise level in dB
noise_std_dev =  10^(NoiseLev_dB /20); % Convert [dB] in natural value
%
x  = zeros(N+1, 1);
for n=1 : N + 1
    x(n) = A * cos(omega*n + phi) +  A * cos(2*omega*n + phi) ...
           + noise_std_dev*randn(1,1);
end
% --- Desired data generation (one step haed) ----------------------------
d = [x( 2 :(N+1) )];
%
. . .
```

In Fig.3.13 are reported the results of a linear prediction experiment of sinusoidal signals with superimposed white noise. The signal is composed of two sinusoids at frequency $f_1 = 0.1$ and $f_2 = 0.2$, of identical amplitude is equal to $A = 1.41$. The superimposed noise is equal to -10 dB, so its standard deviation is equal to $\sigma_\eta = 10^{-10/20}$. The length of prediction filter is equal to $M = 50$.

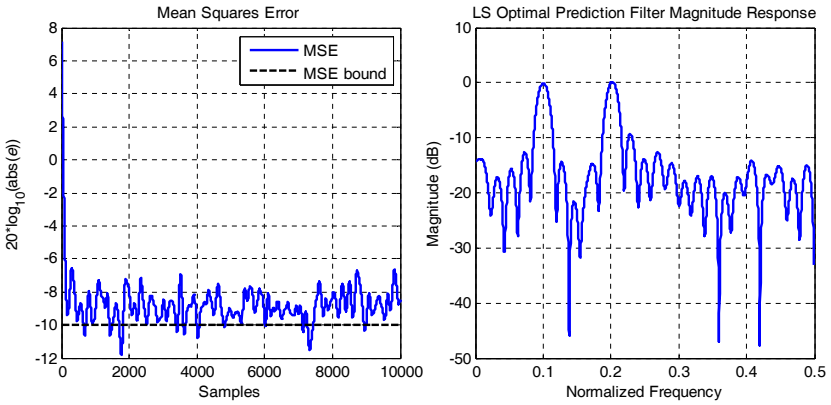

Fig. 3.13 Results of sinusoidal prediction problem by LS approach with a prediction filter length $M = 50$. Left) Learning curve of the LMS algorithm; Right) Frequency response of the LS optimal filter.

3.4 Inverse Modeling

Another adaptive filtering application that is important in many important sectors, is related to the estimation of the inverse of the physical model. This problem also known as *inverse filtering* and, for the linear case, we also refer to *deconvolution*. Given a certain system, the estimation of the inverse model can be made by inserting the adaptive circuit upstream or downstream of the system itself.

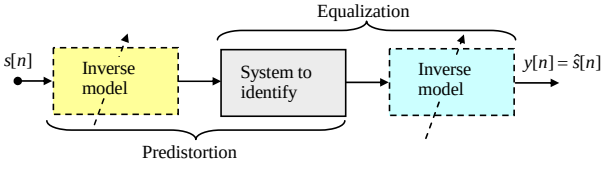

Fig. 3.14 Inverse model identification. The downstream or upstream estimation schemes for inverse modeling estimation.

Considering the scheme in Fig. 3.14, in the downstream case filter that performs the inverse estimation model is said *equalizer* while, in the case in which the estimate is made with the filter placed upstream of the system, the filter is said *predistorter*.

3.4.1 Communication Channel Equalization

One of the main applications of adaptive filtering methods in communication technology, concerns the equalization of the communication channel. In fact, it is well known that in the case of digital transmissions, the effect of the distortion (usually consider only the linear ones) due to the transmission channel, determines a change of the transmitted pulse shape causing the phenomenon of so called *intersymbol interference* (ISI), resulting in a drastic increase of the *bit error rate* (BER) at the receiver side.

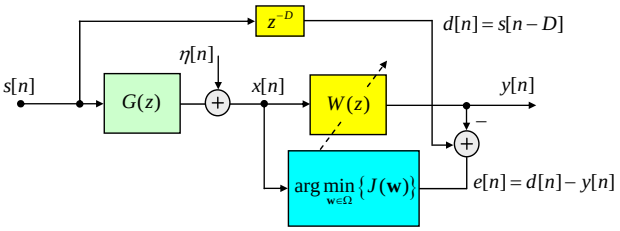

Fig. 3.15 Inverse model identification. Example of channel equalization.

In the transmission of signals, the generic communication channel (wired or wireless) can be modeled as a FIR or IIR filter with a transfer function that is here indicated as $G(z)$. In addition, in the wireless communications, in the presence of fading channel or in the case of radio links of mobile devices, the filter that models the transmission channel has time-varying nature.

The problem of communication channel equalization can be formalized as illustrated in Fig. (3.15). For example in the case of multiphat interference, the channel is modeled as an L taps FIR filter $g[n]$, for n=0, 1, $L-1$. So, the channel TF $G(z)$ is defined as $G(z) = g[0] + g[1]z^{-1} + \ \ldots \ + g[L-1]z^{-L+1}$. While in other case we my consider IIR filter model.

Let $s[n]$ the transmitted symbol, considering a "base-band" model, the received signal can be modeled as

$$x[n] = g[0]s[n] + \sum_{k=1}^{L-1} g[i]s[n-k] + \eta[n] = \mathbf{g}^T\mathbf{s} + \eta[n] \tag{3.19}$$

where the term $\sum_{k=1}^{L-1} g[i]s[n-k]$ represents the contribute of ISI which describes interference superimposed to the symbol $s[n]$ which depends on the past transmitted symbols $s[n-1]$, $s[n-2]$, $\cdots$, that should be eliminated from the equalizer, and where $\eta[n]$ is the additive white Gaussian noise.

Example 3.7. Lets $g[n] = e^{j\pi/4}[0.3\ 0.8\ 0.3]$ be the impulse response of transmission channel model. In the first experiment, we consider a transmitted signal $s[n]$ that is 4 PSK (*phase-shift keying*) modulated symbols.

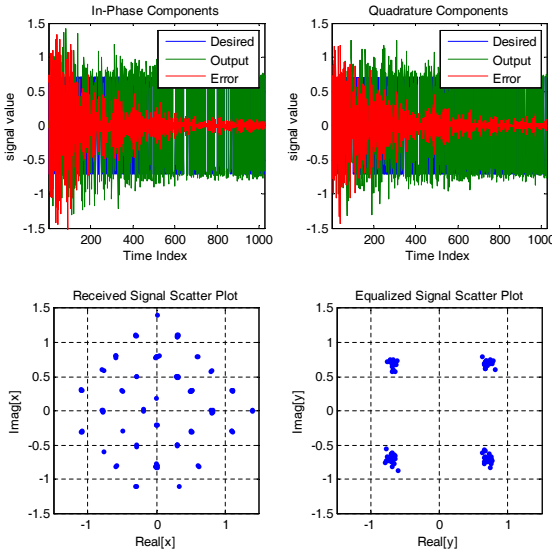

Fig. 3.16 Signals and scatter plot for equalization problem with transmission channel impulse response: $g[n] = e^{j\pi/4}$. [0.3 0.8 0.3], when the transmitted signal is a 1024-length preamble of 4-PSK (*phase-shift keying*) modulated symbols.

The simulation program of channel transmission and equalization is reported in the following MATLAB code

```
%-------------------------------------------------------------------------
%  Channel equlization by complex LMS algorithm
%-------------------------------------------------------------------------
close all; clear all;
nSamples =1024;    % No of transmitted symbols (preamble length)
M = 31;            % Equalizer length
D = (M+1)/2;       % Target signal delay

% --- Generate xxQAM signal ------------------------------------------------
s = xGen_QAM_Symbols(4, nSamples + D)';

% --- Channel transfert function (un-comment/comment to change ch. model)
%b = exp(1i*pi/4)*[-0.7 1];  % Channel #1
%a = [1 -0.7];
b  =  exp(1i*pi/4)*[0.3 0.8 0.3];  % Channel #2
a  = 1;

% --- Channel transfert function implementation + AWGN --------------------
r  = filter(b,a,s) + 0.01*(randn(nSamples+D,1) + 1i*randn(nSamples+D,1));

% --- delayed input signal and desired preamble ----------------------------
x  = r(1+D:nSamples+D);
d  = s(1:nSamples);

% --- Creation of equalizer ad equalization process ------------------------
[F] =  create_STRC_LMS_AF(32, 0.04);
[F, y, e] = AF_STRC_LMS_F_B(F, nSamples , x, d);

