POLITECNICO DI MILANO
Scuola di Ingegneria dell'Informazione

POLO TERRITORIALE DI COMO

Master of Science in
Computer Engineering

An Empirical Weight Update approach for NonLinear Active Noise Control

Supervisor:	**Prof. Luigi PIRODDI**	
Master Graduation Thesis by:	**Emanuele SPIRITI**	**matr. 750316**
	Simone MORICI	**matr. 754596**

Academic Year 2011/12

POLITECNICO DI MILANO
Scuola di Ingegneria dell'Informazione

POLO TERRITORIALE DI COMO

Corso di Laurea Specialistica in
Ingegneria Informatica

An Empirical Weight Update approach for NonLinear Active Noise Control

Relatore:	**Prof. Luigi PIRODDI**	
Tesi di Laurea di:	**Emanuele SPIRITI**	**matr. 750316**
	Simone MORICI	**matr. 754596**

Anno Accademico 2011/12

SOMMARIO: Spesso, nei problemi di Controllo Attivo del Rumore (ANC), ci si imbatte in non-linearità come la saturazione e la distorsione dei microfoni e degli altoparlanti. Per affrontare questo tipo di problema devono essere impiegati dei filtri non-lineari insieme ad un algoritmo adattivo capace di aggiornarne i parametri. Il Controllo Attivo Non-lineare del Rumore (NANC) è particolarmente complicato se il cammino secondario è non-lineare, poiché in questo caso il meccanismo di update degli algoritmi di tipo Least Mean Squares (LMS) deve tener conto del gradiente del cammino secondario con ricorsioni computazionalmente pesanti. In questa tesi è proposto un metodo più semplice e computazionalmente meno esigente che evita il calcolo del gradiente dell'errore e si basa su valutazioni dirette della funzione di costo. Il metodo proposto può anche essere utilizzato per affrontare il problema della selezione della struttura del modello, un altro punto cruciale per la riduzione dei tempi di calcolo. Sono riportate alcune simulazioni che mostrano l'efficacia dell'algoritmo.

ABSTRACT: Nonlinear effects, such as saturation and distortion of microphones and loudspeakers, are often experienced in complex Active Noise Control (ANC) problem settings. A nonlinear control filter must be employed to adequately address these issues, together with an ad hoc adaptive algorithm to tune its parameters. The nonlinear ANC (NANC) problem is particularly involved if the secondary path is nonlinear, since in that case the controller weight update mechanism of Least Mean Squares (LMS)-type algorithms must account for the secondary path gradient with computationally heavy recursions. A simpler and computationally less demanding approach is here proposed that avoids the use of the error gradient and relies on direct cost function evaluations. The proposed method can also tackle the model structure selection problem, which is crucial to reduce the on-line computational effort. Some simulations are reported to show the effectiveness of the algorithm.

Ringraziamenti

A conclusione del percorso di studi da noi intrapreso circa cinque anni fa vorremmo ringraziare tutti quelli che ci hanno sostenuto. Desideriamo innanzitutto ringraziare il Prof. Luigi Piroddi per la sua disponibilità, competenza e professionalità con le quali ci ha sempre invogliato a migliorarci. In particolare:

Ringrazio i miei genitori, Roberto e Vittoria, che mi hanno sempre sostenuto sia moralmente sia economicamente in questi anni, grazie per i vostri consigli, per le vostre critiche che mi hanno fatto crescere. Grazie a mio fratello Francesco e sua moglie Valentina per i momenti di svago che mi hanno saputo regalare e a tutti i miei parenti per aver creduto sempre in me. Grazie a Edo, Maura, Martina, Andrea, Ricky, Silvia, Elena, Christian, Elio e Roby per aver saputo rallegrare le mie giornate anche nei giorni più grigi. Grazie a Vale, Luca e Lucio che mi hanno adottato per diversi mesi. Grazie a Caddo per l'aiuto e il suo delirio. Grazie ad Angelica che riesce sempre a farmi sorridere. Ringrazio tutti i compagni di viaggio con i quali ho condiviso le gioie e i dolori dell'università. Grazie a Simone, compagno di studi e amico con cui ho avuto il piacere di concludere questa mia esperienza. Un ringraziamento a tutti coloro che non ho citato e che mi hanno sopportato in questi anni. Infine, un grazie a me stesso.

Como, 19 Settembre 2012

Emanuele Spiriti

Ringrazio tutta la mia famiglia per avermi sostenuto lungo tutto il percorso di studi, tutti i miei amici per i loro preziosi consigli e per il loro supporto in uno dei momenti più difficili della mia vita. Un grazie a Roberto che ha riletto parte delle bozze, fornendo preziosi consigli per poterne migliorare la stesura. Un caloroso saluto va a tutti i compagni che ho conosciuto, dei quali riserverò un ricordo speciale che mi accompagnerà per il resto della mia esistenza. In particolare, ringrazio Emanuele, amico e studente instancabile, con cui ho condiviso ogni giornata di lavoro degli studi magistrali.

Como, 19 Settembre 2012

Simone Morici

Contents

Chapter 1

Introduction

1.1 Introduction

Noise reduction is important in order to improve the quality of life and to prevent hearing loss, therefore more and more severe thresholds have been defined through new regulations and laws. In particular, Active Noise Control (ANC) (see, e.g. [10]) concerns the attenuation of acoustic noise using secondary acoustic sources and exploiting the principle of destructive interference. Adaptive feedforward schemes are more often used since sound transmission involves delays and time-varying dynamics, which makes the feedback control design more difficult. An adaptive digital filter is employed as controller, which processes the input reference signal (correlated with the acoustic noise), to produce the driving signal for the secondary acoustic source. The filter parameters (or weights) are tuned based on the residual noise measured by the error microphone. The weight update rule is the core of the feedforward ANC scheme, and is typically a simple gradient-based method of the Least Mean Squares (LMS) family, such as the Filtered-x Least Mean Squares (FxLMS) and the Filtered-u LMS (FuLMS) algorithms [10]. The described system is typically subject to various sources of nonlinearity, such as distortion or saturation on the involved microphones, amplifiers, loudspeakers [23]. The noise signal may itself present nonlinear characteristics [16]. For these reasons several nonlinear ANC (NANC) methods have been recently introduced in the literature, based on different nonlinear model structures of the

control filter. Adaptive algorithms of the LMS family can be extended to several of those filter types, in the simplifying assumption that the secondary path (SP) dynamics is linear (see e.g. [26, 11]). The linearity of the secondary path together with the special structure of the control filter allows in fact a commutation of the secondary path with the linear portion of the control filter (containing the weights) as in the FxLMS scheme. In the more general case where the secondary path is nonlinear, this commutation is no longer feasible and the adaptation process remains unavoidably indirect, in that the desired filter output signal is not accessible. As a result, the gradient of the cost function with respect to the weights depends on the input-output gradient of the secondary path through complex recursions, as explained e.g. in [30] and in [18]. Accordingly, the resulting gradient-based adaptation algorithms are more complex (see e.g. [18]). As noted in [18], the indirect structure of the adaptation process is also a complication for model structure selection, since the available algorithms [4] for on-line model selection are based on a direct identification setting. In practice, this task is essential since the model size (in terms of number of weights) is often huge, to provide the model with the necessary descriptive flexibility, to the detriment of the computational cost which should be kept as low as possible for on-line operation. Notice also that model overparametrization is a well known cause of several undesired effects in identification problems, such as overfitting, parameter fluctuation, poor model generalization capabilities, local minima, and even model instability (see, e.g., [1], [20]). Besides that, model selection may be necessary to track structural variations of the nonlinear dynamics of the system.

To circumvent these problems a non-gradient-based error minimization algorithm is developed in this work for the adaptation of general nonlinear control filters, inspired by Stochastic Approximation (SA) methods. These are optimization algorithms that operate without directly exploiting the gradient of the loss function, but using only measurements of the function itself. These algorithms can be advantageous in ANC since they can operate without the knowledge of the secondary path dynamics, which is required for gradient calculation only. Unfortunately, in order to obtain reliable measurements of the loss function, a transient time must be allowed after each parameter perturbation is effected, with a consequent increase in the convergence time, and

a temporary performance loss may also be experienced.

In this thesis, an SA method is employed to address the NANC problem with a different purpose than avoiding the use of a SP model. Rather, the gradient-free optimization feature is exploited to circumvent the costly recursions required to account for the SP input-output gradient in the general NANC adaptation schemes. The estimated SP model is exploited to perform a virtual backward reconstruction of the effects of a given parameter perturbation, accelerating the process convergence. This also allows in particular the parallel adaptation of multiple filter instances, subject to different perturbations, a property which can be used both for plain weight update and model selection purposes. Accordingly, an algorithm is here proposed that uses local information obtained by multiple evaluations of the loss function to establish a convenient update direction in the parameter space.

The thesis is organized as follows. An overview of the most important aspects of Active Noise Control and its nonlinear variant are presented in chapter 2. Chapter 3 is dedicated to the description and the formal definition of the proposed Empirical Weight Update (EWU) algorithm, while its usage for model selection purposes is described in chapter 4. Chapter 5 reports all the tests performed to stress and evaluate the algorithm accuracy. A performance analysis of the Empirical Weight Update is performed in 6, using both analytical and experimental arguments. Some conclusions are reported in chapter 7.

Chapter 2

ANC And NANC Overview

2.1 History of ANC

In the past the noise reduction problem was addressed only with passive techniques, based on combining different absorbing materials in order to reduce the unwanted interference. The main issues related to this approach are:

- the lower the frequency to be canceled, the thicker the absorbing material is required to be. This is a critical issue on small and moving objects such as cars, boats or planes because of the weight, dimension and cost of the absorbing material.
- usually the frequency limit above which passive techniques work well is around $1kHz$ (see [8])

As a consequence, active techniques, that work particularly well for low frequencies, were developed; usually they are combined with passive ones in order to cancel the disturbances on the whole audible spectrum. Active techniques rely on the generation of a suitable in-phase interfering signal. This concept was introduced (see figure 2.1) in the first Active Noise Control patent, registered by Paul Lueg in 1936. In the section labelled $fig.1$ the scheme adopts a microphone for the detection of the interference name it s_1, an electronic system named V that accounts for the generation of a suitable interfering signal named s_2 which must be reproduced by the speaker denoted by the

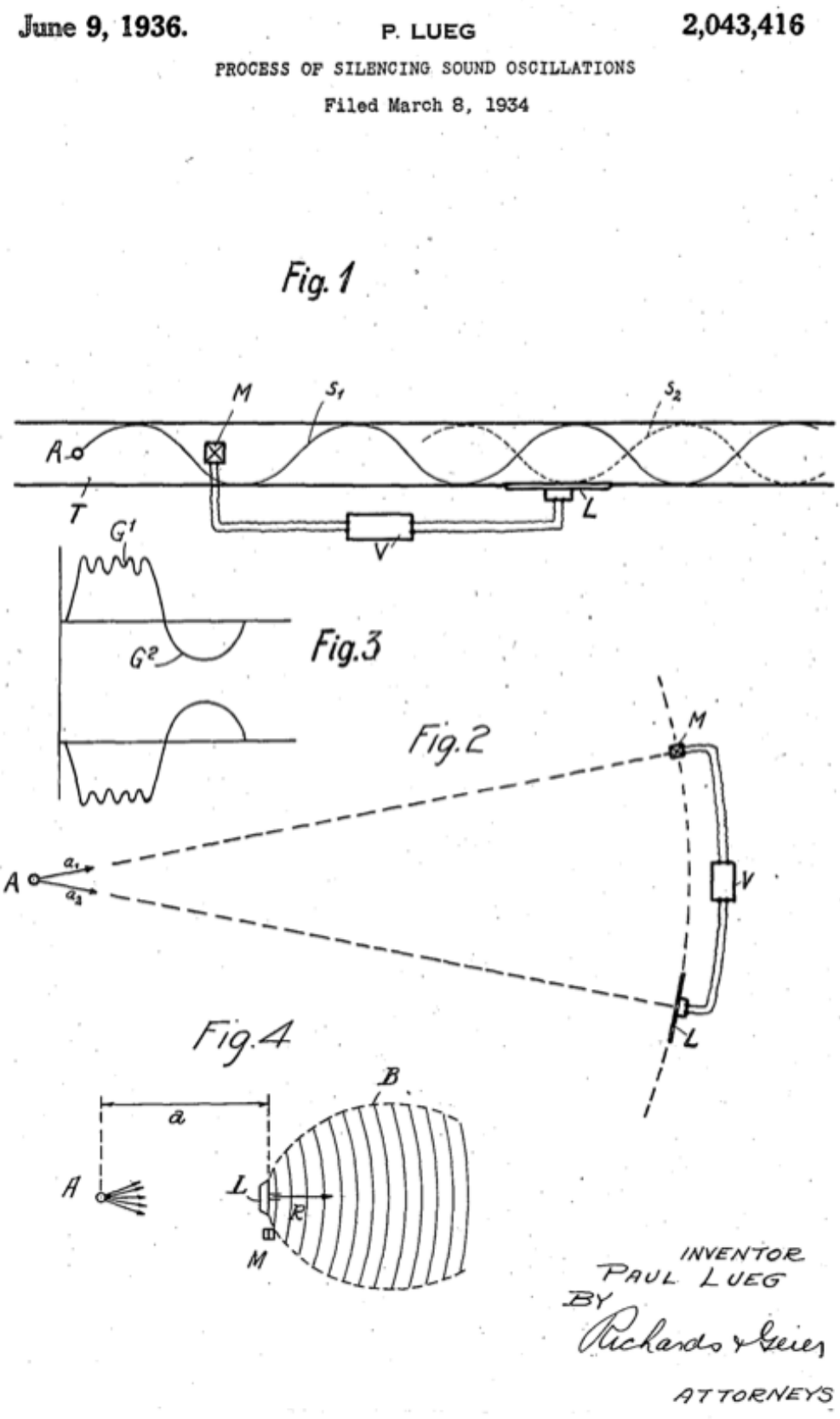

Figure 2.1: First page of Paul Lueg's patent

letter L. The scheme is similar to the ones adopted nowadays for controlling the interference of plane waves in ducts if V is interpreted as a digital filter. The main difficulties in the practical implementation of this scheme were due to the filter V and only with the advent of inexpensive digital signal processors it was possible to build the first practical application, relying upon the work done by Kido and Ross in the second half of seventies. In [29] the authors were the first to define an adaptive noise canceling scheme. This led to design an appropriate filter V which was able to cope with time varying interferences.

2.2 ANC Overview

Active Noise Control represents a group of algorithms and schemes with the goal of reducing an unwanted noise in a specific environment, based on active control methods and adaptive algorithms. A sensor is used to measure the interference and the collected signal is then processed by a DSP which implements an algorithm for the adaptation of the filter coefficients. This produces an output used for driving a speaker that is in charge of producing the appropriate cancelling interference in the acoustical domain. The two main scheme categories are "feedforward ANC" (figure 2.3) and "feedback ANC" (figure 2.4), the choice between the two schemes depending the application. The main difference between the two schemes is the signal used to update the filter coefficients. In the feedforward approach the reference signal is measured with a microphone or a sensor, depending on the frequency content of the noise, and the information is sent to the adaptation algorithm along with the measured error. The updating of the coefficients is then performed based on these two signals. In the feedback scheme only the measured error is used to update the filter coefficients. The latter approach is useful when it is not physically possible to measure the reference signal. Its main drawback is that it works well only with narrowband noise [15].