% --- plot results ---------------------------------------------------------
```

```
rxy = 1.5;
subplot(2,2,1); plot(1:nSamples, real([d, y, e]));
axis([1 nSamples -rxy rxy]);
legend('Desired','Output','Error'); title('In-Phase Components');
xlabel('Time Index'); ylabel('signal value');
subplot(2,2,2); plot(1:nSamples,imag([d, y, e]));
axis([1 nSamples -rxy rxy]);
legend('Desired','Output','Error'); title('Quadrature Components');
xlabel('Time Index'); ylabel('signal value');
%
subplot(2,2,3); plot(x(nSamples-100: nSamples),'.');
axis([-rxy rxy -rxy rxy]);
title('Received Signal Scatter Plot'); axis('square');
xlabel('Real[x]'); ylabel('Imag[x]'); grid on;
subplot(2,2,4); plot(y(nSamples-100: nSamples),'.');
axis([-rxy rxy -rxy rxy]);
title('Equalized Signal Scatter Plot'); axis('square');
xlabel('Real[y]'); ylabel('Imag[y]'); grid on;
%-------------------------------------------------------------------------

function [Sq_QAM] = xGen_QAM_Symbols(nQAM, nSamples)
% ************************************************************************
% Generation of 4-16-64 QAM Symbols
%    xGen_QAM_Symbols
%
%  Call as:
%     s = xGen_QAM_Symbols(nQAM, nSamples);
%
% -------------------------------------------------------------------------
% A. Uncini,"Fundamentals of Adaptive Signal Processing",
% Springer Editor, ISBN 978-3-319-02806-4, 2015.
%
%   Copyright 2016 - Aurelio Uncini
%   DIET Dept. - 'Sapienza' University of Rome
%   $Revision: 1.0$  $Date: 2016/03/24$
% -------------------------------------------------------------------------
if nargin==0, help xGen_QAM_Symbols; return; end
% -------------------------------------------------------------------------
    if nQAM == 4  % 4-PSK symbols
        LUT4QAM(1) = (1 +  1i)/sqrt(2.0);
        LUT4QAM(2) = (1 -  1i)/sqrt(2.0);
        LUT4QAM(3) = (-1 + 1i)/sqrt(2.0);
        LUT4QAM(4) = (-1 - 1i)/sqrt(2.0);
        for n = 1 : nSamples
            Sq_QAM (n)= LUT4QAM(randi(4,1,1));
        end
    end
% -------------------------------------------------------------------------
    if nQAM == 16
        k=-3;
        for i=1 : 4
            LUT16QAM(i)=k/(3.0*sqrt(2.0));
            k=k+2;
        end
        for n = 1 : nSamples
            Sq_QAM (n)= LUT16QAM(randi(4,1,1)) + 1i*LUT16QAM(randi(4,1,1)) ;
        end
    end
% -------------------------------------------------------------------------
    if nQAM == 64
        k=-7;
        for i=1 :8
            LUT64QAM(i)=k/(7.0*sqrt(2.0));
            k=k+2;
        end
        for n = 1 : nSamples
            Sq_QAM (n)= LUT64QAM(randi(8,1,1)) + 1i*LUT64QAM(randi(8,1,1));
        end
    end
```

```
end
%--------------------------------------------------------------------------
```

The result of the simulation, in the case that the transmitted signal is 1024-length preamble of 4-PSK (*phase-shift keying*) modulated symbols, is reported in Fig. 3.16

Example 3.8. Lets $G(z) = e^{j\pi/4} \cdot \frac{-0.7+z^{-1}}{1+0.7z^{-1}}$ be the transfer function of transmission channel. Now we consider a transmitted signal taht is $s[n]$ is 16 QAM (*quadrature amplitude modulation*) modulated symbol. The simulation code is that presented in the previous example (see the relative commented line).

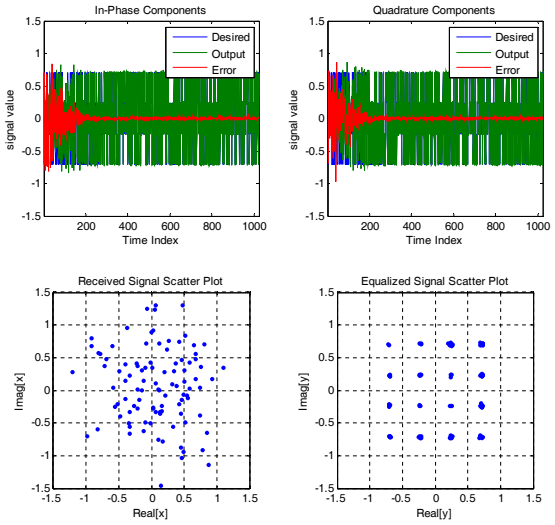

Fig. 3.17 Signals and scatter plot for equalization problem with transmission channel modeled by a transfer function: $G(z) = e^{j\pi/4} \cdot \frac{-0.7+z^{-1}}{1+0.7z^{-1}}$, when the transmitted signal is a 1024-length preamble of 16-QAM modulated symbols.

3.4.2 Predistortion by Filtered-X LMS Algorithm

Let $S(z)$ an unknown linear system, also called *secondary path*, the problem of *predistortion* consists in the estimation of its inverse model $W(z)$, also called *controller*, when it is located upstream with respect to the secondary path. In the case of linear transfer function the estimation of predistorter can be solved is similar way of the equalization problem [23].

Considering the diagram of Fig. 3.18-a), we want to estimate the controller transfer function such that $W(z) \to 1/S(z)$, for the commutative property we have that $W(z)S(z) = S(z)W(z) = 1$. In the case where the transfer function $S(z)$ is *a priori* known, for the estimate of $W(z)$ is sufficient to use the scheme of Fig. 3.18-b), and in the normal operation re commute the two transfer functions.

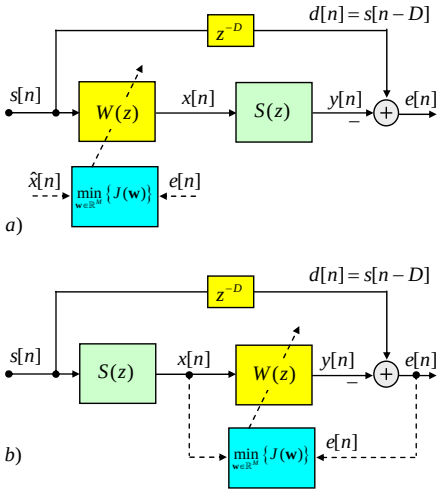

Fig. 3.18 Inverse modeling of secondary path $S(z)$. In the case of linear transfer function the upstream estimation schemes can be solved by switching the two transfer functions. a) inverse modeling by upstream predistortion schme; b) inverse modeling by downstream equalization scheme.

Usually the transfer function $S(z)$ is unknown, and it is necessary to proceed to an estimate $\hat{S}(z)$. When the secondary path is easy accessible, and presumably stationary, its estimate can be made before the adaptation process (or calibration process). In this case, the estimates of $W(z)$ can be made, by LMS like algorithms, with the diagram in Fig. 3.18-b), using the secondary path estimation $\hat{S}(z)$.

In many practical cases, the secondary path is not simply accessible and also it can be a nonstationary system. So, for its estimate we may proceed with an on-line approach as, for example, that in the scheme shown in Fig. 3.19, denoted as *filtered-x least mean squares* (FX-LMS). In FX-LMS the input signal is filtered with the estimated $\hat{S}(z)$ (from which the name "filtered-x"), in this way it is produced the signal $\hat{x}[n]$ such that the adaptation of $W(z)$ can be traced to the switched scheme illustrated in Fig. 3.18-b). Thus in FX-LMS the on-line learning rule is

$$\mathbf{w}_{new} = \mathbf{w}_{old} + \mu \hat{\mathbf{x}} e[n].$$

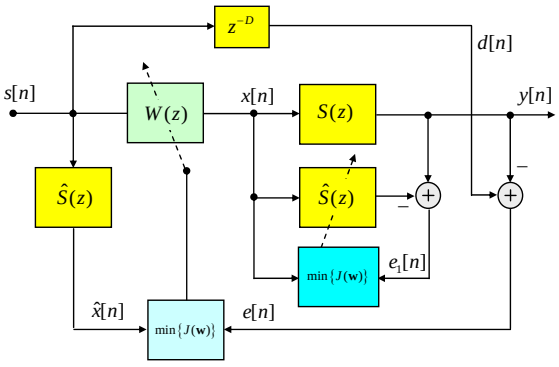

Fig. 3.19 Block diagrams of FX-LMS algorithm for $S(z)$ system compensation by predistortion. The filter $W(z)$ is adapted considering the input signal $\hat{x}[n]$ and the output error $e[n]$.

Example 3.9. Below is a MATLAB code that implements the FX-LMS algorithm, in the case that we proceed to the on-line estimation both of the secondary path and the controller.

The adaptive controller $W(z)$ is implemented with the structure `FXLMS_W` (see the code). Moreover, its filter's weights are initialized as $\mathbf{w} = \delta[0]$. After the creation of the filter's structure as `FXLMS_W = create_STRC_LMS_AF(Mw, 0.001);` the first element of the vector $\mathbf{w}$ is forced as $w[1] = 1$, i.e. `FXLMS_W.w(1) = 1`.

With this initial condition (I.C.), the filter output $W(z)$ is identical to its input. For which if the block (see the parameter `BlkSize` in the MATLAB code) is long enough, in the first iteration the secondary path $S(z)$ is fed with white noise and this facilitates the estimation procedure of the $S(z)$ (see the line code `[LMS_s0] = AF_STRC_LMS_F_B(LMS_s0, BlkSize, x, y(nn));`).

The Fig. 3.20, shows the results of the FX-LMS algorithm in the case in which the secondary path is random modeled as

$$s_0[n] = e^{-0.1n} \cdot \eta[n], \quad n = 0,\ 1,\ \ldots\ M_{s0} - 1$$

where $\eta[n]$ is WGN.

In the figure we can observe that the error signal $e[n]$ tend to zero. Moreover, note that $h[n]$ is the convolution between $s[n]$ and $w[n]$ and, as we can see from the figure, it is very close to a translate unitary impulse such that $H(z) \sim z^{-D}$.

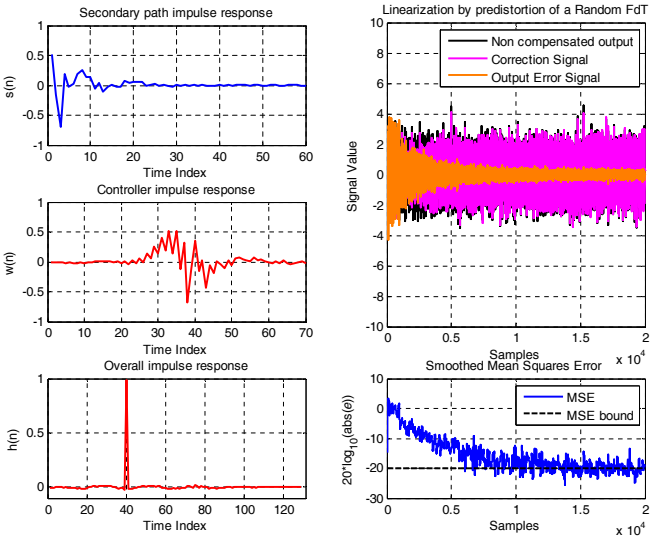

Fig. 3.20 Results of the FX-LMS algorithm. $s_[n]$ is the impulse response of secondary path, $w[n]$ is the impulse response of estimated controller and $h[n] = s[n] * w[n] \sim \delta[n - D]$ (in the simulation $D = 40$).

```
%--------------------------------------------------------------------------
%
% Fx LMS demo program #1 - Linear Predistortion
%
%   Copyright 2016 - A. Uncini
```

```
%   DIET Dpt - University of Rome 'La Sapienza' - Italy
%   $Revision: 1.0$  $Date: 2016/03/25$
%-------------------------------------------------------------------------
clear all; close all;
rng(29);     % Set seed for randon nunber generation
Ms0 = 60;    % Secondary path S(z) model length
Mp0 = 50;    % Primary path  length
Mw  = 70;    % Controller W(z) model length

% *** Set random secondary path  model #1 --------------------------------
for t=1 : Ms0
    s0(t) = exp( -0.1*(t-1) )*randn(1,1);
end
s0 = s0/ norm(s0); % Unitary gain normalization

% *** Create FIR filter struct -------------------------------------------
[FS0] = create_FIR_F(s0); % Create and set secondary path imp. response
[FW]  = create_FIR_F(zeros(Mw,1));       % Controller
FW.h(1) = 1;                             % Controller I.C.
[FS0_hat] = create_FIR_F(zeros(Ms0,1));  % FS0_hat Estimayted Secondary path
[PRM1] = create_FIR_F(zeros(Mp0,1));     % Primary path
PRM1.h(40) = 1;                          % set simple z^(-40) delay

% *** Create AF filter struct --------------------------------------------
[LMS_s0] = create_STRC_LMS_AF(Ms0, 0.001); % LMS_AF for S_hat(z) estimation
[FXLMS_W]  = create_STRC_LMS_AF(Mw, 0.001);  % FXLMS_AF for W(z) estimation
FXLMS_W.w(1) = 1; % Set I.C.
% Set Algortihm parameters
BlkSize = 500 ;
N  = 20000;
s  = randn(N,1); % input signal
y  = zeros(N,1);
d  = zeros(N,1);
e  = zeros(N,1);
NoiseLev_dB   = -20; % Noise level in dB
noise_std_dev =  10^(NoiseLev_dB /20); % Convert [dB] in natural value
for n = 1 : BlkSize : N
    nn = n : n + BlkSize - 1;
    [PRM1,d(nn)] = FIR_STRC_F_B(PRM1, BlkSize, s(nn));  % Primary path
    [FW, x] = FIR_STRC_F_B(FW, BlkSize, s(nn));         % Controller output
    [FS0, y(nn)] = FIR_STRC_F_B(FS0, BlkSize, x); % Secondary path output
    d(nn) = d(nn) + noise_std_dev*randn(BlkSize,1); % Add GWN
    e(nn) = d(nn) - y(nn);
    % --- Estimation of secondary path model S(z)
    [LMS_s0] = AF_STRC_LMS_F_B(LMS_s0, BlkSize, x, y(nn)); % plant modeling
    % --- Copy the estimated S(z) to filter struct FX_S0
    FS0_hat.h = LMS_s0.w;
    [FS0_hat, x_hat] = FIR_STRC_F_B(FS0_hat, BlkSize, s(nn));  % x-filter
    % --- FX-LMS adaptation for controller W(z) estimation
    for k = 1 : BlkSize
        FXLMS_W.xw = [x_hat(k); FXLMS_W.xw(1 : FXLMS_W.M-1)];
        FW.h = FW.h + FXLMS_W.mu*conj(e(n+k-1))*FXLMS_W.xw;
    end
end
% --- Plot Results -----------------------------------------------------
h = conv(s0, FW.h);
%-------------------------------------------------------------------------
figure;
LineWidth = 2;
subplot(3,2,1); hold on;  grid on;  box on;
plot(1 : Ms0, s0, 'b', 'LineWidth', LineWidth ); axis([0 Ms0 -1 1]);
title('Secondary path impulse response');
xlabel('Time Index'); ylabel('s(n)');
subplot(3,2,3); hold on;  grid on;  box on;
plot(1:Mw , FXLMS_W.w,'r','LineWidth', LineWidth );axis([0 Mw -1 1]);
title('Controller impulse response');
xlabel('Time Index'); ylabel('w(n)');
subplot(3,2,5); hold on;  grid on;  box on;
plot(1:length(h), h,'r','LineWidth', LineWidth ); axis([0 Mw+Ms0+1 -0.1 1]);
```