2.2.1 Mathematical Aspects

It is useful to take figure 2.5 as a reference scheme where:

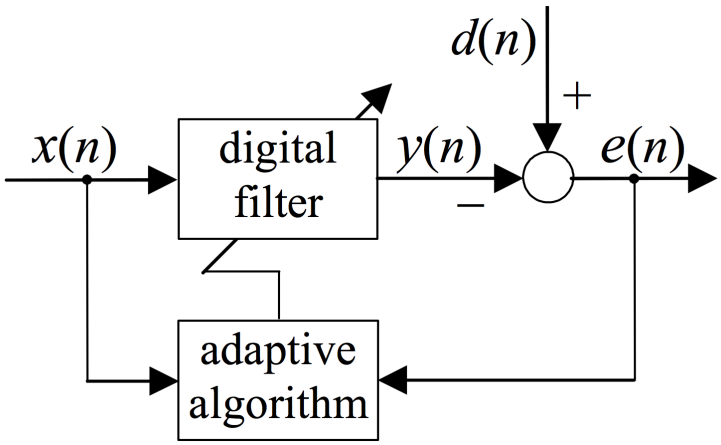

Figure 2.2: General ANC scheme: d(n) is the desired response (the noise filtered by the primary path), x(n) the noise reference signal, y(n) the filter output and e(n) the error

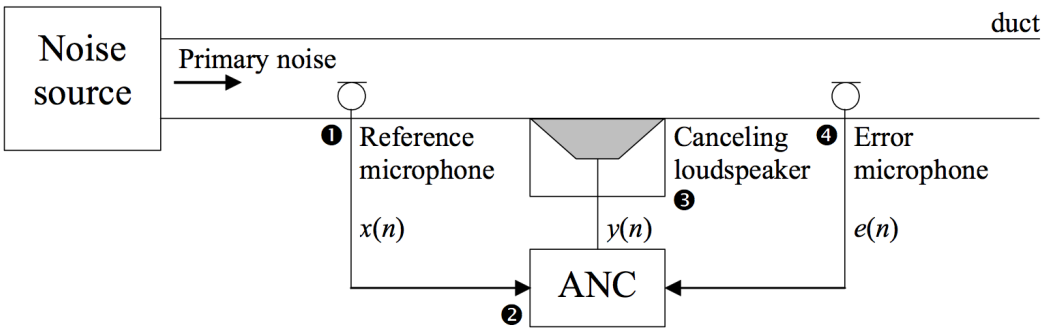

Figure 2.3: General feedforward ANC scheme

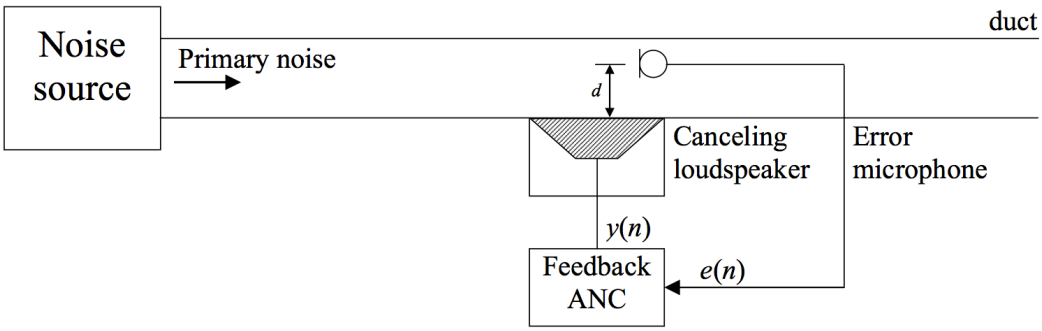

Figure 2.4: Feedback ANC general scheme

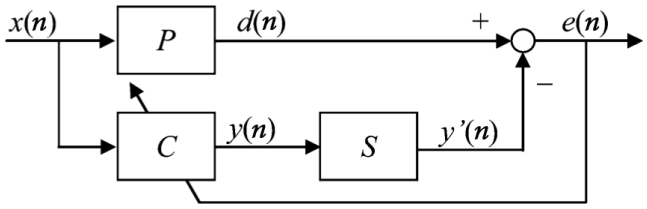

Figure 2.5: ANC reference scheme

- $x(n)$ is the noise
- P is the primary path transfer function which accounts for the modifications of the interference until it reaches the summing junction
- C is the controller filter
- $y(n)$ is the controller output
- S is the secondary path transfer function
- $y'(k)$ is the secondary path output signal used to drive the speaker
- $e(n)$ is the error signal used to adapt the filter coefficients of the controller

In the most straightforward implementation of an ANC scheme the controller is an adaptive Finite Impulse Response (FIR) filter defined as:

$$y(n) = \sum_{l=0}^{L-1} w_l x(n-l) = \boldsymbol{w}(n)^T \boldsymbol{x}(n)$$

where:

- $\boldsymbol{x}(n) = [x(n)\, x(n-1) \cdots x(n-L+1)]^T$ is the vector containing the last L samples of the reference input signal
- $\boldsymbol{w} = [w_0(n)\, w_1(n) \cdots w_{L-1}(n)]^T$ is the vector of the coefficients of the adaptive filter
- L is the length of the filter

The goal of ANC is to adapt the weights $\boldsymbol{w}(n)$ in order to minimize $e(n) = d(n) - y(n) = d(n) - \boldsymbol{w}(n)\boldsymbol{x}(n)$. More specifically the cost function

$$J(n) = E[e^2(n)] = E[(d(n) - \boldsymbol{w}^T(n)\boldsymbol{x}(n))^2]$$

is typically employed. Thanks to the assumption that $\boldsymbol{w}(n)$ is a deterministic sequence, it can be rewritten as:

$$J(n) = E[e^2(n)] = E[d^2(n)] - 2\boldsymbol{p}^T\boldsymbol{w}(n) + \boldsymbol{w}^T(n)\boldsymbol{R}\boldsymbol{w}(n)$$

where:

- $\frac{\partial J(n)}{\partial \boldsymbol{w}(n)} = -2\boldsymbol{p} + \boldsymbol{R}\boldsymbol{w}(n)$ is the gradient of the cost function with respect to w
- $\boldsymbol{p}$ is the correlation between the input signal x and the reference signal d
- $\boldsymbol{R}$ is the autocorrelation matrix of x
- $\frac{\partial^2 J(n)}{\partial \boldsymbol{w}(n)^2}$ is the Hessian matrix

The Wiener Filtering solution can be adopted, but, since the statistics of the signals are usually unknown, only iterative methods can be employed in practice. Moreover, in complex cases the matrices involved in the Wiener solution are really big and this can lead to a significant increase in the computational load. Many iterative methods are based on the Newton-Raphson algorithm, which defines the coefficient update rule as follows:

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \Big(\frac{\partial^2 J(n)}{\partial \boldsymbol{w}(n)^2}\Big)^{-1} \frac{\partial J(n)}{\partial \boldsymbol{w}(n)}$$

In order to reduce the complexity, the steepest descent algorithm was derived from the Newton-Rapshon. Since the main computational problems are generated by the inversion of the autocorrelation matrix R, the Hessian is replaced by a variable μ; the convergence of the weight vector is maintained, but it is also slowed down and the gradient still relies on $\boldsymbol{p}$ and $\boldsymbol{R}$.

The problem of dealing with the statistics of the signals was solved by Widrow in 1970 with the Least Mean Square algorithm (LMS from now on),

which uses the instantaneous squared error to approximate the mean square error: $\hat{J}(n) = e^2(n)$. The expression of the gradient becomes

$$\frac{\partial \hat{J}(n)}{\partial w(n)} = 2\frac{\partial e(n)}{\partial w(n)}e(n) = -2\boldsymbol{x}(n)e(n)$$

and consequently the weight update is finally defined as

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \mu\boldsymbol{x}(n)e(n).$$

As in the previous algorithms the convergence is ensured, but small oscillations around the optimal value of the coefficients are expected, due to the approximations involved. The greater the gain μ, the more this phenomenon is emphasized, with the advantage of a faster speed of convergence and the risk of compromising the stability of the algorithm. The resulting error, with respect to the Wiener-Hopf optimum, is often named Excess Mean Square Error and is described in [10].

2.2.2 The Secondary Path Implications

Accounting only for the primary path is usually not sufficient in a feedforward scheme, since some additional elements are required such as a D/A converter, a reconstruction filter and a power amplifier. Moreover, the acoustic path from the speaker to the summing junction and the transfer function of the speaker must be accounted for as well. Since the signal output of the the primary and secondary paths are summed in the acoustical domain and measured by an error microphone, their transfer functions must be taken into account. The path between the summing junction and the microphone and at least an antialiasing filter and a A/D converter must be considered in a real scenario.

It is common to group together all the above mentioned elements in a filter named $S(z)$ which works as in figure 2.6. The z-transform of the error is defined as:

$$E(z) = [P(z) - W(z)S(z)]X(z)$$

As a consequence, the optimal filter coefficients are obtained as:

$$W_{opt}(z) = \frac{P(z)}{S(z)}$$

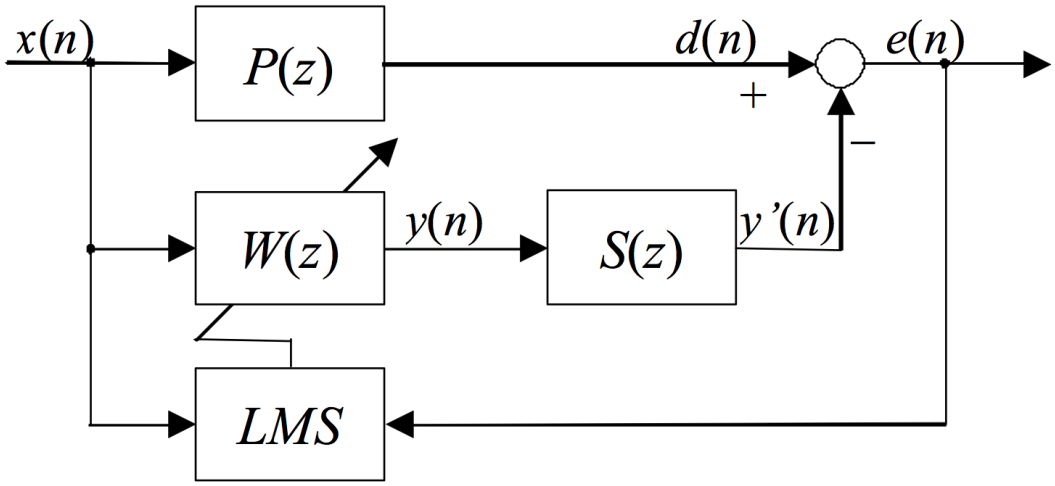

Figure 2.6: Secondary path simplified reference scheme

The most straightforward solution to account for the presence of the secondary path would be to calculate an inverse of the filter $S(z)$ and insert it between $W(z)$ and $S(z)$, but as explained in [10], this is possible only if $S(z)$ is minimum phase, which is generally not true in the acoustical context. One must also take into account that, even if it is possible to obtain a good estimate of the filter, the secondary path may also be sometimes time variant.

2.2.3 The FxLMS Variant

To take properly the secondary path into account in the LMS, in 1981 Widrow and Burgess derived independently the FxLMS algorithm, bypassing the problem of the inversion of $S(z)$ stated above. Their idea was to make a filter $\widehat{S}(z)$ approximate $S(z)$ and use it to filter the reference signal, in order to have a consistent weight update. The adopted scheme is represented in 2.7.

Since the standard LMS weight update formula is given by

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \mu \frac{\partial e^T(n)}{\boldsymbol{w}(n)} e(n)$$

and, as we can see in figure 2.7, the error is defined as

$$e(n) = d(n) - y'(n) = d(n) - s(n) * y(n) = d(n) - s(n) * [\boldsymbol{w}^T(n)\boldsymbol{x}(n)]$$

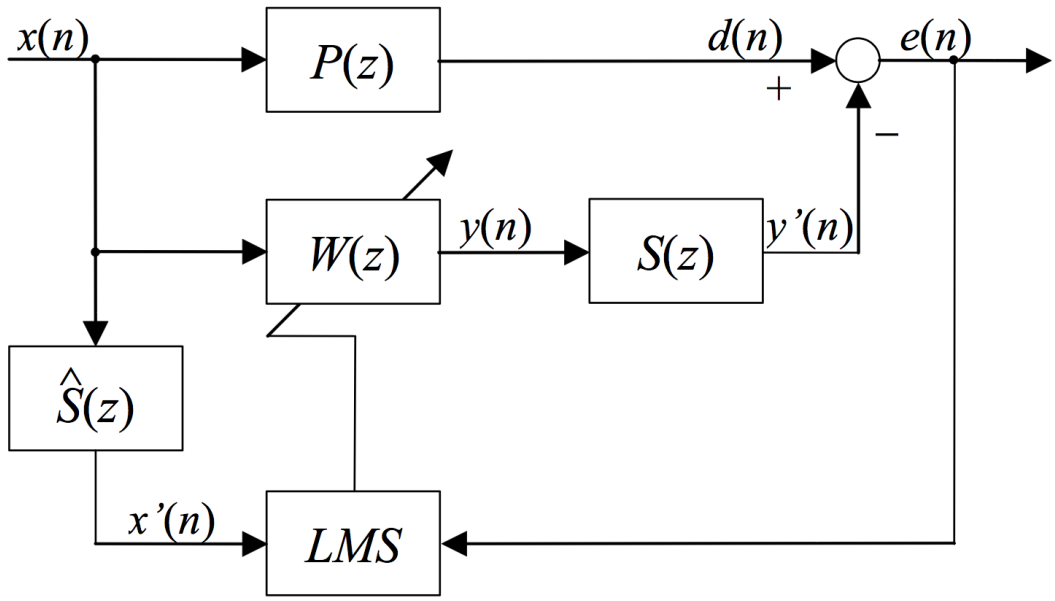

Figure 2.7: FxLMS reference scheme

the error gradient can be obtained as

$$\frac{\partial e(n)}{\partial \boldsymbol{w}(n)} = -s(n) * \boldsymbol{x}(n) = -\boldsymbol{x}'(n)$$

as a direct consequence of the previous two formulas. The name Filtered X Least Mean Squares comes straightly from the fact that the input signal is filtered by an estimate of the secondary path. When this scheme is adopted the filter can tolerate errors in the estimation of $\widehat{S}(z)$ if the phase difference between $\widehat{S}(z)$ and $S(z)$ is below 90 degrees, provided the gain is sufficiently small [10].

2.3 NANC Overview

In many cases linear algorithms are able to reach good noise attenuation in practice, because the linear approximation is sufficiently accurate, but when nonlinearities become non negligible, a new suitable class of models and algorithms must be used. Such methods are collectively denoted Nonlinear ANC (NANC) methods.

Nonlinearities arise when:

- The noise possesses chaotic features [16];
- Hardware components of the ANC system saturate or operate out of their linear dynamic range [12];
- High noise pressure levels cause the primary path to be nonlinear and generate distortions at the canceling point [26];
- The electronics and the speaker excite harmonics related to the frequencies of interest [23];
- Components of the ANC system suffer of aging and corrosion [11].

NANC algorithms are also used when the secondary path is not minimum phase and cannot be inverted [3, 25]. The ability of a filter to represent nonlinearities can lead to the installation of cheaper hardware that has a small linear dynamic range.

2.3.1 NANC With Linear Secondary Path

Several types of filters can be used in a NANC scenario, such as the Volterra expansion described in [25, 26], which is defined as

$$y(n) = \sum_{m_1=0}^{L} w_1(m_1;n)x(n-m_1) + \sum_{m_1=0}^{L}\sum_{m_2=m_1}^{L} w_2(m_1,m_2;n)x(n-m_1)x(n-m_2)$$

where $x(n)$ and $y(n)$ are the input and output signals and w_1 and w_2 are two time varying coefficients. As explained in [18] the usage of this particular FIR-type filter can cause several problems:

- It is not possible to model systems with a complex static behavior, since the nonlinear FIR filter structure admits only one static point for a given constant input value;
- The model limits the possibility of representing more complex nonlinearities than the second order ones, due to the excessive number of parameters that would be required;

- A FIR filter requires usually a bigger number of parameters L than an Infinite Impulse Response (IIR) filter. This leads to a poor parameter estimation accuracy that can have an impact on the model robustness. In fact, it is possible to have many redundant terms and the system can show wrong correlation between the input signal and the model parameters. Other unwanted effects may also arise due to this overparametrization issue [1, 20]

The main advantage of this model is represented by the fact that it allows the usage of the FxLMS algorithm with small modifications, because the second order Volterra filter can be represented through a linear model like

$$y(n) = \mathbf{w}(n)^T \mathbf{x}(n)$$

where

$$\mathbf{w}(n) = [w_1(0;n)\, w_1(1;n)\, \ldots\, w_1(L-1;n)\, w_2(0,0;n)\, w_2(0,1,n) \ldots\, w_2(L-1,L-1;n)]^T$$

and

$$\mathbf{x}(n) = [x(n)\, x(n-1) \ldots x(n-L+1)\, x^2(n)\, x(n)x(n-1)\, \ldots\, x^2(n-L+1)].$$

This kind of representation, along with the assumption of a linear secondary path, allows the usage of the the standard FxLMS formula for the weight update

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu \mathbf{x}'(n) e(n)$$

This control scheme is commonly denoted Volterra FxLMS (VFxLMS).

A different way to approach the problem is represented by the FLANN (Functional-Link Artificial Neural Network) structure. The input vector $\mathbf{x}(n)$ is mapped into a new signal $\mathbf{x}'(n)$ through nonlinear functions, in order to increase the ability of the system to represent non linearities and subsequently the FxLMS algorithm can still be applied with minor modifications. There are various algorithms that are based on this kind of linear filters, that mainly differ for the strategy adopted for the mapping function:

- the Filtered-s LMS algorithm described in [5] uses trigonometric functional expansion. In [7, 6] a filter-bank featured scheme that implements

the Filtered-s LMS in the frequency domain is also proposed, in order to reduce the complexity

- in [22] piecewise linear expansion is presented

The method proposed by Sicuranza and Carini [22] works with both Volterra filters and FLANN structures.

Kuo and Wu proposed in [11] another adaptation of the FxLMS for the nonlinear case. They rely on a different family of filters named Adaptive Output Error Bilinear Filters, which can be represented as:

$$y(n) = \sum_{i=0}^{L} a_i(n)x(n-i) + \sum_{j=1}^{L} b_j(n)y(n-j) + \sum_{i=0}^{L}\sum_{j=1}^{L} c_{i,j}(n)x(n-i)y(n-j)$$

where:

- y is the output signal of the controller
- x is the input signal, sensed by the reference microphone
- a_i, b_j and $c_{i,j}$ are the adaptive coefficients corresponding to the FIR part, the IIR part and the second order input-output part of the filter respectively

Following a similar procedure adopted for the VFxLMS, it is possible to rewrite the model structure as

$$\mathbf{w}(n) = [a_0(n) \ \dots \ a_L(n)\, b_0(n) \ \dots \ b_L(n)\, c_{0,1}(n) \ \dots \ c_{L,L}(n)]^T$$

$$\mathbf{x}(n) = [x(n) \ \dots x(n-L)\, y(n-1) \ \dots \ y(n-L)\, x(n)y(n-1) \ \dots \ x(n-L)y(n-L)]^T$$

and filter element-wise the signal x with the secondary path $S(z)$ to obtain the signal x' required by the FxLMS algorithm. The main advantage is that they are Infinite Impulse Response filters and thus are able to represent more complex nonlinearities than the Volterra ones, using less coefficients. The main drawbacks of Volterra filters related to the overparametrization issue are solved, but, as explained by the authors of the paper, the filter can become unstable if the algorithm is not carefully designed. Another drawback is represented by the fact that the error function is nonlinear with respect to the

coefficient values and thus the adaptation algorithm can be trapped in a local minimum. In the literature it is common to refer to this particular scheme as BFxLMS, which stands for Bilinear FxLMS.