```
title('Overall impulse response');
xlabel('Time Index'); ylabel('h(n)');
%-------------------------------------------------------------------------
subplot(3,2,[2 4])
hold on;  grid on;  box on;
plot(1:N, d,'k', 'LineWidth', LineWidth );
plot(1:N, y,'m', 'LineWidth', LineWidth );
plot(1:N, e,'color',[1 0.5 0.0],'LineWidth', LineWidth );
ylim([-10 10]);
title('Linearization by predistortion of a Random FdT');
legend('Non compensated output', 'Correction Signal' ,'Output Error Signal');
xlabel('Samples'); ylabel('Signal Value');
%-------------------------------------------------------------------------
subplot(3,2,6)
MSE_Bound(1:N) = NoiseLev_dB;
MSE_dB = 10*log10(2*abs(e).^2);
hold on;  grid on; box on;
[b,a]  = butter(4,0.02);  % butterworth smooth filter
Smooth_MSE_dB = filtfilt(b,a, MSE_dB);
plot(Smooth_MSE_dB,  'LineWidth',LineWidth);
plot(MSE_Bound, '--', 'color','k', 'LineWidth',LineWidth);
ylim([NoiseLev_dB-10 10]);
title('Smoothed Mean Squares Error ');
xlabel('Samples');
ylabel('20*log_1_0(abs({\ite}))');
legend('MSE','MSE bound');
%-------------------------------------------------------------------------
```

3.4.3 Active Noise Control by Filtered-X LMS Algorithm

The *active noise cancellation* or *active noise control* (ANC) consists in producing of an acoustic wave, said *antinoise*, in phase opposition with respect to the wave generated by the noise source. This wave has the objective of creating of a silence zone in a given region of space [24].

Referring to the operating principle that is shown in the Fig. 3.21, the first task in ANC is to estimate the impulse response of the *secondary propagation path* denoted as $S(z)$. This step is usually performed prior to noise control using a synthetic random signal, as for example WGN or *pseudo random binary sequences*[1] (PRBS), played through the loudspeaker while the unwanted noise is not present [24]. The transfer function $S(z)$ is the model between the loudspeaker, that produce the anti-noise, and error microphone. Note that the estimate of the secondary path must necessarily be made when the primary noise source is off. Unlike the generic predistorter, in the case of the ANC this stage can not be done online. In addition, always referring to Fig. 3.21, the transfer function $P(z)$ is the model between the noise source and the error microphone and represents the *primary acoustic path*.

It is easy to verify that the schematization the ANC operating principle shown in the Fig. 3.21 is that reported in Fig. 3.22. The ANC scheme is very similar to the

[1] A *pseudo random binary sequence* is a 2-level signal (i.e. binary) that has statistical behavior similar to a truly-random sequence. It is used in identification techniques in that it has a constant amplitude and, therefore, can work in a zone in which the system the to be identified and the device for generating the excitation source (e.g. a loudspeaker), are linear.

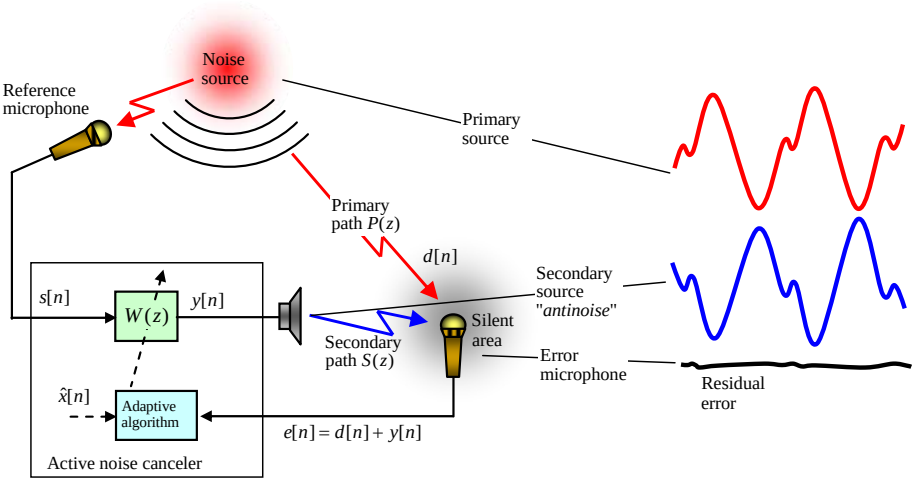

Fig. 3.21 Diagram of the operating principle of an *active noise canceler* (ANC). The loudspeaker makes a wave in phase opposition with respect to the noise at the point where is located the *error microphone*. The reference microphone should be placed as close as possible to the acoustic source of noise.

scheme of the predistorter of Fig. 3.19. The only difference is that, in ANC case, $P(z)$ is an *acoustic transfer function* (ATF) that model the distance and the riverberation between the primary acoustic noise and the error microphone. Thus, in order to create a silent area, we need to determine a controller $W(z)$ that has a transfer function such that at the signal at the error microphone is minimized, in other word such that $W(z)S(z) \sim P(z)$.

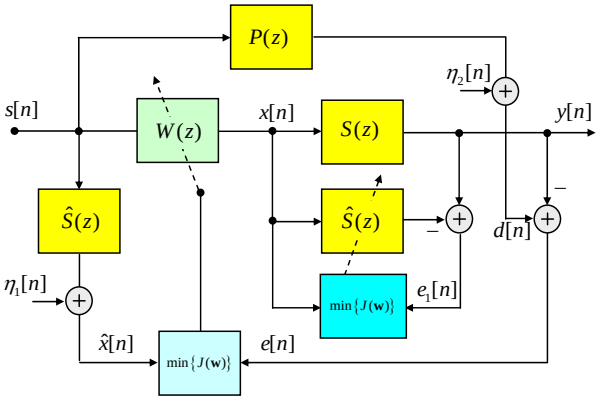

Fig. 3.22 Block diagrams of FxLMS algorithm for for active noise control. The transfer function $P(z)$ is the primary acoustic propagation path. In the ANC simulation, the noise $\eta_1(n)$ taking account of any error of the secondary path estimates, while $\eta_2(n)$ is the not cancellable noise.

Example 3.10. In ANC simulation systems, in order the generate the primary acoustic propagation, we may consider the frequency range 50-3000 [Hz], and that for the

filter length we may consider an impulse response of duration in the range 0.1 - 0.3 [s]. For example, considering a duration of 0.1 [s], for a sampling frequency of 8 [kHz], the impulse response has length equal to 800 samples. In addition, note in the secondary path we must also consider the loudspeaker's frequency response that, in our tests, was chosen in the range 30 - 1000 [Hz]. In the simulation, the estimation of the secondary path has been made prior to the noise control phase, using a WGN signla with a duration of 2 sec.

Below is reported the MATLAB code used in the simulations.

```
%--------------------------------------------------------------------------
%
%   Active Noise Control by Filtered-X LMS algorithm
%
% -------------------------------------------------------------------------
% A. Uncini,"Fundamentals of Adaptive Signal Processing",
% Springer Ed., ISBN 978-3-319-02806-4, 2015.
%
%   Copyright 2016 - A. Uncini
%   DIET Dpt - University of Rome 'La Sapienza' - Italy
%   $Revision: 1.0$  $Date: 2016/03/26$
%--------------------------------------------------------------------------
clear all; close all;
rng(19);     % Set seed for random number generation
Fs = 8000;   % Sampling frequency
t1 = 0.1;    % Secondary/Primary path  duration in seconds
t2 = 0.2;    % Controller path duration in seconds
Ts0 = t1*Fs; % Secondary path length
Tp0 = t1*Fs; % Primaary path length
Tw0 = t2*Fs; % Controller filter length
Tmax = 15;   % Seconds
N = 2^( 1 + fix(log(Tmax*Fs)/log(2)));
BlkSize = 512;
% -------------------------------------------------------------------------
% Secondary path impulse response generation
% -------------------------------------------------------------------------
% Secondary path impulse response generation. Exponentianal decay of a
% pass-band white noise filtered signal by the frequency response of the
% loudspeaker that is in the range 30  1000 Hz
%
s0 = filter ( fir1(71,[30 1000]/(Fs/2), 'bandpass'), 1, ...
    randn(1,Ts0).*exp(-0.01* (1:Ts0)) );
s0 = s0/ norm(s0); % normalized 0dB gain
%
% -------------------------------------------------------------------------
% Primary path impulse response generation
% -------------------------------------------------------------------------
% Primary path impulse response generation. Exponentianal decay of a
% pass-band from 160 to 800 Hz white noisefiltered signal
% Prm model #1
p0 = filter ( fir1(91,[50 3000]/(Fs/2), 'bandpass'), 1, ...
    randn(1,Ts0).*exp(-0.01* (1:Ts0)) );
p0 = p0/ norm(p0); % normalized 0dB gain

% Prm model #2
p1 = filter ( fir1(91,[50 3000]/(Fs/2), 'bandpass'), 1, ...
    randn(1,Ts0).*exp(-0.02* (1:Ts0)) );
p1 = p1/ norm(p1); % normalized 0dB gain

%--------------------------------------------------------------------------
% Secondary path estimete using 2 seconds signal
%--------------------------------------------------------------------------
[LMS_s0] = create_STRC_LMS_AF(Ts0, 0.001);  % pre-estimate model s0
LMS_s0.w(1) = 1;
Ns = 2*Fs;
ss0 = randn(Ns,1);
dd0 = filter(s0, 1, ss0); % add some noise measure
```

```
[LMS_s0, y, err] = AF_STRC_LMS_F_B( LMS_s0, Ns, ss0, dd0 );

%-------------------------------------------------------------------------
% Structs creation for the ANC by FxLMS Algortihm
%-------------------------------------------------------------------------
muW = 5e-6;
[FXLMS] = create_STRC_FXLMS_AF_1(Tw0, s0, muW,'d'); % FX-LMS struc
[FP0]   = create_FIR_F(p0);       % Primary path FIR struct
[FX_S0] = create_FIR_F(LMS_s0.w); % Estimated s0 path FIR struct or I.C.

%-------------------------------------------------------------------------
% Genaration signal of the  noise source
%-------------------------------------------------------------------------
s = zeros(N,1);
F0 = 50;      % [Hz]
n = 1:N;
s = zeros(N,1);
W0 = 2*pi*F0/Fs;
% Sum of 10 sinusoids
for k=1:10;
    W0 = 2*pi*(  k*F0 + 80*(0.7 + 0.3*(rand(1,1)-0.5)) )  /Fs;
    s = s + ( 0.1 + 0.1*rand(1,1) )*sin(W0*n)';
end
% Set signal dimension
y  = zeros(N,1);
d  = zeros(N,1);
e  = zeros(N,1);
NoiseLev_dB   = -20; % Noise level in dB
noise_std_dev =  10^(NoiseLev_dB /20); % Convert [dB] in natural value
%-------------------------------------------------------------------------
% ANC by Fx LMS Algorithm
%-------------------------------------------------------------------------
fprintf('Start DSP %d samples,   BlkSize = %d   T: %4.1f [s] \n', ...
    N, BlkSize, N/Fs );
t = clock;
for n = 1 : BlkSize : N
    nn = n : n + BlkSize - 1;
    [FP0,d(nn)] = FIR_STRC_F_B(FP0, BlkSize, s(nn));
    d(nn) = d(nn) + noise_std_dev*randn(BlkSize,1); % add some not cancelable
     noise
    % --- x_hat generat by filtering s with the estimated secondary path
    [FX_S0, x_hat] = FIR_STRC_F_B(FX_S0, BlkSize, s(nn));
     % --- Change primary path  model
     %     if n==(100*BlkSize+1)
     %         fprintf('Change PR_PATH and SC_PATH models \n');
     %         FP0.h=p1;
     %     end
    [FXLMS, e(nn), y(nn)] = AF_STRC_FXLMS_F_B_1(FXLMS, BlkSize, ...
        x_hat, s(nn), d(nn));
end
fprintf('ANC_filter  Elapsed time: %f [s]\n',etime(clock, t));
%-------------------------------------------------------------------------
%-------------------------------------------------------------------------
% Play results
%-------------------------------------------------------------------------
% p8K = audioplayer(d/max(abs(d)), Fs);
% playblocking(p8K);
%  p8K = audioplayer(e/max(abs(e)), Fs);
%  playblocking(p8K);
%-------------------------------------------------------------------------
% S(z) vs Sh(z) and P(z) imp. resp. plot
%-------------------------------------------------------------------------
figure;
subplot(2,2,1)
LineWidth = 2;
hold on; grid on;  box on;
tt = t1*[ 0: Ts0-1 ]/(Ts0-1);
plot(tt, s0,'r','LineWidth', LineWidth);
plot(tt, LMS_s0.w,'-.','color','k','LineWidth', LineWidth);
```