2.3.2 NANC With Nonlinear Secondary Path

All the algorithms presented until now assume a linear secondary path. The main assumption of all the FxLMS based algorithm presented so far is that the linear SP can be swapped with the controller which is linear in the parameters so that its output can be interpreted as a linear regression. When nonlinearities appear in the SP, more complex algorithms must be used, since the swapping does not generally apply. Snyder and Tanaka in [23] defined a control structure based on MLANNs (Multiple Link Artificial Neural Network) filters. They are used both for the controller and the SP path but in the second case the coefficients are fixed and never adapted. The authors show that the algorithm is an extension of the FxLMS for the non linear case and that the structure of the filter is an extension of a neural network. The authors show that the predictability of the results is the main problem concerning the usage of a neural network, thus, as they proposed, more work must be done in this direction before put aside the FxLMS based algorithms.

In [30] Zhou and DeBrunner proposed a method based on the so called virtual secondary path, which can work as a standard FxLMS if the SP is linear, but it is capable of generalizing the concept also to nonlinear SPs. The controller accounts for nonlinearities following the concept of linear function expansion already used in [5, 22]. Moreover it is worth to mention that the proposed method subsumes both the VFxLMS ([26]) and the FsLMS ([5]), being able to reduce the computational complexity of the latter.

In order to increase the representation capabilities of nonlinearities, the NARX (Nonlinear AutoRegressive models with eXogenous variables) model structure was proposed in [13].

The NARX model is a recursive input-output relationship that is defined as

$$y(n) = f(y(n-1), \cdots, y(n-L), x(n), \cdots, x(n-L))$$

where $f(\cdot)$ is a function that meshes up different lagged input and outputs; $x(\cdot)$ and $y(\cdot)$ are respectively the input and the output signals and L is the maximum lag admitted in the structure. $f(\cdot)$ can be formally defined in different ways, such as neural networks and wavelet networks, but one of the most common is the polynomial one:

$$\sum_{m=0}^{l}\sum_{p=0}^{m}\sum_{n_1=1}^{L}\cdots\sum_{n_m=0}^{L} c_{p,m-p}(n_1,\cdots,n_m)\times\prod_{i=1}^{p} y(n-n_i)\prod_{i=p+1}^{m} x(n-n_i) \quad (2.1)$$

where l is the maximum degree of the polynomial expansion and $c_{p,m-p}(n_1,\cdots,n_m)$ are constant parameters. Looking at formula 2.1 is easy to notice that subsumes both the Volterra and the Bilinear models. Again the output $y(n)$ can be interpreted as a linear regression:

$$y(n) = \boldsymbol{\varphi}(n)^T\boldsymbol{\theta}$$

where $\boldsymbol{\varphi}(n)$ is the regressor vector and $\boldsymbol{\theta}$ is the correspondent vector of the associated coefficients. It introduces new regressors with respect to Volterra and Bilinear models, therefore, in order to retain only the regressors that better describe the filter, a model structure selection algorithm should be used to reduce greatly the coefficients that must be adapted and consequently the overall complexity. Most of the literature for the model structure selection has been developed for NARX and many algorithms, such as FROE (Forward-Regression Orthogonal Estimator) [2], SEMP (Simulation Error Minimization With Pruning) [20] and FRA (Fast Recursive Algorithm) [14] rely on it.

An adaptation law suitable for NARX models is the so called Nonlinear Filtered-Gradient Least Mean Squares (NFGLMS) [18]. One of the main assumptions of the FxLMS is that the SP can be swapped with the controller, but since this operation can not be done with of nonlinear SPs, Napoli and Piroddi tried to solve the problem using the recursive calculation of the error gradient. The NFGLMS faces the NANC problem by performing an indirect model identification, which implies the control structure estimation by off-line processing (discussed later in chapter 4).

Since the NFGLMS is taken as a benchmark in this work, an exhaustive explanation is provided next. Let's consider a system like the one pictured in figure 2.5, assuming a deterministic polynomial NARX structure described

formula (2.1) for both the SP and controller. The goal is to minimize the output error of an NANC system, with both the primary and the secondary paths. Being nonlinear, the update equation used by NFGLMS is based on the steepest descent algorithm and is derived as follows:

$$\boldsymbol{\theta}(n+1) = \boldsymbol{\theta}(n) - \frac{\mu}{2}\Big(\frac{\partial e(n)^2}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta(n)}}\Big)^T$$
$$\boldsymbol{\theta}(n+1) = \boldsymbol{\theta}(n) - \mu\Big(\frac{\partial e(n)}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta(n)}}\Big)^T e(n)$$
$$\boldsymbol{\theta}(n+1) = \boldsymbol{\theta}(n) + \mu\Big(\frac{\partial y'(n)}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta(n)}}\Big)^T e(n)$$

where $\boldsymbol{\theta}$ is the coefficient vector. The gradient $\frac{\partial y'(n)}{\partial \boldsymbol{\theta}}$ is iteratively computed by means of the nonlinear filter:

$$\frac{\partial y'(n)}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta(n)}} = \sum_{j=0}^{M} \frac{\partial y'(n)}{\partial y(n-j)} y_{\boldsymbol{\theta}}(n-j) \tag{2.2}$$

where M is the memory of the secondary path (here j starts from 0 while in the algorithm found in paper [18] starts from 1). Formula (2.2) is obtained through Feintuch's assumption, which neglects the recursion based on old output gradients, with the goal to reduce the computational burden. In fact the correct filter should be:

$$\frac{\partial y'(n)}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta(n)}} = \sum_{j=1}^{M} \frac{\partial y'(n)}{\partial y'(n-j)} y'_{\boldsymbol{\theta}}(n-j) + \sum_{j=0}^{M} \frac{\partial y'(n)}{\partial y(n-j)} y_{\boldsymbol{\theta}}(n-j) \tag{2.3}$$

The same reasoning is done to compute $y_\theta(n)$, that in its simplified form looks like:

$$y_\theta(n) = \prod_{i=1}^{\bar{p}} y(n-\bar{n}_i) \prod_{i=\bar{p}+1}^{m} x(n-\bar{n}_i) \tag{2.4}$$

while it should be calculated as:

$$y_\theta(n) = \sum_{j=1}^{M} \frac{\partial y(n)}{\partial y(n-j)} y_{\boldsymbol{\theta}}(n-j) + \prod_{i=1}^{\bar{p}} y(n-\bar{n}_i) \prod_{i=\bar{p}+1}^{m} x(n-\bar{n}_i). \tag{2.5}$$

if Feintuch's assumption is not considered.

As we can read in [18], the adaptation equation structured so far includes gradient filtering of the controller output with respect to its weights. From this observation comes the name Nonlinear Filtered-Gradient LMS (NFGLMS).

In order to evaluate the consequences of Feintuch's assumption, we considered three different configurations, always keeping the recursion based on the old output gradients in (2.3):

- Maintain the equation (2.4) together with (2.2) (standard NFGLMS);
- Preserve all the recursive terms in $y_\theta(n)$ using formula (2.5), in order to obtain the correct gradient and, (2.2) to calculate the controller output (name it Full-Gradient);
- Hold only a fixed number $\bar{M}$ of recursive terms, in order to achieve an hybrid setup between the full $y_\theta(n)$ gradient and the simplified one described by formula (2.4) (name it Half-Gradient). The following equation (2.6) shows the approach to calculate progressive Feintuch's approximations, where $0 \leq \bar{M} \leq M$

$$y_\theta(n) = \sum_{j=1}^{\bar{M}} \frac{\partial y(n)}{\partial y(n-j)} y_{\boldsymbol{\theta}}(n-j) + \prod_{i=1}^{\bar{p}} y(n-\bar{n}_i) \prod_{i=\bar{p}+1}^{m} x(n-\bar{n}_i). \quad (2.6)$$

A test was performed to verify the behavior of the NFGLMS with the modifications previously described. The chosen test is very simple, so that we have an overwhelming probability that differences in performance should not be attributed to the complexity of the model. Let's take a linear controller and a second order SP like:

$$y(n) = ay(n-1) + bx(n) \quad (2.7a)$$

$$y'(n) = \alpha y'(n-1) + \beta y(n) + \gamma y(n-1)y(n) \quad (2.7b)$$

The primary path is the exact product of the SP and an ideal controller with $a = 0.4$ and $b = 0.2$, while $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ are 0.1, 0.3 and 0.2 respectively. As we can see in the MSE chart reported in figure 2.8, the three algorithms have similar performances but the approximations in the NFGLMS, and the consequent loss of information, are nevertheless noticeable, even if the test is set up with starting coefficients of the controller near the ideal ones. On the other hand, when the coefficient a moves away from its correct value, a significant improvement in the performance is experienced, both with Half-Feintuch and the Full-Gradient approaches with respect to the NFGLMS, by

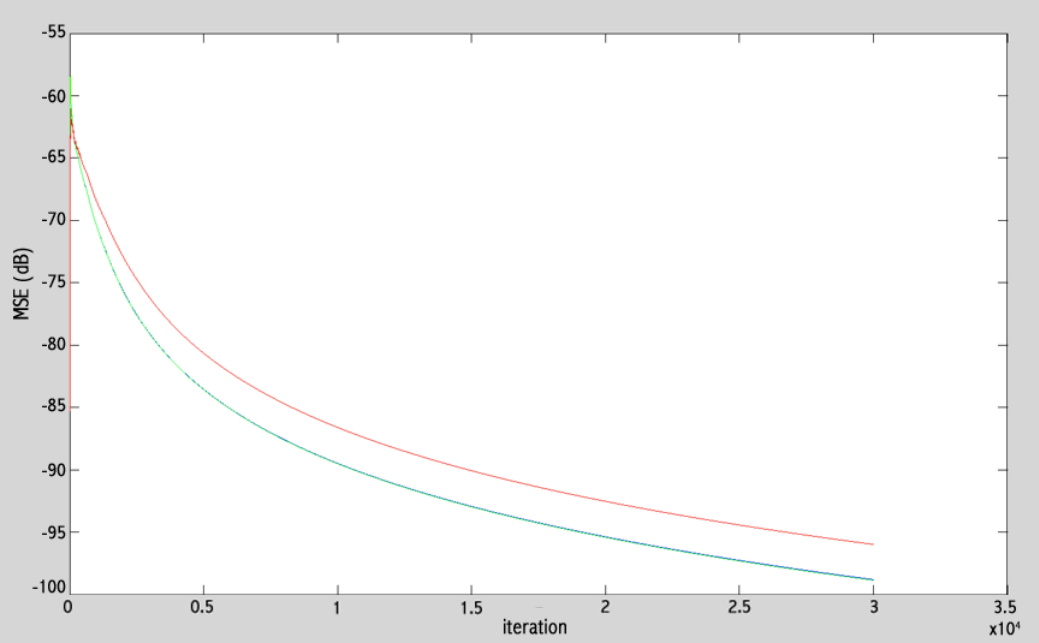

Figure 2.8: The three algorithms running with starting parameters $a = 0.25$ and $b = 0.1$, red = NFGLMS, blue = Half-Gradient approach, green = Full-Gradient

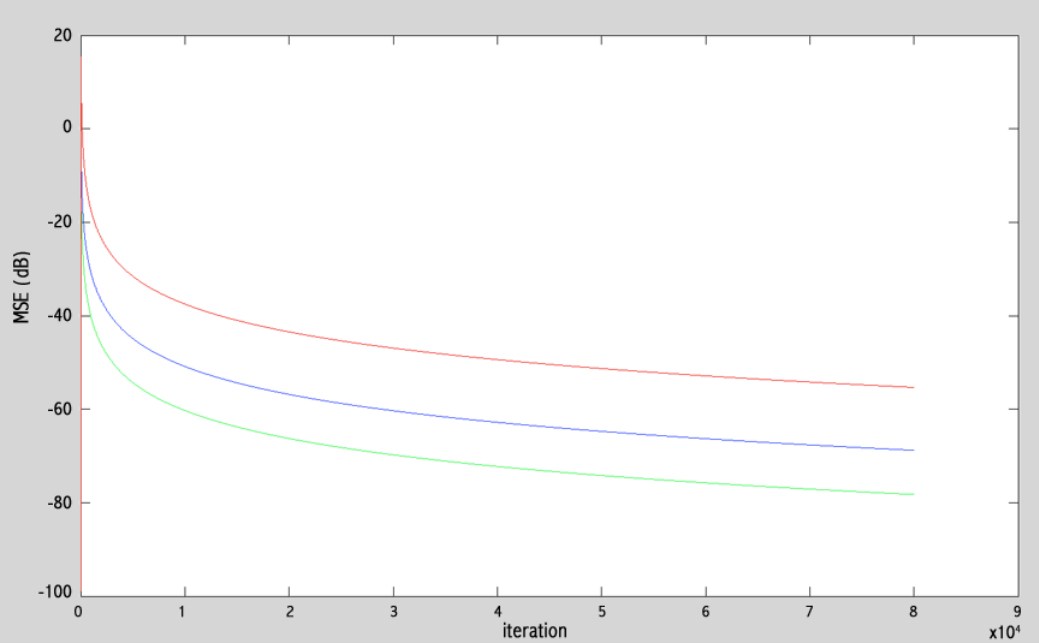

Figure 2.9: 80000 samples, $a = 1.5$, $b = 0.2$, red=NFGLMS, blue = Half-Feintuch approach, green = Full-Gradient

approximately $10 - 20$ dB. As we can see from figures 2.8 and 2.9, the more the gradient is approximated the more time is needed to reach convergence as a diverges from its optimal value. These results are in line with theory: in fact the more the gradient approximations, the least reactive the performance. In any case, it is worth noticing that the NFGLMS requires less calculations, while maintaing a comparable behavior.

Another class of algorithms that can be used in the presence of nonlinearities, even in the secondary path, is represented by the one that relies on a stochastic approach. These algorithms try to achieve the best attenuation finding a suitable controller coefficient update based on experience, hence avoiding the direct computation of the gradient. The theoretically obtainable advantages are:

- lower complexity
- easy adaptation to secondary path variations
- applicability evening the case of unknown secondary path

The main drawback is represented by the fact that convergence is often slower than with gradient methods. Stochastic methods can be grouped in two categories: those which try to estimate the gradient from successive observations like the SPSA [31] and those, like the PSO [21], which look only to the outcomes for updating the coefficients.

Stochastic Approximation (SA) methods try to solve the problem of minimizing the loss function, without directly exploiting the gradient information, which is approximated from subsequent measurements of the function itself. Their main advantage is that they do not require any type of information about the functional relationship between the parameters and the loss function to be minimized [24]. In the ANC field this means that they do not necessarily require an estimate of the secondary path. They can also achieve a computational advantage, although the lower speed of convergence with respect to gradient based ones should be considered.

Spall in [24] lists at least three factors that can give SA algorithms a potential advantage:

1. Gradient-based methods need a reliable knowledge of the input/output relationships, while SA methods can work with poor or even without system models;

2. The total cost to achieve convergence must consider also the computational cost for each iteration (typically lower in SA methods);

3. Convergence rates are based on asymptotic theory and may not be representative with finite samples.

The simplest method based on this approach is the Finite Difference Stochastic Approximation (FDSA) proposed by Kiefler and Wolfowitz in [9]. It adds a perturbation to one controller coefficient at a time and measures the outcome. An estimate of the gradient is then obtained by subtracting the outcomes and dividing the result by an interval equal to two times the applied perturbation. This is the standard definition of gradient as a vector of partial derivatives, each one computed with the difference quotients method. Analytically the i-th component of the approximated gradient is represented by:

$$g_{ni}(\theta_n) = \frac{J(\theta_n + c_n e_i) - J(\theta_n - c_n e_i)}{2c_n} \tag{2.8}$$

where:

- J is the loss function
- θ_k is the coefficient vector
- e_i is a zero-vector with a one in the i-th position
- c_k is the magnitude of the perturbation

This algorithm requires at least a number of measures of the output equal to twice the number of coefficients n_c, because two perturbations (negative and positive) are applied for each coefficient.

A smarter stochastic algorithm described by Spall in [24] is the SPSA (Simultaneous Perturbation Stochastic Approximation). With the aim of improving the performance, the perturbation is applied simultaneously to the entire set of coefficients with a randomly chosen magnitude. The number of

required measurements reduces from $2n_c$ to 2. The resulting approximation of the gradient becomes:

$$g_{ni}(\theta_n) = \frac{J(\theta_n + c_n\delta_n) - J(\theta_n - c_n\delta_n)}{2c_n\delta_{ni}} \tag{2.9}$$

where δ_n is the perturbation vector, while all the other parameters retain the previous meaning. To reach convergence, a careful choice of the elements of vector δ_n is in order. A common solution ([24, 31]) is to consider all the δ_{ni} as symmetric Bernoulli distribution. Both for the FDSA and SPSA the update rule is:

$$\theta_{n+1} = \theta_n - a_n g_n \tag{2.10}$$

In the results shown in [24] it is possible to see that SPSA reaches convergence faster than FDSA, but it follows a much longer adaptation path. This is explained by the fact that the Finite Difference Stochastic Approximation algorithm tries to follow at each step the direction towards the best coefficients while the SPSA, through the reduction of collected data, often does not follow the locally steepest descent path.

One ANC application of SPSA was operated in [31] by Zhou, Zhang, Li, and Gan both for linear and nonlinear secondary paths. The model of the SP is assumed to be unknown and the objective to reduce a periodic noise. A feedback ANC configuration was chosen. The controller is defined as

$$u(n) = c_1^c cos(\omega_1 n) + c_1^s sin(\omega_1 n) + c_2^c cos(\omega_2 n) + c_2^s sin(\omega_2 n)$$

where $u(n)$ is the output of the controller, while ω_1 and ω_2 are the frequencies of the canceling signal chosen to match the frequency content of the noise signal. The weight vector is consequently defined as

$$w = [c_1^c\, c_1^s\, c_2^c\, c_2^s]$$

Simultaneous perturbations are added and subtracted to the coefficient vector and the output error of the ANC system is collected for a predefined number of steps defined by the variable λ. The error (loss) function used in the gradient estimate was:

$$J(y(n)) = \frac{1}{2}\sum_{n=1}^{\lambda} e^2(n) \tag{2.11}$$

The results described in [31] show that the algorithm is capable of good results in terms of attenuation but the performance is strongly affected by the differences between the frequencies of the sinusoids that made up the controller and the frequency components of the noise. In fact only the magnitude of the canceling signal is adapted by the SPSA and a modification of the algorithm is required to account also for the adaptation of the frequency. It is important to notice that as in gradient methods, the algorithm is sensible to spikes since the magnitude of the correction terms is proportional to the error measured by the microphone. Since the magnitude of the correction term is not bounded, stability is not guaranteed.

Another algorithm that is useful in the presence of a nonlinear secondary path is the PSO. Particle Swarm Optimization stochastic algorithm is "based on the premise that social sharing of information among members of a species offers and evolutionary advantage" [19]. It is based on the social behavior reflected in flock of birds and unlike the algorithms presented so far is a part of global optimization algorithm. This type of algorithms as well as Genetic Algorithms [27] cannot be directly used in ANC systems because they depend on a set of errors and not on an instantaneous outcome of the error sensor. In fact PSO was introduced for batch applications where all the data is known in advance. A scheme for online ANC adaptation is proposed by Rout, Das, and Panda in [21]. The PSO relies on positioning the particles, represented by the adaptive filters, in the coefficients field. This is equivalent to say that each filter is initialized with different coefficient values. Each particle is described by its position and velocity that are modified at each update with respect to the particles that led to the best performances. Particles virtually "talk" to each other and the information of the best position is shared. The above cited PSO application for ANC exploits this characteristic to find the best controller configuration. Each particle represents a specific configuration in the parameters of the controller's dynamic filter. The algorithm can be explained as follows:

1. initialize a matrix $\mathbf{W}$ with as many columns as the number of particles P and as many rows as the order of the controller filter N with random

values in the range [0,1] and a zero matrix $\mathbf{V}$ with the same dimensions:

$$\mathbf{W} = \begin{bmatrix} w_1^1 & w_1^2 & \cdots & w_1^P \\ w_2^1 & w_2^2 & \cdots & w_2^P \\ \vdots & \vdots & \cdots & \vdots \\ w_N^1 & w_N^2 & \cdots & w_N^P \end{bmatrix}$$

2. select the i-th particle (filter) and compute the output error

3. repeat step 2 $M-1$ times and compute the MSE of the collected errors, where M represents how many samples are collected for each particle

4. increase i and repeat step 2 and 3 until $i = P$ then go to step 5

5. update $\mathbf{W}$ and $\mathbf{V}$ with the following rules:

$$\begin{aligned} \mathbf{V}_i(n) = \sigma \mathbf{V}_i(n-1) + r1(\mathbf{W}_{pbest_i} - \mathbf{W}_i(n)) \\ + r2(\mathbf{W}_{gbest} - \mathbf{W}_i(n)) \end{aligned}$$

$$\mathbf{W}_i(n) = \mathbf{W}_i(n-1) + \mathbf{V}_i(n)$$

where $\mathbf{W}_{pbest_i}$ is the position of the i-th particle that led to its personal lowest MSE, while $\mathbf{W}_{gbest}$ represents the best coefficient vector among all the $\mathbf{W}_{pbest_i}$. Parameter σ is the so called inertia coefficient that prevents the update of $\mathbf{V}$ from being a pure integrator, analogously to the leakage factor sometimes used in LMS-type algorithms. $r1$ and $r2$ are two random values taken randomly in the range [0,1].

6. repeat from step 2 to step 6 until convergence is achieved.

The block scheme of the algorithm is represented in figure 2.10. The main advantage of this particular method is that no estimates of the secondary path are used in the update of the algorithm. This theoretically means that the PSO is not affected by errors in the approximation of the secondary path. It is worth mentioning that the real SP can change due to temperature variations [15] and since online estimation is difficult, this can represent a great improvement with respect to the algorithm presented so far. Another advantage is that it generally avoids to be trapped in local minima since it is a global optimization method. In fact particles are initially spread around in the coefficients space

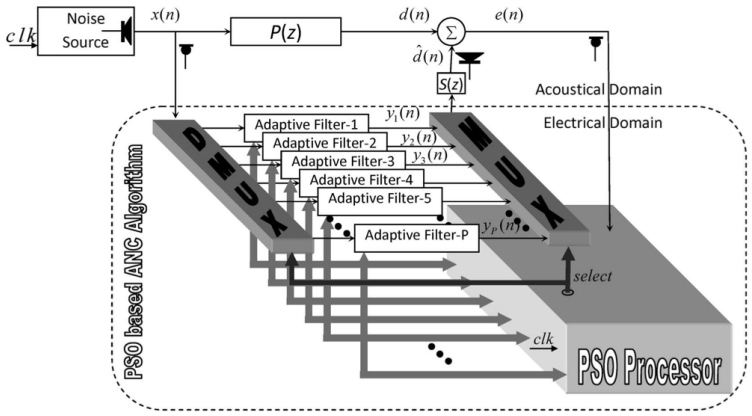

Figure 2.10: PSO-based ANC block scheme

at random. Moreover, it is not sensible to spikes in the primary signal because the value of the error is not directly part of the coefficients update and it can be easily adapted to work with on line variations of the secondary and primary path by reinitializing the particle matrices when the variation is detected.

The main disadvantage of the approach described so far is that the speed of convergence is low. The update comes only after the whole set of filters correspondent to the particles has been tested. Furthermore for each filter we must collect the error for M steps, where M is a sufficiently large value in order to measure a "clean" error (usually at least double with respect to the length of the filters, which is equal to N). In fact, previous filter values should not affect the error, since it depends from the memory of the controller and the secondary path. Subsequently, one has to correctly adjust M as well as the order of the controller and the number of particles taken into consideration. This tuning is not easy due to the unawareness of the secondary path, thus all the setup parameters need to be found empirically.

Chapter 3

The Empirical Gradient Approach

3.1 Introduction

As we have seen in the previous chapters, gradient methods suffers from the problem of explicit computation of the partial derivatives. In fact, it is quite challenging to find an automatic and efficient procedure to calculate them. Moreover, the complexity of the gradient computation is directly proportional to the order of the controller and of the secondary path. On the other hand, stochastic methods are too slow in convergence to be compared with the above mentioned methods given that each update requires a significant number of steps.

All the stochastic methods seen so far are model-free. This means that they do not rely at all on secondary path estimates. This could be a winning approach if the SP is difficult to be estimated or if it varies strongly in time. In all the other situations there is no reason why the SP estimate should not be somehow used to improve both speed of convergence and performance. Starting from an approach similar to FDSA, we use the secondary path estimate to compute in parallel the errors coming from different configurations of the controller. Though, this adds complexity with respect to the stochastic method, but it also increases the speed of convergence. A careful tuning of the algorithm parameters allows to improve both performance, execution time

and sometimes stability. The method is explained in detail in the rest of the chapter.

3.2 Empirical Weight Update (EWU)

In this section we introduce The Empirical Weight Update or EWU. The name comes from the fact that the update of the controller coefficients is done by empirically testing various configurations. This choice comes from the idea of using the estimate of the secondary path to try different controller filters at the same time. In this way we can benefit in speed, even if the introduction of an SP estimate may also introduce some approximation error. For this reason we choose not to compute any estimate of the gradient but to empirically move the controller filter parameters in the coefficients field. We test a number of configurations equal to twice the coefficients number as in FDSA [9], but simultaneously. As in PSO [21] the algorithm discriminates the best direction on the error output, but at each decision step, only one error is a real output measured, while the others are virtually computed just considering their instantaneous values. Theoretically, the method moves each coefficient on its axis and chooses the best direction in which to proceed. Then, all the chosen directions are linearly combined and the controller's weights are updated.

The idea behind EWU (whose block scheme is depicted in figure 3.1) is to discriminate which is the best direction by looking at the instantaneous errors produced by different filters. In order to set up the algorithm, we have input-output delay of the SP. This parameter defines the minimum number of steps necessary to propagate the effect of a single input on y' and consequently the effect of a perturbation on the controller coefficient. Accordingly, EWU has to wait for that number of steps to try simultaneously different configurations thus finding the best instantaneous solution. If, for example, the secondary path includes $y(n)$, EWU can suddenly compute the virtual errors coming out from all the considered filters. Otherwise, if we do not have $y(n)$ but $y(n-k)$ EWU has to wait for k steps in order to measure the first effect of the coefficient changes.

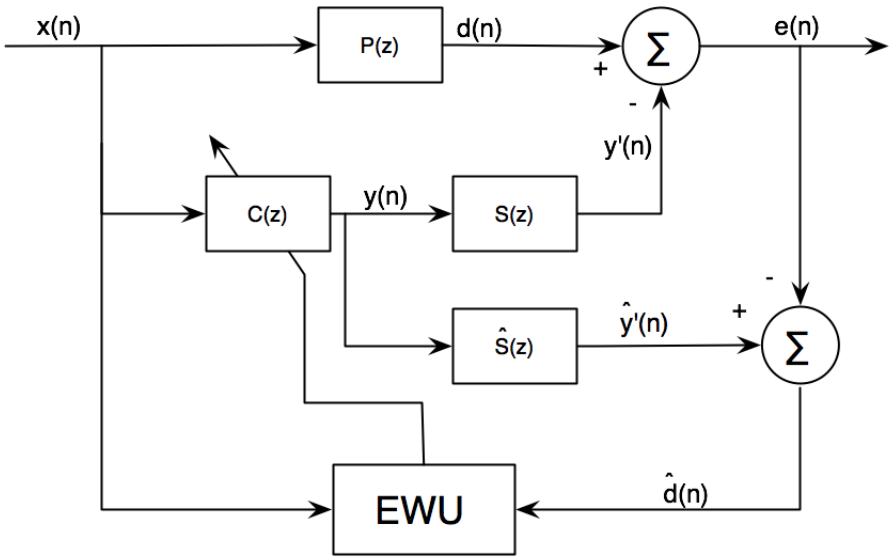

Figure 3.1: EWU block scheme

To introduce the idea, we will explain the update procedure of the algorithm and, to have a concrete example, we will take the two-coefficients case ($W = \begin{bmatrix} w_1 & w_2 \end{bmatrix}^T$) with a secondary path including $y(n)$, without forgetting that EWU can be extended to any number of coefficients. At each iteration the algorithm performs the following tasks:

1. it computes the output $y(n)$ of the controller

$$y(n) = W^T \Phi(n)$$

where Φ and W are respectively the regressor and the coefficient vectors of the controller

2. it collects the error $e_0(n)$ coming from the current configuration

$$e_0(n) = d(n) - y'(n)$$

where $y'(n)$ is the secondary path output and $d(n)$ is the signal which has to be attenuated

3. it computes the output of the secondary path estimate $\widehat{y'}(n)$ without perturbations on the controller coefficients

$$\widehat{y'}(n) = \Theta^T \Gamma(n)$$

where Γ and Θ are respectively the regressor vector and the coefficient vector of the SP estimate

4. for each coefficient it computes the virtual errors corresponding to the two possible directions on its axis, v_{i^+} and v_{i^-}, moving with a step-size μ. Let's define:

$$v_{i^\pm} = n_c \text{ elements zero-vector with a } \pm\mu \text{ in the i-th position}$$

where n_c is the number of coefficients of the controller. Hence, in the case taken into consideration, we have four directions:

$$v_{1^+} = \begin{bmatrix} +\mu & 0 \end{bmatrix} v_{1^-} = \begin{bmatrix} -\mu & 0 \end{bmatrix} v_{2^+} = \begin{bmatrix} 0 & +\mu \end{bmatrix} v_{2^-} = \begin{bmatrix} 0 & -\mu \end{bmatrix}$$

In this way v_{1^+} and v_{1^-} are associated to w_1, while v_{2^+} and v_{2^-} to w_2. The virtual errors are obtained by reconstructing the primary path output d starting from the result of the previous steps:

$$\widehat{d}(n) = e_0(n) - \widehat{y'}(n)$$

$$y_{i^\pm}(n) = (W + v_{i^\pm})^T \Phi(n)$$

$$\widehat{y'}_{i^\pm}(n) = \Theta^T \Gamma_{i^\pm}(n)$$

$$\widehat{e}_{i^\pm}(n) = \widehat{d}(n) - \widehat{y'}_{i^\pm}(n)$$

where $\widehat{e}_{i^\pm}$ are the errors referred to the i-th coefficient with respect to the positive or negative direction and $\Gamma_{i^\pm}$ is the regressors' vector of the SP using $y_{i^\pm}(n)$ instead of $y(n)$

5. for each coefficient it chooses the direction $v_{i^{min}}$ that leads to the minimum error

$$v_{i^{min}} = v_j \text{ where } j \in \{0, i^+, i^-\} \text{ such that } \hat{e}_j = \min\left(\{e_0, \hat{e}_{i^+}, \hat{e}_{i^-}\}\right)$$

$$\hat{e}_{i^{min}} = \min\left(\{e_0, \hat{e}_{i^+}, \hat{e}_{i^-}\}\right)$$

6. it moves the coefficients accordingly to a linear combination of the selected directions scaling them with a factor q_i:

$$W = W + \sum_{i=0}^{n_c} q_i v_{i^{min}}$$

$$q_i = \frac{(e_0^2 - \hat{e}_{i^{min}}^2)}{\sqrt{\sum_{j=1}^{n_c}(e_0^2 - \hat{e}_{j^{min}}^2)^2}}$$

In this way:

$$\sum_{i=1}^{n_c} q_i^2 = 1$$

For example if the selected directions are v_{1^+} and v_{2^+}

$$q_1 = \frac{(e_0^2 - \hat{e}_{1^+}^2)}{\sqrt{\sum_{j=1}^{2}(e_0^2 - \hat{e}_{j^{min}}^2)^2}}$$

$$q_2 = \frac{(e_0^2 - \hat{e}_{2^-}^2)}{\sqrt{\sum_{j=1}^{2}(e_0^2 - \hat{e}_{j^{min}}^2)^2}}$$

$$W = W + q_1 \cdot v_{1^+} + q_2 \cdot v_{2^+}$$

The algorithm ends when perturbations do not lead to smaller errors for a predefined number of steps. EWU tries to implement instantaneously what is described in figure 3.2, 3.3 and 3.4 but without the exact knowledge of the error in every probed direction.

The Empirical Weight Update method reaches convergence earlier than the PSO and SPSA, thanks to the virtualization of the perturbation system. Another good feature of this algorithm is that the update of each single coefficient is not proportional to the magnitude of the error but only to the step-size; this makes EWU particularly resistant to error spikes. However, as is, EWU is still computationally too complex to be compared with NFGLMS. In fact, for each step it needs a number of convolutions proportional to 4*n_c (2*n_c to propagate every single perturbation through the controller and 2*n_c to do the same with the SP estimate).

3.2.1 Complexity Optimizations

The steps of EWU are computationally heavy because of the multiple convolutions involved in the calculations, but we can reduce this load by exploiting the relations between the initial position of the coefficients in the coefficients field

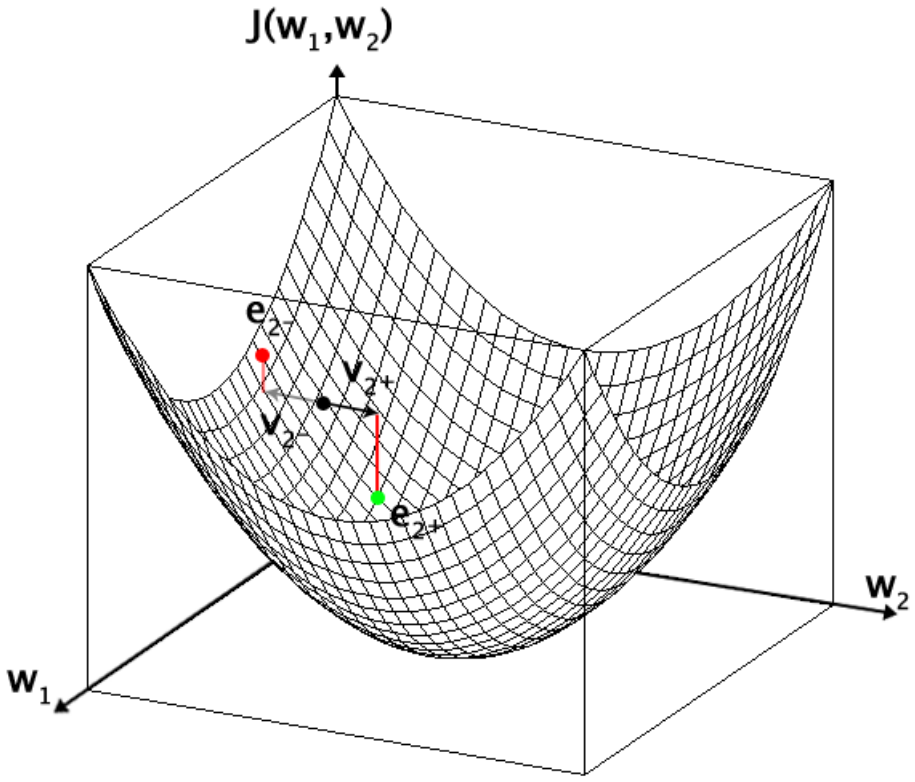

Figure 3.2: perturbations on the coefficients w1. In green the smallest error.