```
title('Secondary path impulse response');
legend('Original','Estimated');
xlabel('Time [s]'); ylabel('s0(n)');
% ------------------------------------------------------------------------
subplot(2,2,3)
hold on; grid on;  box on;
tt = t2*[ 0: Tp0-1 ]/(Tp0-1);
xlim([0 t2]);
plot(tt,p0 ,'b','LineWidth', LineWidth);
title('Primary path impulse response');
xlabel('Time [s]'); ylabel('p0(n)');
%-------------------------------------------------------------------------
% Plot Noise, antinoise and error signals
%-------------------------------------------------------------------------
subplot(2,2,2);
LineWidth = 2;
hold on;grid on; box on;
tt = Tmax*[ 0: N-1 ]/(N-1);
plot(tt, d,'b', 'LineWidth', LineWidth );
plot(tt, y,'m','LineWidth', LineWidth );
plot(tt, e,'color',[1 0.5 0.0],'LineWidth', LineWidth );
ylim([-3 3]);
title('Active Noise Control of a Random Noise Signal');
legend('Noise','Anti-Noise','Mic. Error Signal');
xlabel('Time [s]'); ylabel('Signal Values');
%-------------------------------------------------------------------------
% Plot PSD of Noise and microhone error signals
%-------------------------------------------------------------------------
Ns = 10000; nfft = 8192;
Pdd = abs(fft(d(N-Ns:N),nfft)).^2/1/Fs;
Hpsd_d = dspdata.psd(Pdd(1:length(Pdd)/2),'Fs',Fs);
subplot(2,2,4);
hold on; grid on; box on;
Pee = abs(fft(e(N-Ns:N),nfft)).^2/1/Fs;
Hpsd_e = dspdata.psd(Pee(1:length(Pee)/2),'Fs',Fs);
hold on;  grid on;  box on;
plot(Hpsd_d.Frequencies, 10*log10(Hpsd_d.Data),'LineWidth', LineWidth  );
plot(Hpsd_e.Frequencies, 10*log10(Hpsd_e.Data),'color',[1 0.5 0.0],...
    'LineWidth', LineWidth  );
xlim([0 Fs/2]);
ylim([NoiseLev_dB-5 25]);
title('Noise Power Spectral Density');
xlabel('Frequency [Hz]'); ylabel('Psd [dB]');
legend('Noise PSD','Err. Mic. PSD');
%-------------------------------------------------------------------------


function  [FXLMS, e, y, ye ] = AF_STRC_FXLMS_F_B_1(FXLMS, BlkSz, x_hat, s , d )
% *************************************************************************
% Standard FX-LMS algorithm
% -------------------------------------------------------------------------
%  Call as:
%       [FXLMS, e ]  = AF_STRC_FXLMS_F_B_1(FXLMS, BlkSz, x_hat, s , d)
%
% Input Arguments:
%   FXLMS    : FX-LMS ADAPTIVE FILTER STRUCT
%   BlkSz    : Length of input signal
%   x_hat    : filtered_x[n] input signal
%   s        : primary microphone reference signal
%   d        : d[n] desired signal sample. Noise to be cancelled
%
% Output Arguments:
%   FXLMS    : ADAPTIVE FILTER STRUCT
%   e        :  e[n] error signal sample
%   y        :  y[n] secondary path output. Antinoise signal
%   ye        :ye[n] controller path output
%
% -------------------------------------------------------------------------
% A. Uncini,"Fundamentals of Adaptive Signal Processing",
```

```
% Springer Editor, ISBN 978-3-319-02806-4, 2014.
%
%   Copyright 2013 - Aurelio Uncini
%   DIET Dept. - 'Sapienza' University of Rome
%   $Revision: 1.0$  $Date: 2013/11/25$
% ------------------------------------------------------------------------
    % --- Controller path output
    [FXLMS.Fz, ye] = FIR_STRC_F_B(FXLMS.Fz, BlkSz, s(:));
    % Secondary path output
    [FXLMS.Sz, y]  = FIR_STRC_F_B(FXLMS.Sz, BlkSz, ye);
    % --- On-line Controller adaptation
    for n = 1 : BlkSz
        FXLMS.Wz.xw = [x_hat(n) ; FXLMS.Wz.xw(1 : FXLMS.Wz.M-1)];
        e(n) = d(n) - y(n);
        FXLMS.Fz.h = FXLMS.Fz.h + FXLMS.Wz.mu*conj(e(n))*FXLMS.Wz.xw;
    end
return
% ------------------------------------------------------------------------

function  [FXLMS] =  create_STRC_FXLMS_AF_1(M, hs, mu, IC)
% ************************************************************************
%
%  STRUCT creation of Standard FX-LMS algorithm
%
%               +-----------+                      +
% s(n) ---+--->|extern def.|---d(n)----------------> sum --+---> e(n)
%         |     +-----------+                        ^-    |
%         |                                          |     |
%         |        \                               y(n)   |
%         |    +-----------+           +-----------+  |    |
%         +--->|   Wz,Fz   |---ye(n)-->|    Sz     |--+    |
%         |    +-----------+           +-----------+       |
%         |            \                                   |
%         |             \----------------\                 |
%         |                               \                |
%         |    +-----------+ x_hat(n)  +-----------+       |
%         +--->|extern def.|---------->|   FxLMS   |<------+
%              +-----------+           +-----------+   e(n)
%
% ------------------------------------------------------------------------
% Input parameters
%      M      : controller AF length
%      hs     : secondary path filter impulse response
%      mu     : LMS learning rate
%      IC     : controller W(z) initial condition
%
% Output parameters
%      FXLMS  : Fx LMS  Struct
% ------------------------------------------------------------------------
%  Sub structs output parameters
%
%    FXLMS.Sz
%        FIR: 'STRUCT standard time-domain on-line and batch FIR filtering'
%          M: 800
%          h: [800x1 double]
%         xh: [800x1 double]
%
%    FXLMS.Wz
%         AF: 'Direct-Form FIR Adaptive Filter struct for LMS algorithm'
%          M: 1200
%         mu: 1.0000e-05
%          w: [1200x1 double]
%         xw: [1200x1 double]
%
%    FXLMS.Fz
%        FIR: 'STRUCT standard time-domain on-line and batch FIR filtering'
%          M: 1200
%          h: [1200x1 double]
%         xh: [1200x1 double]
```

```
%
%-------------------------------------------------------------------------
% A. Uncini ,"Fundamentals of Adaptive Signal Processing",
% Springer Ed.,  ISBN 978-3-319-02806-4, 2015.
%
%   Copyright 2016 - A. Uncini,
%   DIET - University of Rome 'La Sapienza' Italy
%   $Revision: 1.0$  $Date: 2016/26/03$
%-------------------------------------------------------------------------
    if nargin==0, help create_STRC_FXLMS_AF_1; return; end
    if nargin<4, IC='z'; end