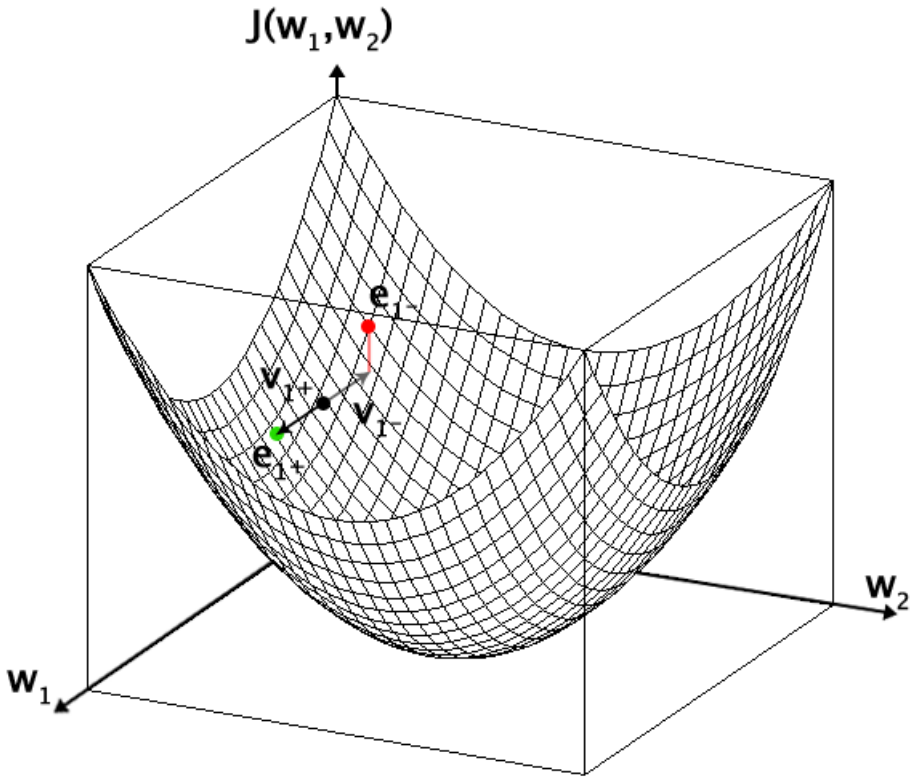

Figure 3.3: perturbations on the coefficients w2. In green the smallest error.

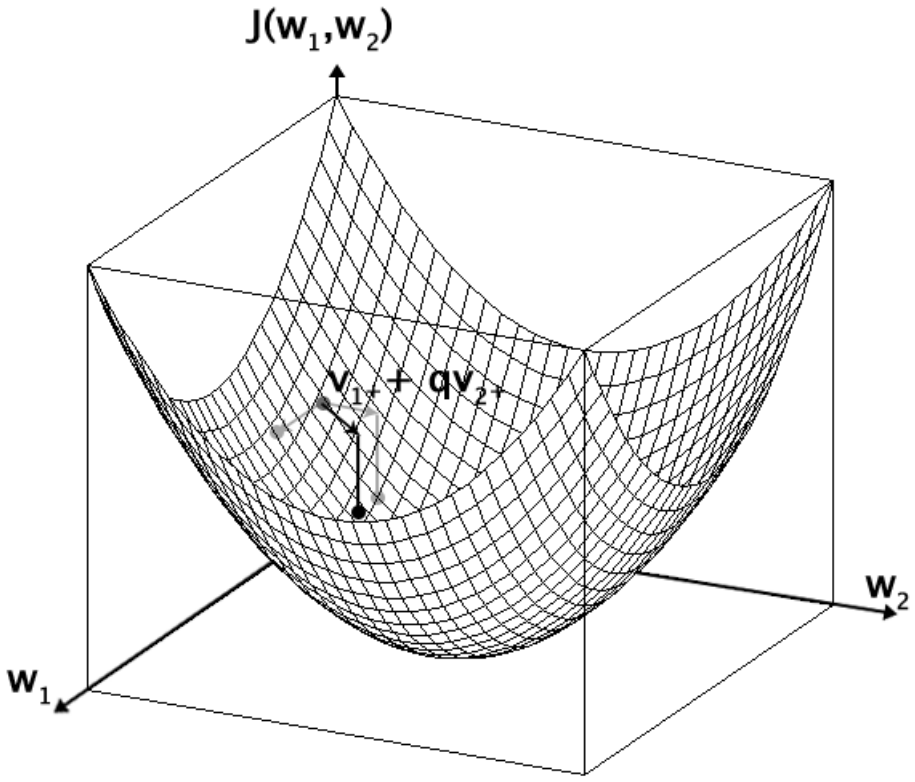

Figure 3.4: linear combinations between the best directions.

Algorithm 1 EWU algorithm

Φ, Γ ▷ controller and SP regressors matrices
$\hat{\Gamma}$ ▷ SP estimate regressor matrix
W, Θ ▷ controller and SP coefficient vectors
$n_c \leftarrow length(W)$
for $i = 1 : n_c$ **do**
  $v_{i^+} \leftarrow zeros(n_c, 1)$;
  $v_{i^-} \leftarrow zeros(n_c, 1)$;
  $v_{i^+}(i) \leftarrow \mu$; ▷ positive direction on the i-th coefficient axes
  $v_{i^-}(i) \leftarrow -\mu$; ▷ negative direction on the i-th coefficient axes
end for
for all n **do**
  $y(n) \leftarrow W \cdot \Phi(n)$ ▷ controller output
  $y'(n) \leftarrow \Theta \cdot \Gamma(n)$ ▷ SP output
  $\hat{y}'(n) \leftarrow \Theta \cdot \hat{\Gamma}(n)$ ▷ SP estimate output
  $e_0(n) \leftarrow d - y'(n)$
  $\hat{d}(n) \leftarrow e_0 + \hat{y}'(n)$ ▷ Primary Path estimate output
  for $i = 1 : n_c$ **do**
    $y_{i^\pm}(n) \leftarrow (W + v_{i^\pm}) \cdot \Phi(n)$ ▷ controller outputs associated to $v_{i^\pm}$
    $\hat{\Gamma}_{i^\pm}(n)$ ▷ SP estimate regressor matrices with $y_{i^\pm}(n)$ in place of $y(n)$
    $\hat{y}'_{i^\pm}(n) \leftarrow \Theta \cdot \hat{\Gamma}_{i^\pm}(n)$ ▷ SP outputs associated to $v_{i^\pm}$
    $\hat{e}_{i^\pm}(n) \leftarrow \hat{d} - \hat{y}'_{i^\pm}(n)$
    $\hat{e}_{i^{min}}(n) \leftarrow \min\left(\{e_0, \hat{e}_{i^+}, \hat{e}_{i^-}\}\right)$
    if $\hat{e}_{i^{min}}(k) = e_0$ **then**
      $v_{i^{min}} \leftarrow zeros(n_c, 1)$
    else if $\hat{e}_{i^{min}}(n) = \hat{e}_{i^+}$ **then**
      $v_{i^{min}} \leftarrow v_{i^+}$
    else if $\hat{e}_{i^{min}}(n) = \hat{e}_{i^-}$ **then**
      $v_{i^{min}} \leftarrow v_{i^-}$
    end if
    $D_i \leftarrow e_0^2 - \hat{e}_{i^{min}}^2$ ▷ Improvement associated to the smallest error
  end for
  for $i = 1 : n_c$ **do**
    $q_i \leftarrow \frac{D_i}{\sqrt{\text{sum}\ (D^2)}}$
    $W \leftarrow W + q_i \cdot v_{i^{min}}$
  end for
end for

and the other possible positions. This is what happens to $y(n)$ if we perturb a single coefficient i on a controller with n parameters:

$$y_i(n) = (W + \mu\Delta_i)\Phi(n) = W\Phi(n) + \mu\Delta_i\Phi(n) = y(n) + \mu\Delta_i\Phi(n)$$

where W is the coefficient vector, Φ is the regressors vector and Δ_i denotes a vector of n_c elements with a one in the i-th place and zeros elsewhere. For this reason we can write:

$$y_i(n) = y(n) + \mu\delta_i\phi_i \tag{3.1}$$

where δ_i and ϕ_i are the i-th component of Δ_i and Φ_i. Now, looking at the SP output we can say that:

$$y'(n) = \Theta\Gamma(n) \tag{3.2}$$

where Θ is the coefficient vector of the SP and Γ is the regressors vector. Moreover

$$n - m = k$$

where m is the smallest delay on y in the secondary path. This means that a perturbation of the coefficients taking place at k will show its effect after m steps. Now, let's define:

$$\begin{aligned}
\Omega_0 &= \{\text{all the regressors in } \Gamma \text{ not containing } y(k)\} \\
\Omega_1 &= \left\{ \frac{\text{all the regressors in } \Gamma \text{ containing } y^1(k)}{y^1(k)} \right\} \\
\Omega_2 &= \left\{ \frac{\text{all the regressors in } \Gamma \text{ containing } y^2(k)}{y^2(k)} \right\} \\
&\vdots \\
\Omega_j &= \left\{ \frac{\text{all the regressors in } \Gamma \text{ containing } y^j(k)}{y^j(k)} \right\}
\end{aligned}$$

and:

$$\begin{aligned}
\Sigma_0 &= \sum \Theta_{\Omega_0}\Omega_0 \\
\Sigma_1 &= \sum \Theta_{\Omega_1}\Omega_1 \\
\Sigma_2 &= \sum \Theta_{\Omega_2}\Omega_2 \\
&\vdots \\
\Sigma_j &= \sum \Theta_{\Omega_j}\Omega_j
\end{aligned}$$

Where all the Θ_{Ω_j} are the sets of the coefficients associated with each Ω_j. Hence, $y'(k)$ can be expressed as:

$$y'(n) = \Sigma_0 + \Sigma_1 y(k) + \Sigma_2 y^2(k) + \cdots + \Sigma_n y^n(k) \tag{3.3}$$

The y'_i associated to a perturbation i is, remembering 3.1:

$$y'_i(n) = \Sigma_0 + \Sigma_1(y(k) + \mu\delta_i\phi_i) + \Sigma_2(y(k) + \mu\delta_i\phi_i)^2 + \cdots + \Sigma_n(y(k) + \mu\delta_i\phi_i)^n \quad (3.4)$$

If we add and remove $\Sigma_1 y(k)$, $\Sigma_2 y^2(k)$, $\cdots$ and $\Sigma_n y^n(k)$ from equation 3.4 the result will be as follows:

$$\begin{aligned} y'_i(n) = \Sigma_0 + \Sigma_1((y(k) + \mu\delta_i\phi_i) + y(k) - y(k)) + \\ + \Sigma_2((y(k) + \mu\delta_i\phi_i)^2 + y^2(k) - y^2(k)) + \\ + \cdots + \\ + \Sigma_n((y(k) + \mu\delta_i\phi_i)^n + y^n(k) - y^n(k)) \end{aligned} \quad (3.5)$$

In this formula we can recognize equation 3.3, so:

$$\begin{aligned} y'_i(n) = y'(n) + \Sigma_1((y(k) + \mu\delta_i\phi_i) - y(k)) + \\ + \Sigma_2((y(k) + \mu\delta_i\phi_i)^2 - y^2(k)) + \\ + \cdots + \\ + \Sigma_n((y(k) + \mu\delta_i\phi_i)^n - y^n(k)) \end{aligned} \quad (3.6)$$

For example if we have a SP designed as follows:

$$y'(n) = ay'(k-2) + by'(k-1)y^2(k) + cy(k-1)y^2(k) + dy'(k-1)y(k-1) + ey(k)$$

we can define:

$$\Omega_0 = \{y'(k-2),\, y'(k-1)y(k-1)\}$$
$$\Omega_1 = \{1\}$$
$$\Omega_2 = \{y'(k-1),\, y(k-1)\}$$

and

$$\Sigma_0 = ay'(k-2) + dy'(k-1)y(k-1)$$
$$\Sigma_1 = e$$
$$\Sigma_2 = by'(k-1) + cy(k-1)$$

From equation 3.6:

$$\begin{aligned} y'_i(n) = y'(n) + e((y(k) + \mu\delta_i\phi_i) - y(k)) + \\ + (by'(k-1) + cy(k-1))((y(k) + \mu\delta_i\phi_i)^2 - y^2(k)) \end{aligned}$$

Exploiting this relationship is particularly convenient with a linear SP. In this case, we can compute y' as a convolution of y with the impulse response

of the SP. If the SP is nonlinear this cannot be done, but if we take a look at equation 3.4, it can be seen that Σ_0 is the same of that in equation 3.3; this means that we have to compute it only once for all the perturbations, thus achieving a considerable saving even in this case. Hence, we have only to compute the terms containing $y(k)$. For example if we have a SP designed as follows:

$$y'(n) = ay'(n-2)+by'(n-1)y^2(k)+cy(n-1)y^2(k)+dy'(n-1)y(n-1)+ey(k)$$

we can define:

$$\Omega_0 = \{y'(n-2),\, y'(n-1)y(n-1)\}$$
$$\Omega_1 = \{1\}$$
$$\Omega_2 = \{y'(n-1),\, y(n-1)\}$$

and

$$\Sigma_0 = ay'(n-2) + dy'(n-1)y(n-1)$$
$$\Sigma_1 = e$$
$$\Sigma_2 = by'(n-1) + cy(n-1)$$

$$y_i'(n) = \Sigma_0 + e(y(k) + \mu\delta_i\phi_i)+$$
$$+(by'(n-1) + cy(n-1))(y(k) + \mu\delta_i\phi_i)^2$$

This approach to the problem needs a preliminary stage where a data structure is created to handle all the matrices taking part in the algorithm, but it considerably reduces the number of multiplications and additions involved in each step. As a matter of fact we will see that, thanks to this optimization, EWU can compete with NFGLMS in terms of computational time for each iteration.

3.2.2 Step size update policy

The performance of EWU is limited if we choose a fixed gain. This the latter avoids the achievement of convergence of the controller coefficients that keep on fluctuating around the optimum value. Accordingly the algorithm can be modified by introducing a suitable adaption of the step size, with the possibility to choose between two different policies. A first policy assumes a single step-size for all the involved coefficients, and reduces (e.g. by half) it when all the probed directions do not lead to a lower error. Otherwise,

a vector of independent gains can be built, each one associated to a specific coefficient. In this case the step size reduction is operated only when the two directions associated to a coefficient do not lower the outcome. This last approach gives the algorithm a more granular control over the parameters, consequently improving the accuracy. Moreover, the additional computational burden is negligible if compared to the additional degrees of freedom that are gained. Each iteration of the algorithm operates as follows:

1. it computes the output $y(n)$ of the controller
2. it collects the error $e_0(n)$ coming from the current configuration
3. it computes the output of the secondary path estimate $\widehat{y'}(n)$ without perturbations on the controller coefficients
4. for each coefficient i it computes the virtual errors corresponding to the two possible directions on its axis, moving with a step-size μ_i.

   $$v_{i^\pm} = n_c \text{ elements zero-vector with a } \pm\mu_i \text{ in the i-th position}$$

5. for each coefficient it chooses the direction that leads to the minimum error. If no perturbation leads to an improvement, it reduces the step size correspondent to the coefficient (e.g. by having it):

   $$\text{if } e_0 = \min\left(\{e_0, \hat{e}_{i^+}, \hat{e}_{i^-}\}\right) \text{ then } \mu_i = \frac{\mu_i}{2}$$

6. it moves the coefficients accordingly to a linear combination of the two optimal orthogonal directions.

in this variant the algorithm ends when all the elements of the step-size vector are close to zero. In the following we will consider the EWU version with the halving of the step size.

3.3 EWU with uniform linear combination (UEWU)

An element that could lead to an increase in the complexity of each iteration of EWU is the computation of the linear combination coefficients. In the previous

Table 3.1: Table representing execution times for each one of the shown tests

repetition	Uniform combination time	Weighted combination time
1	0.458443 seconds	0.490062 seconds
2	0.445222 seconds	0.495592 seconds
3	0.453671 seconds	0.508008 seconds

described versions of the algorithm the calculations include divisions. As we know, division is a complex operation (computationally speaking) and here is repeated $n_c - 1$ times, where n_c is the number of coefficients of the controller. In order to speed up the algorithm, we can assume that all the coefficients of the linear combination are equal to one, instead of obtaining them from the divisions explained above:

$$q_i = 1 \forall i$$

Obviously this choice ends up to be less accurate, as we can see in figure 3.5, 3.6 and 3.7. These three graphs represent the MSE (in dB) of the system with the two discussed types of combination. They are obtained by feeding the systems with the same three input sets and running the algorithm for 1000 iterations. Notice that the uniform linear combination EWU (from now on UEWU) does not reach the performance of the standard algorithm. Looking at the execution time (table 3.1) we can see that there is an actual time reduction earned by avoiding the computation of the combination coefficients. However this is negligible with respect to the loss in performance.

We can better understand the gap between the two methods by looking at figures 3.8 and 3.9. These represent the update path of one coefficient in the first 115 steps for both the methods. It is clear that UEWU leads to a rougher updating path that delays the convergence. A weighted combination is also important because it protects the algorithm from approximation errors. In fact the smaller is the advantage in taking one direction, the smaller will it be weighted in the linear combination.

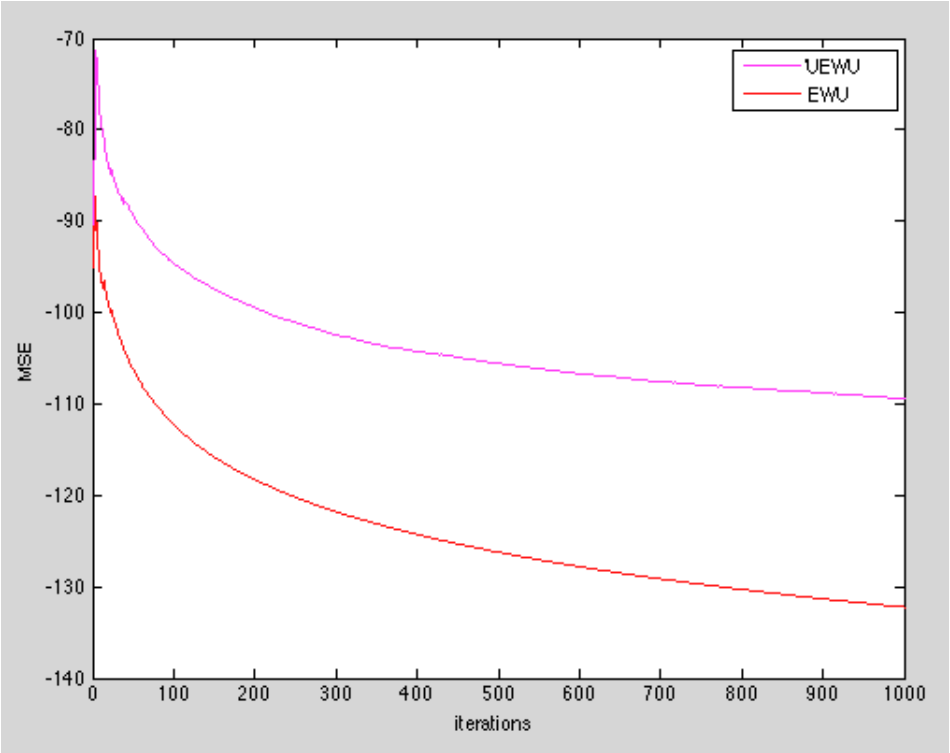

Figure 3.5: EWU MSE(red) and UEWU (violet) 1/3

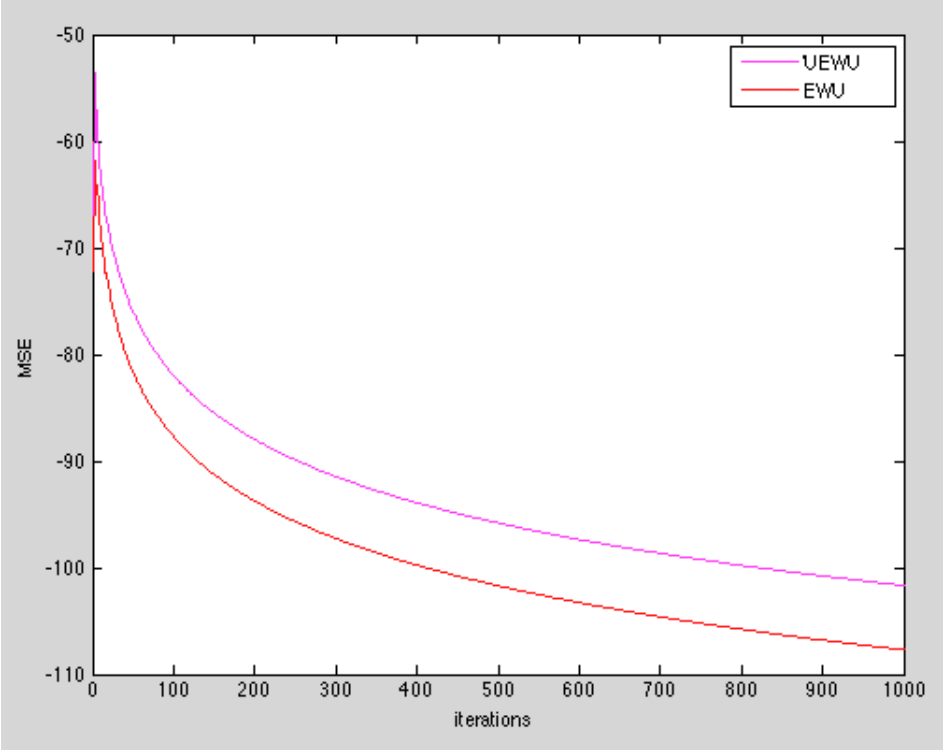

Figure 3.6: EWU MSE(red) and UEWU (violet) 2/3

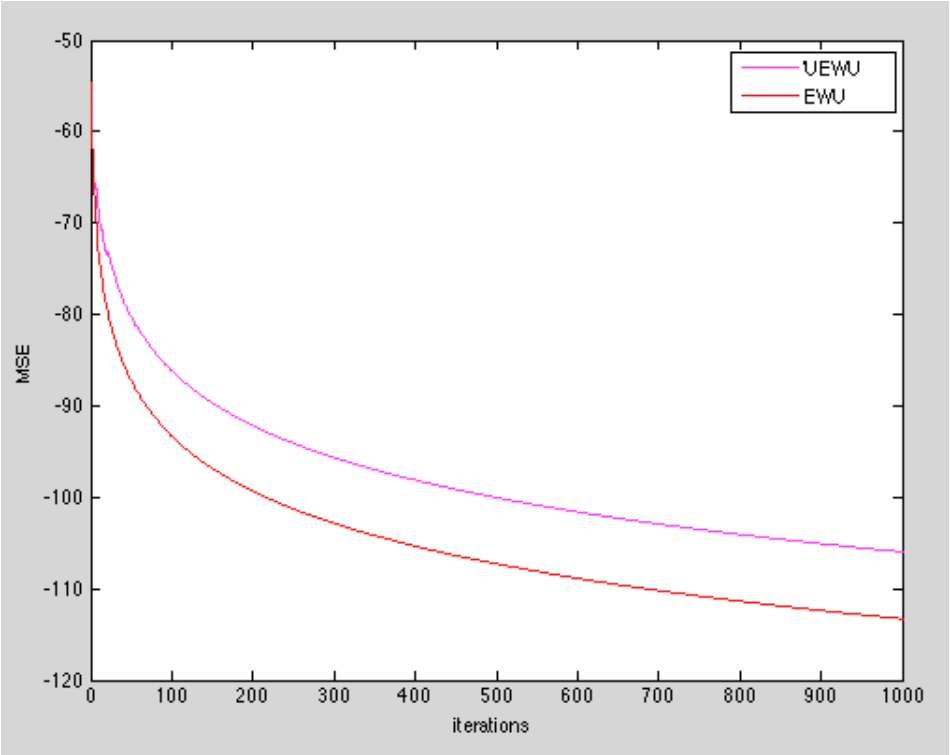

Figure 3.7: EWU MSE(red) and UEWU (violet) 3/3

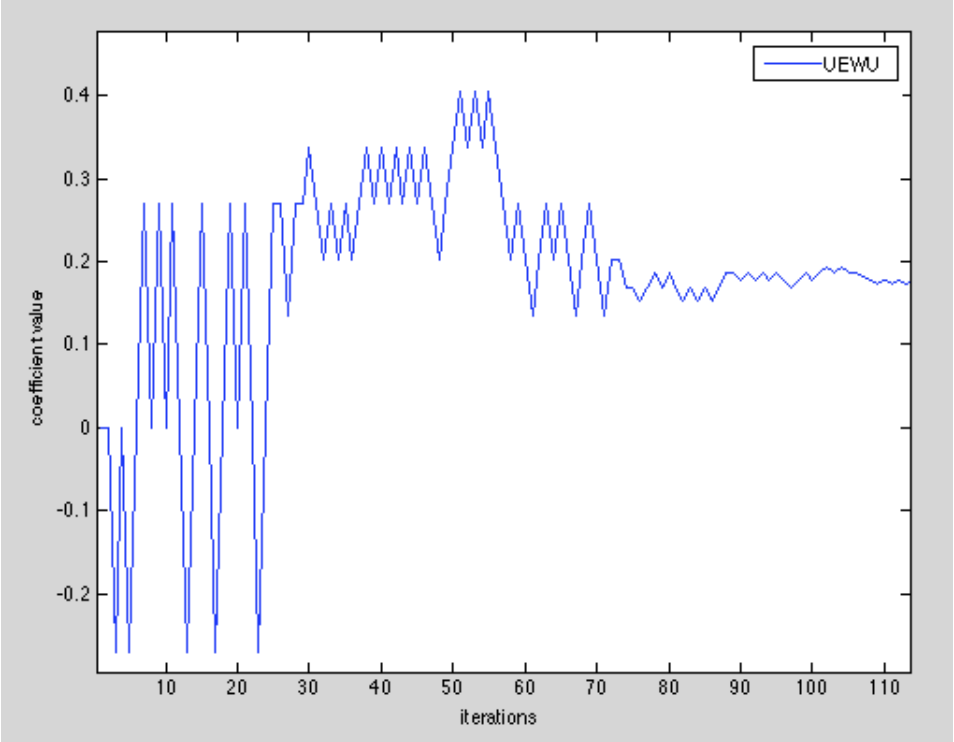

Figure 3.8: coefficient update path using EWU with uniform linear combination

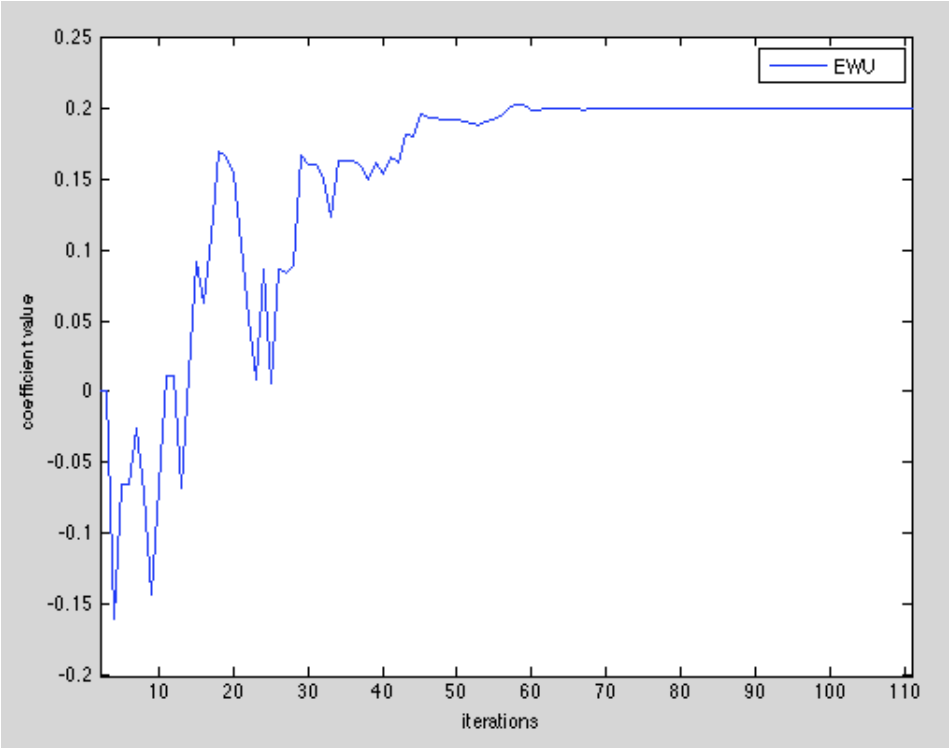

Figure 3.9: coefficient update path using EWU with weighted linear combination

3.4 Considerations about the instantaneous error

Even if EWU reaches good attenuation results the instantaneous error is not an efficient discriminant because, as already observed for stochastic algorithms, the computation of the error at each iteration and for each direction, is affected by the previous outputs. In fact, if we look at figure 3.8, 3.9 and 3.10 we can notice that NFGLMS follows a well defined direction in the update of the coefficients, while the EWU and UEWU coefficients paths are more chaotic. In some cases the impact of this approximation is so strong that UEWU can not even converge to the correct parameters values. This is the case of figure 3.11, where the coefficient is not able to reach the convergence value of 0.2. This is caused by subsequent wrong evaluations of the virtual errors. These mislead the algorithm that begins halving the steps size, reducing it to zero before the coefficients reach the convergence value.

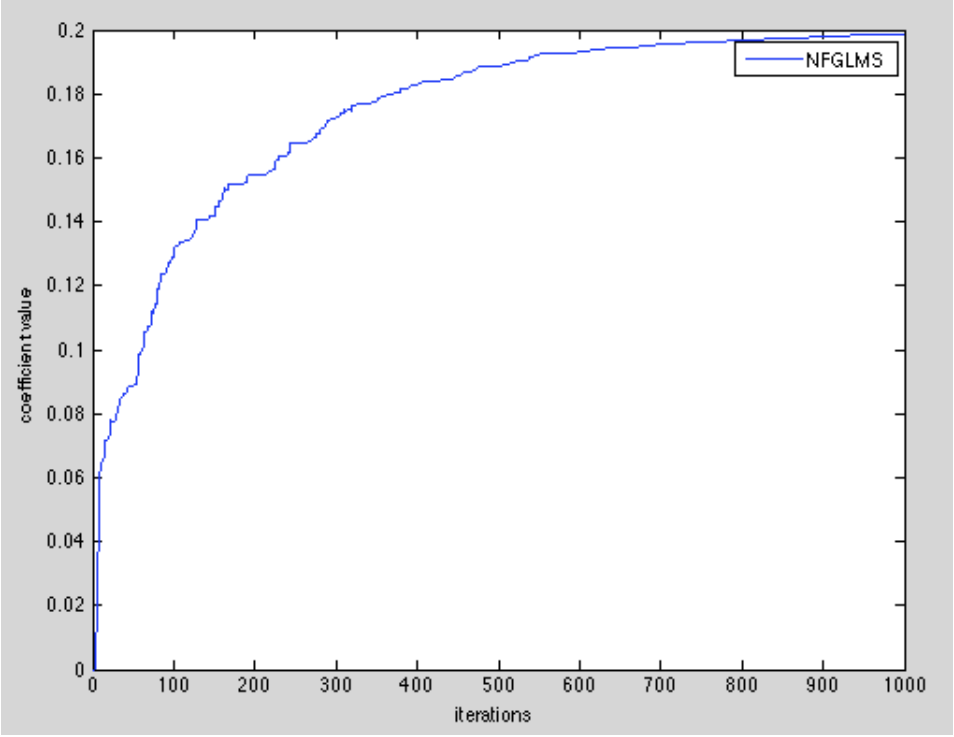

Figure 3.10: NFGLMS coefficient update-path

The most natural solution would be to compute, in parallel, the errors of all the considered directions, for a number of steps at least equal to the memory

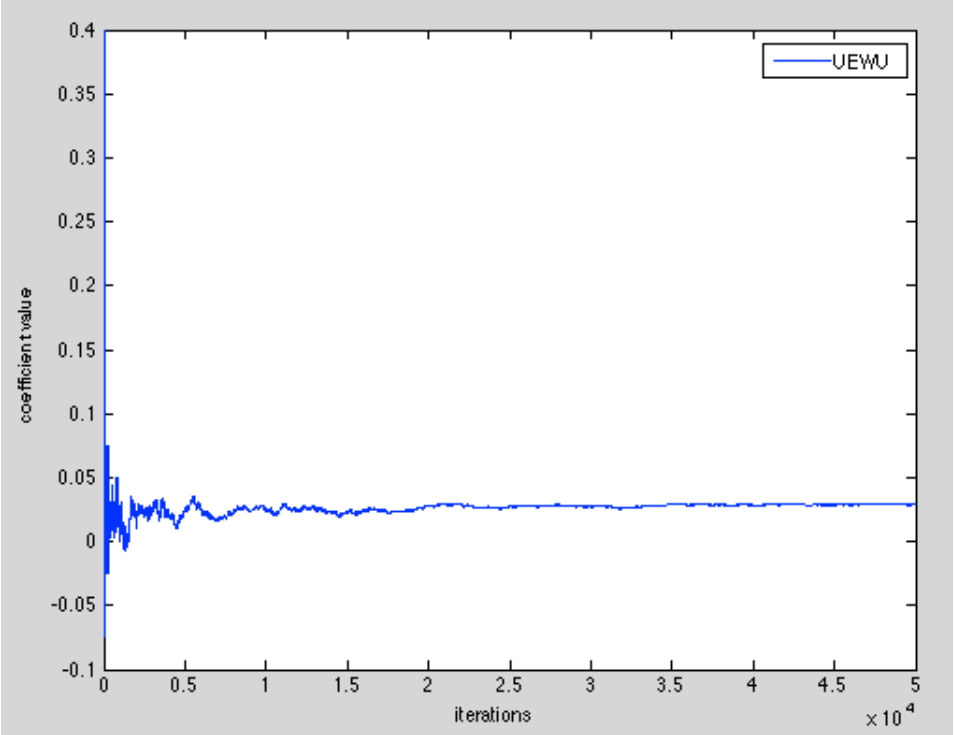

Figure 3.11: UEWU coefficient update-path. The convergence value should be 0.2 .

of the secondary path. This solution is impractical because it does not scale properly as the complexity of the model increases, which is one of the main targets of EWU.