    [Sz] = create_FIR_F(hs);   % Secondary path output
    [Wz] = create_STRC_LMS_AF(M, mu);  % Controller adaptation
    [Fz] = create_FIR_F(zeros(M,1));   % Controller output
    % --- set I.C of the controller W(z)
     if IC =='r'
        Wz.w = 0.3*(rand(M,1)-0.5);
    end
    if IC =='d'
        Wz.w(1) = 1;
    end
    FX = 'STRUCT Filterd-x time-domain LMS algorithm';
    FXLMS = struct('FX',FX, 'Wz',Wz, 'Sz',Sz,'Fz',Fz);
return
%-------------------------------------------------------------------------
```

In the tests, to simulate the noise source, it has been considered as a sum of sinusoids with random frequency, generated in the range between 50 and 720 [Hz].

The results of the simulation is reported in Fig. 3.23.

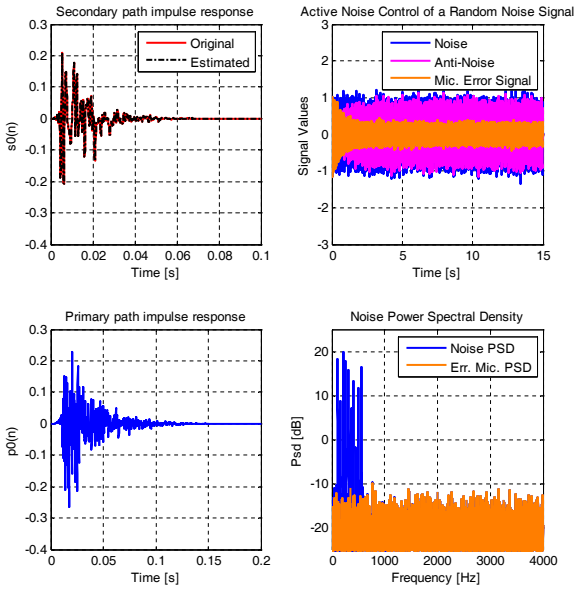

Fig. 3.23 Active noise control by FxLMS algorithm. Simulation results.

3.4.4 ANC by Adjoint LMS Algorithm

The adjoint LMS (AD-LMS) algorithm, represents an alternative way of FX-LMS algorithm for the controller adaptation in predistortion and ANC systems. Referring to Fig. 3.24 the AD-LMS can be derived by considering three steps.

1. As shown in Fig. 3.24-b) we must switch the adaptation block LMS and the block of the secondary path estimation $S(z)$. In order for this operation to be consistent, as shown in Fig.s 3.25-a) and b), the transfer function path $\hat{S}(z)$ become non causal as $\hat{S}(z) \rightarrow \hat{S}(z^{-1})$, i.e. in the transfer function definition the delays are transformed in anticipation elements $z^{-1} \rightarrow z^{+1}$. This means that the signal $\hat{e}[n]$ is dependent on the future samples of the error output $e[n+k]$.
2. In order to causalize the transfer function $\hat{S}(z^{-1})$, it is sufficient to appropriately delay, the output error signal. This means that $\hat{S}(z^{-1}) \rightarrow z^{-D}\hat{S}(z^{-1})$, where D is a suitable delay
3. Finally, for the correct LMS adaptation the input and the error signals must be perfectly aligned. Thus, as shown in Fig. 3.24-c), is also necessary to delay, of a factor D, the input signal of the LMS block.

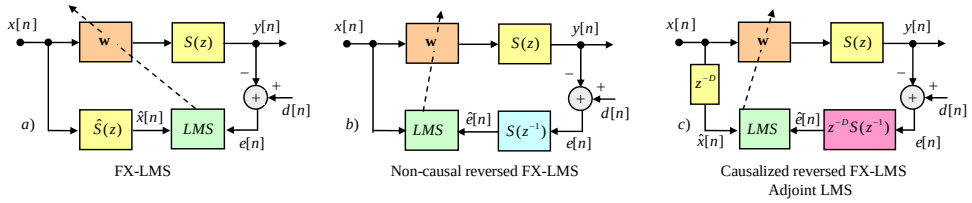

Fig. 3.24 The adjoint LMS AD-LMS algorithm derivation.

Remark 3.5. By a simple visual inspection of Fig. 3.25-c), we can observe that the SFG of the transfer function in the dotted box corresponds exactly to the so-called *adjoint graph* or *adjoint network*[2] of the SFG of $S(z)$ (not reported for brevity). The name of the algorithm adjoint LMS, originally developed in [25], is derived from this property.

In AD-LMS, the LMS update is performed considering a filtered version of the delayed error signal. Let $\hat{\mathbf{s}} = [\hat{s}_0 \;\; \hat{s}_1 \cdots \hat{s}_{M_s-1}]^T$, be the estimated impulse response of the secondary path, in order to have a correct LMS update, the transfer function $\hat{S}(z^{-1})$, in order to compensate the anticipation elements, is fed by a delayed error signal $e[n-D]$. In other word, we have that

[2] Given a discrete-time circuit defined by a graph $\mathcal{G}$, we define the adjoint network a circuit whose graph is determined by $\mathcal{G}$ with the following modifications: 1) the paths verses are reversed; 2) junction nodes are switched with sum nodes; 3) delay elements are replaced with anticipation elements.

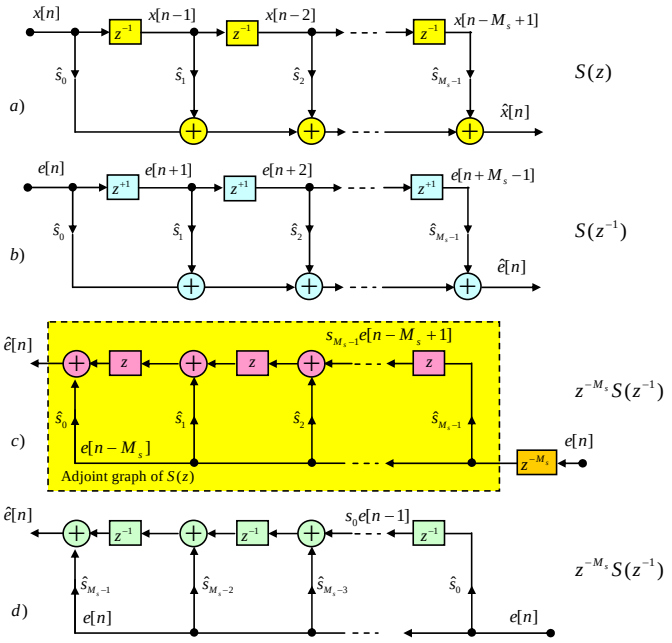

Fig. 3.25 The signal flow graph (SFG) of the error signal filtered by secondary path transfer function. a) SFG of S(z); b) SFG of non causal $S(z^{-1})$. c) Causalized SFG $z^{-M_s}S(z^{-1})$. d) Equivalent causalized SFG, implemented with only the delay elements, but with the filter coefficients arranged in reversed order.

$$\frac{\hat{E}(z)}{E(z)} = z^{-D}\hat{S}(z^{-1}).$$

More formally, for $D = M_s - 1$, the filter error transfer function can be written as

$$\begin{aligned} z^{-M_s+1}\hat{S}(z^{-1}) &= z^{-M_s+1}\left(\hat{s}_0 + \hat{s}_1 z^1 + \cdots + \hat{s}_{M_s-1} z^{M_s-1}\right) \\ &= \hat{s}_0 z^{-M_s+1} + \hat{s}_1 z^{M_s} + \cdots + \hat{s}_{M_s-1} z^0 \\ &= \hat{s}_{M_s-1} + \hat{s}_{M_s-2} z^{-1} + \cdots + \hat{s}_0 z^{-M_s+1}. \end{aligned}$$

Note that the last expression corresponds to the equivalent causalized SFG version of $z^{-M_s+1}S(z^{-1})$, implemented with only the delay elements, and with the coefficients arranged in reverse order (see Fig. 3.25-d)). Thus, in the time domain we have that the filtered error $\hat{e}[n]$ should be evaluated as

$$\hat{e}[n] = \sum_{k=0}^{M_s-1} \hat{s}[M_s - k]e[n-k].$$

Moreover, in order to keep the filtered version of the error aligned with the signal that is used for the LMS update, the input signal must by delayed by a delay element

z^{-M_s+1}. The resulting algorithm is illustrated in Fig. 3.26. The corresponding AD-LMS adaptation rule can be written as

$$\mathbf{w}_n = \mathbf{w}_{n-1} + \mu \mathbf{x}_{n-M_s} \hat{e}[n]. \tag{3.20}$$

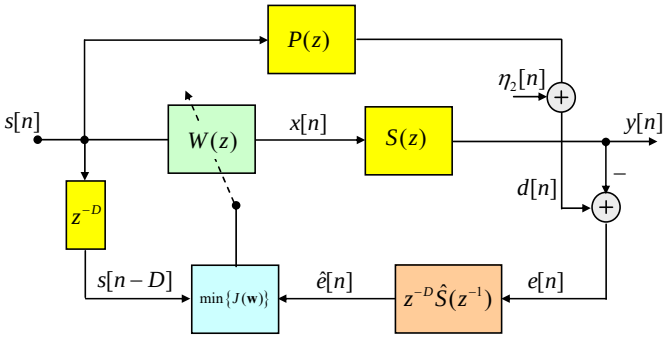

Fig. 3.26 Block diagrams of the adjoint LMS algorithm for active noise control.

Example 3.11. Below is reported the MATLAB code of the AD-LMS algorithm for the same problem previously presented in Example 3.10. The simulation results are reported in Fig. 3.27.

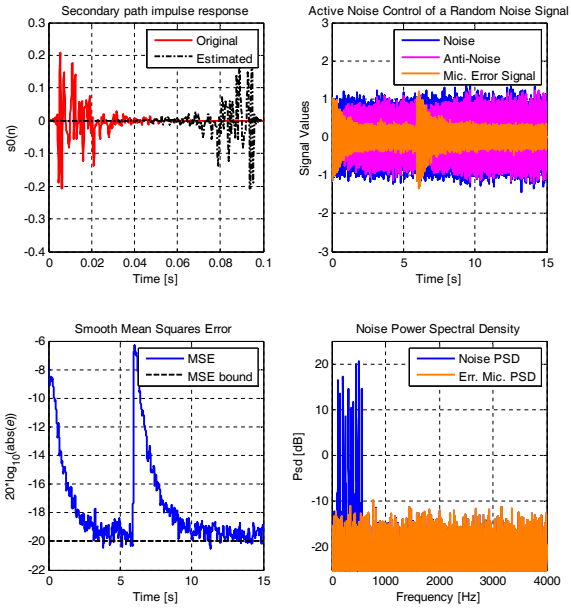

Fig. 3.27 Active noise control by AD-LMS algorithm. Simulation results.