3.5 Convergence solutions

As previously shown, UEWU sometimes does not reach convergence or has a consistent bias in the parameters. This is caused by the use of the instantaneous error as a discriminant to find the best directions that, in some cases, can lead us to a wrong evaluation or can stop the coefficients evolution before proper convergence. In the following sub-sections we will list some possible solutions to this problem. These try to help the algorithm to reach convergence without affecting too much the general complexity.

3.5.1 EWU step size doubling

An element which we can act upon is the step size. Until now EWU halves the step size when it does not find, in the contour of the current starting coefficient, a better position. This, together with the problem analyzed previously regarding the instantaneous error, could take rapidly the step size to zero even if the best coefficient is not found, so an alternative rule was introduced. If a given direction is chosen more than n times consecutively the corresponding step size is doubled (EWU-SSD). To prevent the algorithm from divergence a clipping threshold for the step size is set up. The additional complexity for this solution is almost negligible. EWU-SSD behaves as follows:

1. it computes the output $y(n)$ of the controller
2. it collects the error $e_0(n)$ coming from the current configuration
3. it computes the output of the secondary path estimate $\widehat{y'}(n)$ without perturbations on the controller coefficients
4. for each coefficient it computes the virtual errors corresponding to the two possible directions on its axis, moving with a predefined step-size μ
5. for each coefficient it chooses the direction that leads to the minimum error. If no perturbation leads to an improvement, it halves the step size correspondent to the coefficient. If the positive (or negative) direction is selected consecutively more than a predefined number n of times it doubles the correspondent step size.
6. it moves the coefficients accordingly to a linear combination of the optimal orthogonal directions.
7. it repeats the previous step until the error is under a chosen threshold.

EWU-SSD adds an important feature to EWU that rewards the trend of a particular coefficient to go in a specific direction and avoids the algorithm to stop in non optimal configurations. The coefficient path shown in figure 3.11, for example, becomes, with this improvement, the one in figure 3.12. Comparing the two pictures we can see clearly that the step size doubling helps

the algorithm to move from a stalemate caused by the decrease in amplitude of the step size. The two figures refer to tests done using UEWU, where the problem is more evident. In fact even if the step size doubling improves the performance of EWU too, we can not see clearly from the figures where it begins to work because the step size is weighted by the linear combination coefficients.

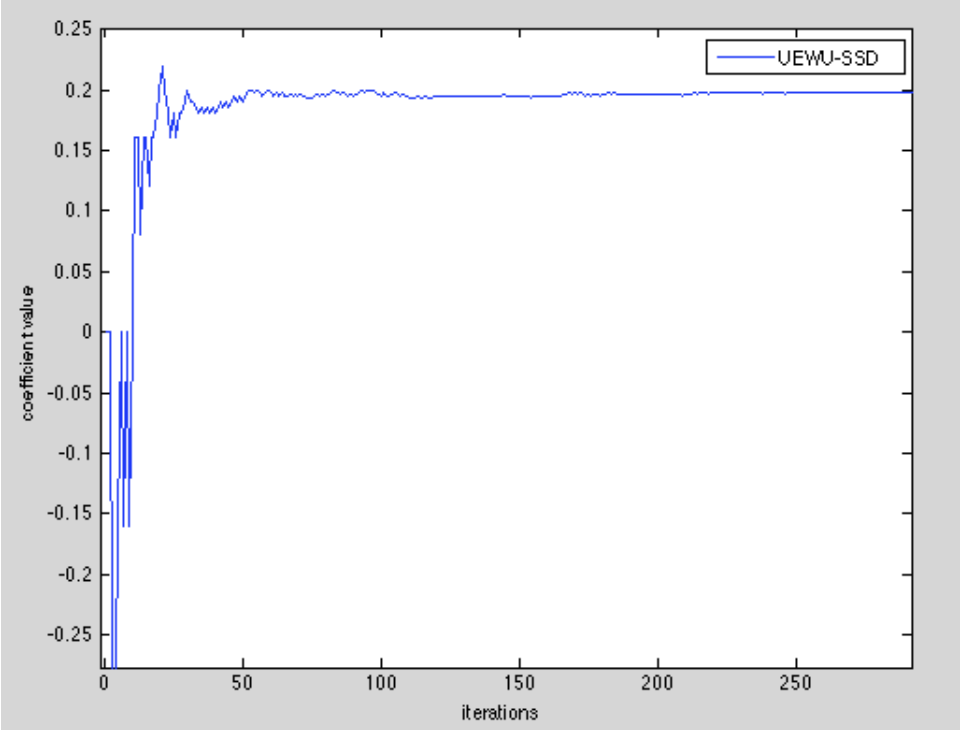

Figure 3.12: UEWU-SSD update-path of one coefficient. The convergence value would be 0.2 .

3.5.2 2-stages EWU

We said that the instantaneous error is not a good discriminator because it is affected by the previous steps, which could be wrong. The maximum between the settling time of the secondary path and the settling time of the controller M_{max} determines the number of steps that influence the error at each instant. In order to obtain a clean error we have to wait at least M_{max} steps after every single perturbation. In order to use this clean error as a discriminant one has to keep in memory, at each of these M_{max} steps, the y_i and y'_i for

each simple direction, performing a number of convolutions equal to twice the number of coefficients. Computationally speaking, this is quite heavy, so it is better to use this concept only when it is strictly necessary. At this purpose we can set up a 2-stages algorithm (2S-EWU). The first stage is a simple EWU but when the step sizes of all the coefficients are under a predefined threshold the updating process stops and the algorithm enters in a new stage. Here it uses the knowledge of the previous M_{max} steps collecting, for each iteration, the virtual error of every single direction, on every coefficient axis, in parallel, without affecting the real output of the system. At the end of these M_{max} steps, the MSEs relative to each direction are compared with the one relative to the real output (where all the coefficient are maintained unaltered) and the coefficients are updated consequently as in the standard EWU. The linear combination coefficients q_i become:

$$MSE_i = \frac{1}{M_{max}} \sum_{j=n-M_{max}}^{n} e_i^2(j)$$

$$q_i = \frac{(MSE_0^2 - \hat{MSE}_{i^{min}}^2)}{\sqrt{\sum_{j=1}^{n_c}(MSE_0^2 - \hat{MSE}_{j^{min}}^2)^2}}$$

Moreover, if some direction is chosen, its correspondent step-size is scaled up by a predefined factor. The M_{max} steps are not effective steps because they are computed in parallel to the system, hence while the latter continues to run with the last parametrization. The algorithm works as follows:

1. it computes the output $y(n)$ of the controller and send it out
2. it collects the error $e_0(n)$ coming from the current configuration
3. it computes the output of the secondary path estimate $\widehat{y'}(n)$ without perturbations on the controller coefficients
4. for each coefficient i it computes the virtual errors corresponding to the two possible directions on its axis, moving with a step-size μ_i.
5. for each coefficient it chooses the direction that leads to the minimum error. If the position without perturbation is chosen it halves the step size correspondent to the coefficient

6. it moves the coefficients accordingly to a linear combination of the optimal orthogonal directions

7. it repeats the previous step until all the step sizes are over a predefined threshold, otherwise it goes to the following step

8. it uses the last M_{max} samples of d_{hat} to simulate the effect of the perturbations in each direction computing the correspondent MSE_i. Meanwhile the system continues to run with the last configuration

9. if one direction leads to an improvement it updates the correspondent coefficient and the correspondent step-size

$$\mu_i = \mu_i * c$$

where c is a predefined value.

10. it goes to 7 until the error is over a predefined threshold

This algorithm improves performances of both EWU and UEWU but, as in the previous case it is more evident in the latter. Figure 3.13 shows the UEWU and 2S-UEWU MSE. We can see how the 2-stages algorithm follows the standard version until a certain point. Then, when the step-sizes goes to zero, it reacts and the second stage verifies the convergence. In figure 3.14 we can see the path of the coefficient involved in the process which has to be compared with the one in figure 3.11.

This variant of EWU is particularly efficient if the system has a non-minimum phase SP. In fact, in the latter case, it is more convenient to evaluate every single direction for more than one step, because it can happen that the instantaneous effect of a perturbation is not a good index of its rightness. Hence, we can set the threshold equal to the initial step-size and run EWU always

3.5.3 2S-EWU-SSD

The last two optimizations are not mutually incompatible because they work on different parts of the algorithm. Hence we can arrange a new algorithm

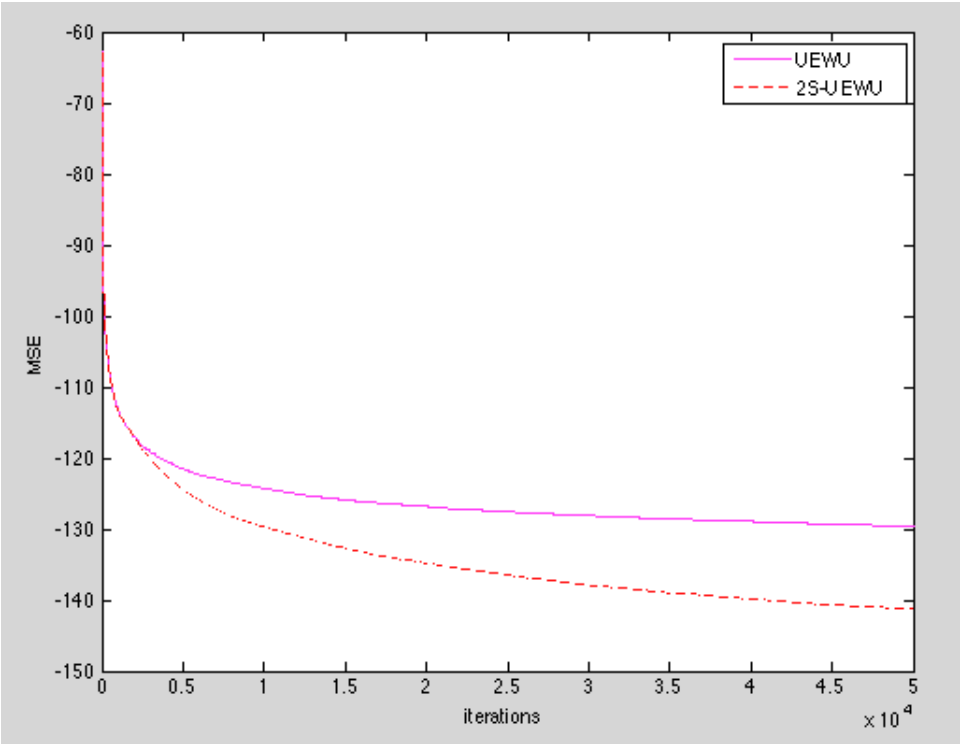

Figure 3.13: 2S-UEWU and UEWU MSE

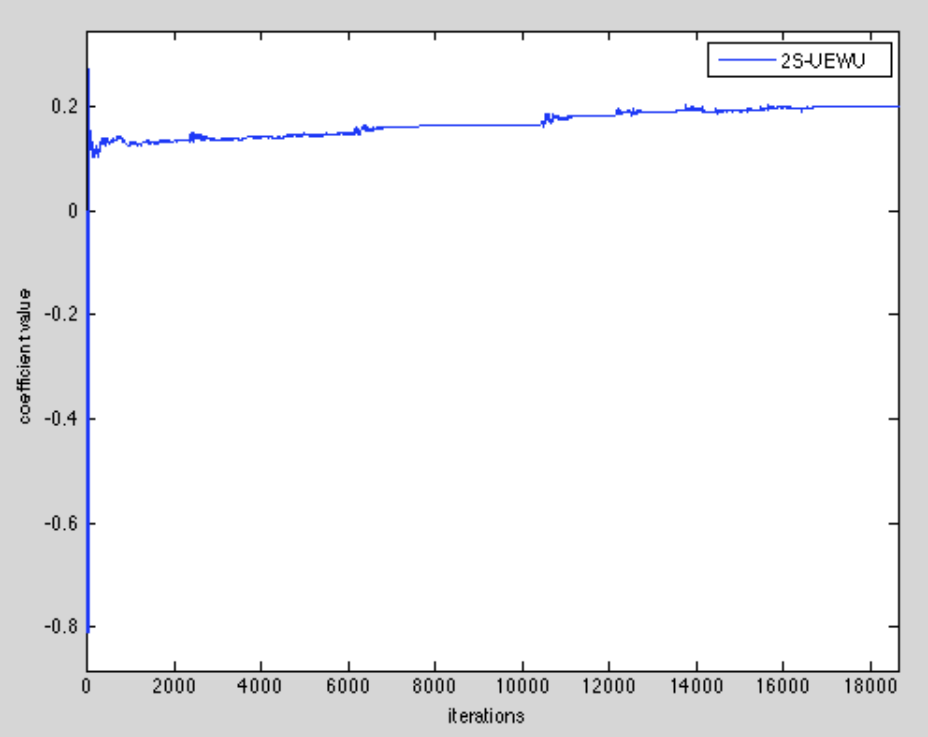

Figure 3.14: 2S-UEWU update-path of one coefficient. It reaches convergence at 0.2.

which takes all the advantages of the two. We call it 2S-EWU-SSD. Its steps are the following:

1. it computes the output $y(n)$ of the controller and send it out
2. it collects the error $e_0(n)$ coming from the current configuration
3. it computes the output of the secondary path estimate $\widehat{y'}(n)$ without perturbations on the controller coefficients
4. for each coefficient it computes the virtual errors corresponding to the two possible directions on its axis, moving with a predefined step-size μ
5. for each coefficient it chooses the direction that leads to the minimum error. If the position without perturbation is chosen, it halves the step size correspondent to the coefficient. If the positive (or negative) direction is selected consecutively more than a predefined number of times n it doubles the correspondent step size.
6. it repeats the previous step until all the step sizes are over a predefined threshold, otherwise it goes to the following step
7. it uses the last M_{max} samples of d_{hat} to simulate the effect of the perturbations in each direction computing the correspondent MSE_i. Meanwhile the system continues to run with the last configuration
8. if one direction leads to an improvement it updates the correspondent coefficient and the correspondent step-size where c is a predefined value.
9. it goes to 7 until the error is over a predefined threshold

Chapter 4

Model Selection

4.1 Introduction

As has been pointed out in Chapter 2 the choice of the model structure is critical to describe accurately the nonlinearities that are present in the system. In particular, high order filters have to be used for this purpose, but they require a high computational complexity. Moreover, filters like Volterra and Bilinear often include redundant regressors. For all these reasons the problem of model structure identification is critical when we deal with nonlinear systems.

There are many ways to describe the nonlinearities, as wavelets, neural networks and fuzzy models [17], but the most common is the polynomial expansion that leads to a model which is linear in the parameters. This feature allows to use algorithms available for linear systems also for nonlinear systems. The application of these techniques introduces problems due to the number of terms included in the model and to the reliability of the estimate parameters [20]. In fact they usually cannot describe the real behavior of the system. From these considerations comes the need to accurately detect the model structure that describes nonlinearities with the minimum number of regressors. Unfortunately the problem of model selection has to deal with several implications. Traditional techniques select terms trying to fit as much as possible the identification data (this is called predictive approach). However, when dealing with nonlinearities there is no relationship between the predictive model and the simulated one [28]. Typical solutions to this problems rely on cross-validation

or on a-priori information about the system, even though they are not always available. These are some of the reasons that make the problem of model selection still an open research issue.

4.1.1 NARX model identification algorithms

Let's from now on consider a particular family of nonlinear models called Nonlinear AutoRegressive models with eXogenous inputs (NARX)(section 2.3.2). As we have already explained the number of terms of a NARX model grows rapidly as the order and the degree of the nonlinearity grows. Trying to estimate the parameters of the complete structure is, hence, not possible if the model is too complex. In fact, the high number of terms leads to a non-reliable parameter estimation. Several studies have shown that models containing a limited number of regressors can describe accurately nonlinearities provided that these are the most significant [28]. So, what is required to the selection algorithm is a way to evaluate the importance of a regressor in a particular model and and a way to compute the associated parameter.

There are many methods that combine these two needs in a single optimization problem. One of those is the Forward-Regression Orthogonal Estimator (FROE) [2]. The FROE increments the model structure at each step until it reaches a specified accuracy. Here the parameters estimation is done through orthogonal least squares (OLS) while the evaluation of a proposed model is left to the error reduction ratio criterion (ERR). The orthogonalization process separates the computation of additional parameters from the ones already present in the model, so each candidate regressor can be evaluated singularly with:

$$[ERR]_i = \frac{\hat{g}_i^2 \sum_{n=1}^{N} w_i^2(n)}{\sum_{n=1}^{N} y^2(n)}$$

where w_i is the i-th auxiliary orthogonal regressor, $\hat{g}_i$ the corresponding parameter [2] and $y(n)$ is the output of the system at step n. $[ERR]_i$ is nothing but an index of similarity between the output and the single regressor weighted with its parameter. Hence, the candidate regressor associated to the highest $[ERR]_i$ is added to the model. Once the regressor is found, a new parameters estimation is done for the non-orthogonal model.