```
%--------------------------------------------------------------------------
%
%   Active Noise Control by Adjoint-LMS algorithm
%
% -------------------------------------------------------------------------
% A. Uncini,"Fundamentals of Adaptive Signal Processing",
% Springer Ed., ISBN 978-3-319-02806-4, 2015.
%
%   Copyright 2016 - A. Uncini
%   DIET Dpt - University of Rome 'La Sapienza' - Italy
%   $Revision: 1.0$  $Date: 2016/03/31$
%--------------------------------------------------------------------------
clear all; close all;
rng(19);     % Set seed for random number generation
Fs = 8000;   % Sampling frequency
t1 = 0.1;    % Secondary/Primary path  duration in seconds
t2 = 0.2;    % Controller path duration in seconds
Ts0 = t1*Fs; % Secondary path length
Tp0 = t1*Fs; % Primaary path length
Tw0 = t2*Fs; % Controller filterlength
Tmax = 15;   % Seconds
N = 2^( 1 + fix( log2(Tmax*Fs)) ); % pow of two
BlkSize = 512;
% -------------------------------------------------------------------------
% Secondary path impulse response generation
% -------------------------------------------------------------------------
% Secondary path impulse response generation. Exponentianal decay of a
% pass-band white noise filtered signal by the frequency response of the
% loudspeaker that is in the range 30 1000 Hz
%
s0 = filter ( fir1(71,[30 1000]/(Fs/2), 'bandpass'), 1, ...
    randn(1,Ts0).*exp(-0.01* (1:Ts0)) );
s0 = s0/ norm(s0); % normalized 0dB gain
%
% -------------------------------------------------------------------------
% Primary path impulse response generation
% -------------------------------------------------------------------------
% Primary path impulse response generation. Exponentianal decay of a
% pass-band from 160 to 800 Hz white noisefiltered signal
% Prm model #1
p0 = filter ( fir1(91,[50 3000]/(Fs/2), 'bandpass'), 1, ...
    randn(1,Ts0).*exp(-0.01* (1:Ts0)) );
p0 = p0/ norm(p0); % normalized 0dB gain

% Prm model #2
p1 = filter ( fir1(91,[50 3000]/(Fs/2), 'bandpass'), 1, ...
    randn(1,Ts0).*exp(-0.02* (1:Ts0)) );
p1 = p1/ norm(p1); % normalized 0dB gain

%--------------------------------------------------------------------------
% Secondary path estimete using 2 seconds of WGN signal
%--------------------------------------------------------------------------
[LMS_s0] = create_STRC_LMS_AF(Ts0, 0.001);  % pre-estimate model s0
LMS_s0.w(1) = 1;
Ns = 2*Fs;
ss0 = randn(Ns,1);
dd0 = filter(s0, 1, ss0); % add some noise measure
[LMS_s0, y, err] = AF_STRC_LMS_F_B( LMS_s0, Ns, ss0, dd0 );
LMS_s0.w = flipud(LMS_s0.w); % reverse S0 impulse response for FE-LMS

%--------------------------------------------------------------------------
% Structs creation for the ANC by FE-LMS Algortihm
%--------------------------------------------------------------------------
muW = 5e-6;
[ADLMS] = create_STRC_LMS_AF(Tw0, muW); % AD-LMS struc
[FW]    = create_FIR_F(zeros(Tw0,1));   % Controller
[FS0]   = create_FIR_F(s0);             % Secondary path filter
[FP0]   = create_FIR_F(p0);             % Primary path FIR struct
```

```
[FX_S0] = create_FIR_F(LMS_s0.w);       % Filter for estimated s0 path
[DL]    = create_DSP_STRC_DL(Ts0);      % Delay line
%-------------------------------------------------------------------------
% Noise source genaration signal
%-------------------------------------------------------------------------
s = zeros(N,1);
F0 = 50;      % [Hz]
n = 1:N;
s = zeros(N,1);
W0 = 2*pi*F0/Fs;
% Sum of 10 sinusoids
for k=1:10;
    W0 = 2*pi*(  k*F0 + 80*(0.7 + 0.3*(rand(1,1)-0.5)) ) / Fs;
    s = s + ( 0.1 + 0.1*rand(1,1))*sin(W0*n)';
end
% Set signals dimension
y  = zeros(N,1);
d  = zeros(N,1);
e  = zeros(N,1);
NoiseLev_dB   = -20; % Noise level in dB
noise_std_dev =  10^(NoiseLev_dB /20); % Convert [dB] in natural value
%-------------------------------------------------------------------------
% ANC by Fx LMS Algorithm
%-------------------------------------------------------------------------
fprintf('Start DSP %d samples,   BlkSize = %d   T: %4.1f [s] \n', ...
    N, BlkSize, N/Fs );
t = clock;
for n = 1 : BlkSize : N
    nn = n : n + BlkSize - 1; % running window indexes
    [FP0, d(nn)] = FIR_STRC_F_B(FP0, BlkSize, s(nn)); % Primary path
    [FW, x] = FIR_STRC_F_B(FW, BlkSize, s(nn));       % Controller output
    [FS0, y(nn)] = FIR_STRC_F_B(FS0, BlkSize, x);     % Secondary path out
    d(nn) = d(nn) + noise_std_dev*randn(BlkSize,1);   % Add GWN
    e(nn) = d(nn) - y(nn);                            % output error
    % --- Change primary path model ---
    if n==(100*BlkSize+1)
        fprintf('Change PR_PATH and SC_PATH models \n');
        FP0.h=p1;
    end
    [FX_S0, e_hat] = FIR_STRC_F_B(FX_S0, BlkSize, e(nn)); % Filt. error
    [DL,x_del] =  DSP_STRC_DL_F_B(DL, BlkSize, s(nn));    % Delayed input
    % --- AD-LMS adaptation for controller W(z) estimation ---
    for k = 1 : BlkSize
        ADLMS.xw = [x_del(k); ADLMS.xw(1 : ADLMS.M-1)];
        FW.h = FW.h + ADLMS.mu*conj(e_hat(k))*ADLMS.xw;
    end
end
fprintf('ANC_filter  Elapsed time: %f [s]\n',etime(clock, t));
%-------------------------------------------------------------------------
%-------------------------------------------------------------------------
% Play results
%-------------------------------------------------------------------------
% p8K = audioplayer(d/max(abs(d)), Fs);
% playblocking(p8K);
% p8K = audioplayer(e/max(abs(e)), Fs);
% playblocking(p8K);
%-------------------------------------------------------------------------
% S(z) vs Sh(z) and P(z) imp. resp. plot
%-------------------------------------------------------------------------
figure;
subplot(2,2,1)
LineWidth = 2;
hold on; grid on;  box on;
tt = t1*[ 0: Ts0-1 ]/(Ts0-1);
plot(tt, s0,'r','LineWidth', LineWidth);
plot(tt, LMS_s0.w,'-.','color','k','LineWidth', LineWidth);
title('Secondary path impulse response');
legend('Original','Estimated');
xlabel('Time [s]'); ylabel('s0(n)');
```

```
% ------------------------------------------------------------------------
% MSE dB
% ------------------------------------------------------------------------
subplot(2,2,3)
MSE_Bound(1:N) = NoiseLev_dB ;
MSE_dB = 20*log10(2*abs(e));
hold on; grid on; box on;
[b,a]  = butter(2,0.002);  % butterworth smooth filter
Smooth_MSE_dB = filtfilt(b,a, MSE_dB);
tt = Tmax*[ 0: N-1 ]/(N-1);
plot(tt, Smooth_MSE_dB,  'LineWidth',LineWidth);
plot(tt, MSE_Bound, '--', 'color','k', 'LineWidth',LineWidth);
title('Smooth Mean Squares Error ');
xlabel('Time [s]');
ylabel('20*log_1_0(abs({\ite}))');
legend('MSE','MSE bound');
%------------------------------------------------------------------------
% Plot Noise, antinoise and error signals
%------------------------------------------------------------------------
subplot(2,2,2);
hold on;grid on; box on;
tt = Tmax*[ 0: N-1 ]/(N-1);
plot(tt, d,'b', 'LineWidth', LineWidth );
plot(tt, y,'m','LineWidth', LineWidth );
plot(tt, e,'color',[1 0.5 0.0],'LineWidth', LineWidth );
ylim([-3 3]);
title('Active Noise Control of a Random Noise Signal');
legend('Noise','Anti-Noise','Mic. Error Signal');
xlabel('Time [s]'); ylabel('Signal Values');
%------------------------------------------------------------------------
% Plot PSD of Noise and microhone error signals
%------------------------------------------------------------------------
Ns = 10000; nfft = 8192;
Pdd = abs(fft(d(N-Ns:N),nfft)).^2/1/Fs;
Hpsd_d = dspdata.psd(Pdd(1:length(Pdd)/2),'Fs',Fs);
subplot(2,2,4);
hold on; grid on; box on;
Pee = abs(fft(e(N-Ns:N),nfft)).^2/1/Fs;
Hpsd_e = dspdata.psd(Pee(1:length(Pee)/2),'Fs',Fs);
hold on;  grid on;  box on;
plot(Hpsd_d.Frequencies, 10*log10(Hpsd_d.Data),'LineWidth', LineWidth  );
plot(Hpsd_e.Frequencies, 10*log10(Hpsd_e.Data),'color',[1 0.5 0.0],...
    'LineWidth', LineWidth  );
xlim([0 Fs/2]);
ylim([NoiseLev_dB-5 25]);
title('Noise Power Spectral Density');
xlabel('Frequency [Hz]'); ylabel('Psd [dB]');
legend('Noise PSD','Err. Mic. PSD');
%------------------------------------------------------------------------
```

3.5 Proposed Problems

Exercise 3.1. Write a MATLAB program to generate a sine wave of a duration equal to 1 [sec], of frequency equal to a 1kHz and amplitude $A = \sqrt{2}$, whereas a sampling frequency of 32 kHz. Add to this sinusoid a withe Gaussian noise, such that the signal to noise ratio (SNR) is equal to 10 dB.

Exercise 3.2. Write a MATLAB program to generate AR(1) stochastic process $x[n]$ with unitary variance, and such that the condition number of the correlation matrix $\chi(\mathbf{R}_{xx}) = 1000$.

Exercise 3.3. Given a zero-mean Gaussian white sequence $\eta[n] = \mathcal{N}(0, \sigma_\eta^2)$, find the autocorrelation sequence of the following stochastic processes:

i) $x[n] = a_0\eta[n] + a_1\eta[n-1] + a_2\eta[n-2]$;
ii) $x[n] = \eta[n] + 0.2\eta[n-1] + 0.1\eta[n-2]$;

Exercise 3.4. An AR(1) stochastic process is defined as $x[n] = ax[n-1] + (1 - a^2)\eta[n-1]$, where $\eta[n] = \mathcal{N}(0,1)$. Determine the value of a such that the condition number of the correlation matrix of $x[n]$, is equal to 1000;

Exercise 3.5. Given an adaptive linear filter with $M = 2$ (two tap filter) such that the following statistics hold

$$\mathbf{R}_{xx} = \begin{bmatrix} 1 & 0.707 \\ 0.707 & 1 \end{bmatrix}; \quad \mathbf{g} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}; \quad \sigma_d^2 = 1.$$

i) Determines and plot the performance surface of the filter;
ii) determines the optimum value $\mathbf{w}_{opt}$ of the Wiener filter;
iii) determines the value $J(\mathbf{w}_{opt})$.

Exercise 3.6. Find the autocorralation matrix associated with tha AR(2) stochastic process defined as

$$x[n] = x[n-1] - 0.2x[n-2] + \eta[n]$$

where $\eta[n] = \mathcal{N}(0,1)$.

Exercise 3.7. Draw the signal flow graph of the discrete-time prediction error filter defined by the Eqn. 3.18.

Exercise 3.8. Modify the MATLAB forward linear prediction program proposed in Section § using the function `function [F, y, e] = AF_STRC_LMS_F_B(F, BlkSz, x, d)`.

Exercise 3.9. Modify the MA system identification problem, proposed in Example 3.4, using the LMS and LS `function [F, y, e] = AF_STRC_LMS_F_B(F, BlkSz, x, d)` function introduced in §2.1.6.

Exercise 3.10. Modify the program of Exercise 3.9 in order to perform several algorithm runs, and using the procedure in §2.1.5, in order to compute the mean and the variance of the estimated MA system.

Exercise 3.11. Modify the code of Example 3.9 in order to have a slow variant non-stationary secondary path $S(z)$.

Exercise 3.12. Modify the code of Example 3.10 in order to have a slow variant non-stationary primary $P(z)$ and secondary $S(z)$, paths.

Exercise 3.13. Write a MATLAB code of that implements the filtered-error LMS (FE-LMS) algorithm illustrated in Fig. 3.28 (see [26]-[28]).

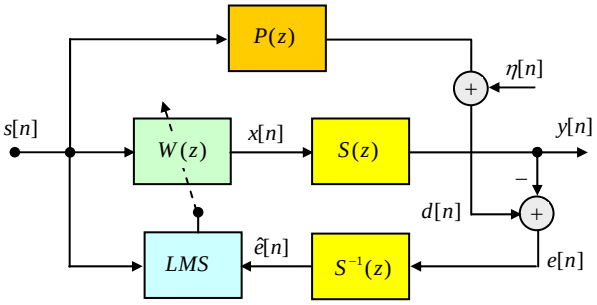

Fig. 3.28 Active noise control by filtered-error LMS (FE-LMS) algorithm.

Exercise 3.14. Write a MATLAB code in order to evaluate the performbce of FX-LMS vs FE-LMS algorithm.

Exercise 3.15. Modify the code of Example in order to evaluate the performance of the algorithm in term of averaged learning curve.

References

1. A. Papoulis, “Probability, random variables and stochastic processes, ” McGraw-Hill in Electrical Engineering, ISBN 0-07-048477-5, 1991.
2. A. Uncini, “Fundamentals of Adaptive Signal Processing, ” Springer, ISBN: 978-3-319-02806-4, 2015.
3. S.J. Orfanidis, “Introduction to Signal Processing,” Prentice Hall, 1996.
4. T.F. Chan, G.H. Golub and R.J. Le Veque, “Algorithms for computing the sample variance: Analysis and recommendations”, The American Statistician, Vol 37, pp. 242–247, 1983.
5. B. Widrow, M. A. Lehr, "‘30 Years of Adaptive Neural Networks: Perceptron, Madaline and Backpropagation,"’ Proceedings of IEEE, Vol. 78, No. 9, Sett. 1990.
6. K.B. Petersen and M.S. Pedersen, “The Matrix Cookbook,” http://orion.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf, Ver. November, 2013.
7. S. M. Kay, “Fundamentals of Statistical Signal Processing Detection Theory”, Prentice Hall, Upper Saddle River, NJ, 1998.
8. R.A. Fisher, “On the mathematical foundations of theoretical statistics”, Philosophical Transactions of the Royal Society, A, 222: 309-368, 1922.
9. D. E. Rumelhart, J. L. McClelland and the PDP Research Group, “Parallel Distributed Processing (PDP): Exploration in the Microstructure of Cognition,” Vol.1, MIT Press, Cambridge, Massachusetts, 1986.
10. W.S. McCulloch and W. Pitts, “A Logical Calculus of Ideas Immanent in Nervous Activity,” Bull. Mathematical Biophysics, Vol, 5, pp. 115–133, 1943.
11. Jacques Hadamard, “Sur les problèmes aux dérivées partielles et leur signification physique, ” Princeton University Bulletin. pp. 49–52, 1902.
12. A. N. Tychonoff, V. Y. Arsenin, “Solution of Ill-posed Problems,” Washington: Winston Sons. ISBN 0-470-99124-0, 1977
13. Ryan Tibshirani (with Larry Wasserman), “Sparsity and the Lasso, ” Statistical Machine Learning, Spring 2015.
14. K. Levenberg, “A Method for the Solution of Certain Problems in Least Squares”, Quart. Appl. Math. 2, pp 164–168, 1944.
15. D. Marquardt, “An Algorithm for Least Squares Estimation on Nonlinear Parameters”, SIAM J. APPL. MATH.11, pp 431–441, 1963.
16. T.M. Cover, “Geometrical and Statistical properties of systems of linear inequalities with applications in pattern recognition”. IEEE Transactions on Electronic Computers. EC-14: 326–334. doi:10.1109/pgec.1965.264137, 1965.
17. Max Kamenetsky and Bernard Widrow, “A Variable Leaky LMS Adaptive Algorithm,” IEEE Thirty-Eighth Asilomar Conf. on Signals, Systems and Computers, pp 125-128, Vol.1, 2004.
18. K. Mayyas and Tyseer Aboulnasr. “Leaky LMS Algorithm: MSE Analysis for Gaussian Data,” IEEE Trans. On Signal Processing, Vol. 45, pp. 927-934, Apr. 1997.
19. P. Händel, “Predictive Digital Filtering of Sinusoidal Signals”, EEE Trans. on Signal Processing, Vol. 46, No. 2, pp. 364-374, February 1968.
20. N. Wiener, “Extrapolation, Interpolation and Smoothing of Stationary Time Series, with Engineering Applications”, New York, Wiley, 1949.
21. G.E.P Box, G.M. Jenkins, “Time Series Analysis: Forecasting and Control”, Holden-Day, San Francisco, 1970.
22. J. D. Hamilton, “Time Series Analysis”, Princeton University Press, ISBN: 9780691042893, 1994.
23. B. Widrow, S.D. Stearns, “Adaptive Signal Processing”, Prentice Hall ed., 1985.
24. S. M. Kuo and D. R. Morgan, “Active Noise Control Systems’ - Algorithms and DSP Implementations” New York: Wiley, 1996.
25. E. A. Wan, “Adjoint LMS: an efficient alternative to the filtered-X LMS and multiple error LMS algorithms,” in Proc. IEEE ICASSP-1996, pp. 1842–1845, 1996.
26. S. Shaffer and C. S. Williams, “The filtered error LMS algorithm,” in Proc. IEEE Int. Conf. Acoust., Speech, Signal Process., Boston, USA, Apr. 1983, pp. 41-44.

27. S. C. Douglas, “An efficient implementation of the modified filtered-x LMS algorithm,” IEEE Signal Process. Lett., vol. 4, no. 10, pp. 286-288, Oct. 1997.
28. S. J. Eliott and P. A. Nelson, “Active noise control,” IEEE Signal Processing Mag., vol. 10, no. 4, pp. 12-35, Oct. 1993.