One of the main problem of this approach lies in the error reduction ration [20]. In fact the value associated to one specific regressor varies depending on the order in which the regressor itself is considered giving it more or less importance in the model. This dependency makes the FROE to fail in finding the best subset of regressors within a given model family. A solution to this problem can be found in [20]. The Simulation Error Minimization with Pruning algorithm (SEMP) adds a new mechanism in the selection process called *pruning* and a new regressor selection criterion. Pruning is an iteration step coming immediately after the addition of a new regressor. This phase selects the worst regressor, that is the one that, if removed, will determine the minimum increment of the squared simulation error. If the increased square simulation error is still better than the one referring to the previous iteration the regressor is eliminated from the model and the *pruning* phase is repeated to verify if it is possible to remove other terms. The idea behind *pruning* is that given two models with the same performance, the smaller is to be preferred. On the other hand the Simulation Reduction Error criterion denotes the decrease in MSSE (mean square simulation error) obtained by the inclusion of a regressor in the model normalized with the output variance:

$$[SRE]_j = \frac{MSSE(M_i) - MSSE(M_{i+1})}{\frac{1}{2}\sum_{n=1}^{N} y^2(n)}$$

where M_i is the model at the i-th iteration and M_{i+1} is the model at the subsequent iteration, with the inclusion of the j-th regressor. Thanks to this two new features the SEMP can extract the optimal model for the system with respect to the simulation data [20]. In fact, pruning assures the removal of useless regressors, while the SRE guarantees the correct selection of the regressors. The main drawback of this approach is its computational complexity. In fact, while in linear system SEMP can filter the input in the frequency domain, in the nonlinear case (that is the one taken into consideration) the cycle computation must be explicit for each new candidate regressor and for each regressor in the pruning phase.

A less accurate but faster selection algorithm is represented by the Fast Recursive Algorithm (FRA) [14]. Unlike OLS this does not require matrix decomposition. FRA uses a particular formula to both select new terms and to compute the parameters of the model. This is the net contribution δE_{t+1}

of each term to the cost function:

$$\delta E_{t+1} = \frac{\boldsymbol{y}^T \boldsymbol{\phi}_{t+1} - \sum_{j=1}^{k} (a_{j,y} a_{j,t+1} / a_{j,j})^2}{(\boldsymbol{\phi}_{t+1})^T \boldsymbol{\phi}_{t+1} \sum_{j=1}^{t} (a_{j,t+1}^2 / a_{j,j})}$$

with $t = 0, 1, \cdots, n_t + 1$ and n_t equal to the total number of candidate model terms. $\boldsymbol{\phi}_i = [\phi_i(x(1)), \cdots, \phi_i(x(N))]$ are the regressors evaluated for each input, $\boldsymbol{y} = [y(1), \cdots, y(N)]$, N is the length of the dataset and the terms in the summations are defined as:

$$a_{t,i} = (\boldsymbol{\phi}_k^{(t-1)})^T \phi_i^{(t-1)}$$

$$a_{t,y} = (\boldsymbol{\phi}_k^{(t-1)})^T \boldsymbol{y}$$

with $i = t, ..., n$ and $t = 1, 2, ..., n_t$. The calculation of the net contribution of each candidate regressor relies on the summation of the contributions of each selected regressor in relation to the current candidate. All the computed δE_{t+1} are added to the updated cost function, starting from the squared error:

$$E_0 = \boldsymbol{y}^T \boldsymbol{y}$$

Once the best model is selected, its parameters are computed as:

$$\hat{\theta}_j = \frac{a_{j,t} - \sum_{j+1} t\hat{\theta}_i a_{j,i}}{a_{j,j}}$$

with $j = t, t-1, ..., 1$.

All the methods seen so far, in the presence of the SP, foresee a preliminary step to retrieve the controller output. In fact, when the secondary path is assumed, only the input of the system and its output are known. Since, at the beginning of the selection, the controller model is unknown, it is only possible to derive y from y'. When the SP is known and linear, this process consists in the inversion of the filter. Obviously, to be inverted, the SP filter must be minimum-phase, but there are also other techniques used to approximate $S^{-1}(z)$. When the filter is unknown, an adaptive algorithm based on the LMS method, is used to provide the inverse.

4.1.2 Conclusions

Model selection is an important practice to lower computational complexity reducing the number of regressors. All the presented approaches to the problem

are intended to be used offline because they need a dataset containing all the inputs and outputs in a specific period of time. They are able to achieve a good (in some cases optimal) selection but cannot deal with a changing in the system that could ask for a controller model update. Moreover in presence of a Nonlinear SP (NSP) the controller output estimation could be difficult to achieve. In the next section we will propose a new method of model selection that can get over these problems.

4.2 EWU selection

EWU selection is a pseudo-online model selection method. The term *pseudo* indicates that the selection of the model is not performed affecting the real output of the system but virtually. This was done not to degrade the performance of the attenuating algorithm and to always use a tested configuration. The overall method is made possible by deriving the primary path output $d(n)$ from the output error:

$$\hat{d}(n) = \hat{y}'(n) + e(n)$$

with $\hat{y}'(n)$ representing the SP estimate output and $e(n)$ the output of the system. Obviously an SP estimate is assumed to be known. Periodically the collected $\hat{d}(n)$ are used to perform selection, adding or removing regressors to the controller model. As we have already said, selection is done virtually not to affect online performance with wrong models. The selection stage works, more or less, like the previously cited SEMP. In fact, it is composed by a phase where new regressors are added to the model and a pruning phase. Here, parameters estimation is done by means of EWU while the discrimination of the best configuration is made by comparing the MSE J:

$$J(\Phi, n) = \frac{1}{N} \sum_{i=0}^{N} e_{\Phi}^2(n-i)$$

where N is the test block length, that is the number of steps on which the different configurations are tested (i.e. the number of $\hat{d}(n)$ given to the selector) and Φ and e_{Φ} represent respectively the set of regressors of the evaluated model and its output error. The whole process is constituted by four different components: the main line, the adder, the pruner and the tester. The main line

deals with the real system: it updates the coefficients of the controller model with a EWU (or one of its variants) and propagates the output to attenuate the noise. The adder, as its name suggests, adds regressors to the model and tests if some configuration can bring advantages. The pruner, that is always used after the adder, removes non significant regressors from the model. The tester is a component that deals with the realignment on the main line of the model, found in the adder-pruner. The last three components represent the selection algorithm and they run in parallel with respect to the main line. EWU selection algorithm works as follows:

1. the main line continues to run the EWU algorithm independently from the others but collecting, at each step, the primary path estimate $\hat{d}(n)$ The controller model of the main line is represented by the set of regressors $\Phi_{current}$

2. when there is a sufficient number of $\hat{d}$ samples, or when a new model has been already selected, the adder machine is invoked

3. the adder runs in parallel a number of EWU equals to the number of available regressors Φ_{out}. This number depends on the current model $\Phi_{current}$ and on the selected order and memory for the NARX controller model. Hence, for each parallel EWU we have a set of regressors Φ^i_{in}:

$$\Phi_{out} = \{ \text{ set of all possible regressors} \} \setminus \Phi_{current}$$

$$\phi^i_{out} \in \Phi_{out}$$

$$\Phi^i_{in} = \Phi \cup \phi^i_{out}$$

   These EWU are fed with the last N $\hat{d}$ collected by the main line until that moment. When they have finished their process all the $J(\Phi^i_{in}, n)$ are computed and the set of regressors Φ^{best}_{in}, that took to the smallest MSE, is selected. At this point if the advantage is significative (i.e. over a predefined threshold) with respect to $J(\Phi_{current}, n)$ the selected set becomes a candidate set Φ_{best} to be inserted in the main line, so:

$$\Phi_{best} = \Phi^{best}_{in}$$

$$\Phi_{out} = \Phi_{out} \setminus \{\Phi_{best}\}$$

If the number of regressors is < 3 then the selection algorithm jumps to the tester, otherwise we enter in the pruning stage

4. the pruner has the same structure of the adder but the number of EWU to be tested is now equal to the number of regressors minus one. This is because it tests on the same $\hat{d}$ the following sets of regressors:

$$\Phi_p^i = \Phi_{best} \setminus \{\phi_{pruned}^i\}$$

$$\phi_{pruned}^i \in \Phi_{best} \setminus \{\text{last inserted regressor}\}$$

where each Φ_p^i is a set of regressor to be tested and ϕ_{pruned}^i is the regressor removed from the current candidate set Φ_{best}. The pruning stage continues like the adder comparing the resulting $J(\Phi_p^i, n)$ with $J(\Phi, n)$. If some of them is smaller, the pruning is confirmed:

$$\Phi_{best} = \Phi_p^{best}$$

$$\Phi_{out} = \Phi_{out} \cup \{\phi_{pruned}^{best}\}$$

The pruning stage is repeated until it is possible to remove regressors maintaining an advantage and until the number of regressors is > 3. When one of the two previous conditions is not satisfied the algorithm goes on to the testing stage

5. the testing phase does nothing but to run a EWU with the last N $\hat{d}$ collected by the main in the time of execution of the adder-pruner machine. This stage is meant to realign the coefficients W associated to the regressors in order to obtain immediately a discrete level of attenuation on the main line. After this stage, the model is updated and a new adding phase is launched

$$\Phi_{current} = \Phi_{best}$$

$$J(\Phi_{current}, n) = J(\Phi_{best}, n)$$

6. the algorithm ends when the error output drops below a specified threshold

Obviously the smaller is the test time the lower is the complexity, on the other hand if N is too small there is no time for the parallel EWUs, in the adding and pruning phase, to reach convergence. In order to obtain a good result with the least computational effort, this trade-off must be taken carefully into consideration. Nevertheless the computational complexity of EWU selection remains high. In chapter 5 we will see if this complexity is rewarded with a good model selection.

Chapter 5

Tests

5.1 Introduction

We will now show the performance of the proposed method through Matlab simulations. A bunch of different tests are used in order to show the behavior of the EWU in different scenarios. In particular we will compare the results of the 2S-EWU-SSD with the NFGLMS, the VFXLMS and the BFxLMS methods. Stochastic methods are too slow in convergence to be compared with the two previously mentioned algorithms, even if they reach good attenuation result. The listed results are obtained setting the best step size for each algorithm.

5.2 Convergence Tests

In order to demonstrate the convergence capability of EWU we set up three tests with ascending complexity in the transfer function. In these tests EWU is fed with the right controller model and the convergence value of each coefficient is known. We will compare the obtained results with the outcomes of the same systems using NFGLMS. For each test we will look at the system output, the Normalized Mean Squared Error (NMSE) and the coefficients update paths. The NMSE is computed at each step as:

$$NMSE(n) = 10log_{10}\left(\frac{\sum_{i=1}^{n} e^2(i)/n}{\sigma_d^2}\right) \tag{5.1}$$

where σ_d^2 is the power of the primary noise at the canceling point and e is the outcome of the system.

5.2.1 Test 1

The first test assumes a linear controller and a second order SP:

$$y(k) = ay(k-1) + bx(k) \tag{5.2a}$$

$$y'(k) = \alpha y'(k-1) + \beta y(k) + \gamma y(k-1)y(k) \tag{5.2b}$$

The primary path is the exact product of the SP and an ideal controller with $a = 0.4$ and $b = 0.2$. $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ are respectively 0.1, 0.3 and 0.2. The input signal is a zero mean white noise.

The charts in figure 5.1 show that both the algorithms are able to find the perfect coefficients, which are $a = 0.4$ and $b = 0.2$ and both achieve good attenuation results, as we can see in figure 5.2. On the other hand, we can see that EWU reaches coefficient convergence more rapidly than NFGLMS even if in a more chaotic way (especially regarding first coefficient). This can be also noticed looking at the NMSEs (figure 5.3) of both the algorithm observing that the one correspondent to EWU is smaller.

5.2.2 Test 2

The second test adds complexity to the secondary path, always assuming a linear controller:

$$y'(n) = \alpha y'(n-1)y(n-2) + \beta y(n) + \gamma y'(n-1)y(n) + \delta y'(n-1)y(n)^2 \tag{5.3a}$$

$$y(k) = ay(k-1) + bx(k) \tag{5.3b}$$

The primary path is the exact product of the SP and an ideal controller with $a = 0.4$ and $b = 0.2$. $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$ and $\boldsymbol{\delta}$ are respectively 0.1, 0.3, 0.2 and 0.2. The input signal is again a zero mean white noise.

Again EWU can reach convergence and exhibits a good speed of convergence compared to NFGLMS. The update paths (figure 5.4) of EWU are still more chaotic than NFGLMS but this does not influence the output, as it is

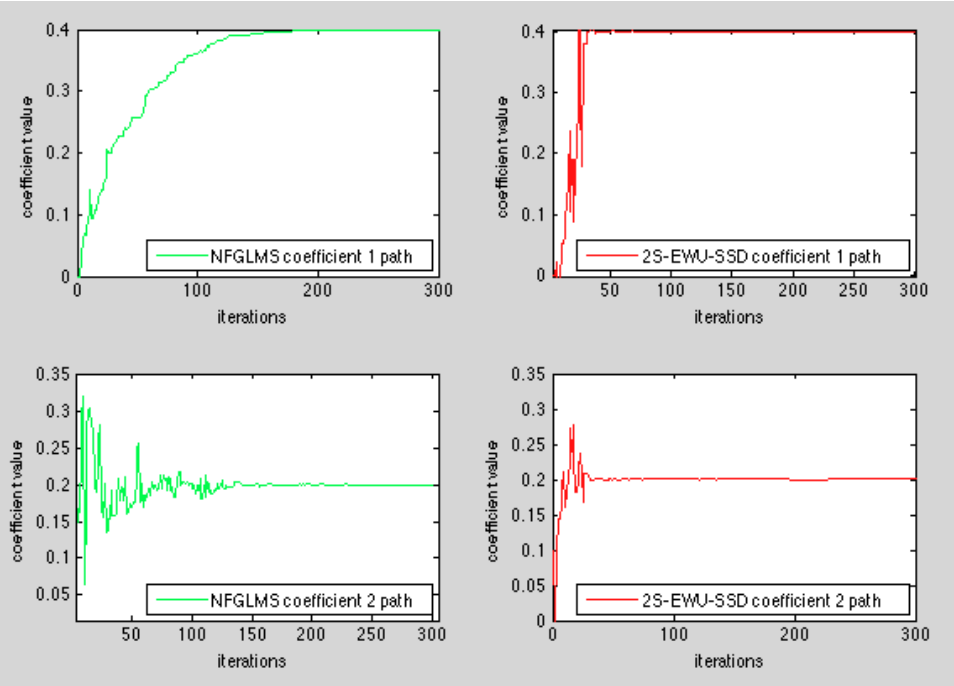

Figure 5.1: coefficients update paths for NFGLMS(left) and 2S-EWU-SSD(right) relative to test 1

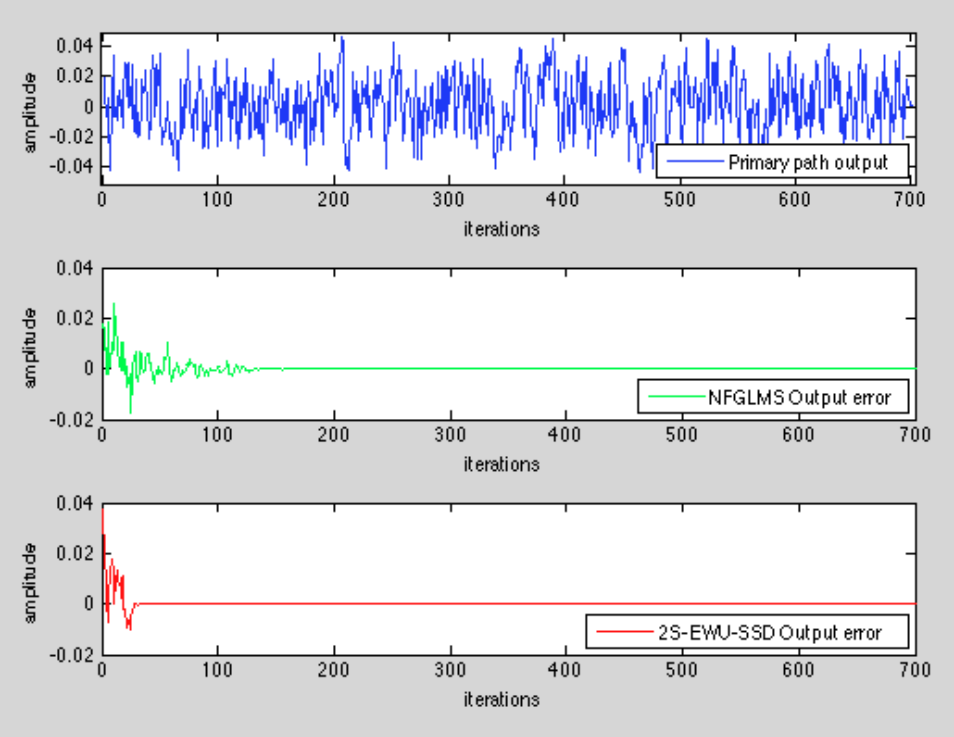

Figure 5.2: output of the system without ANC(blue), with NFGLMS (green) and with EWU(red) relative to test 1

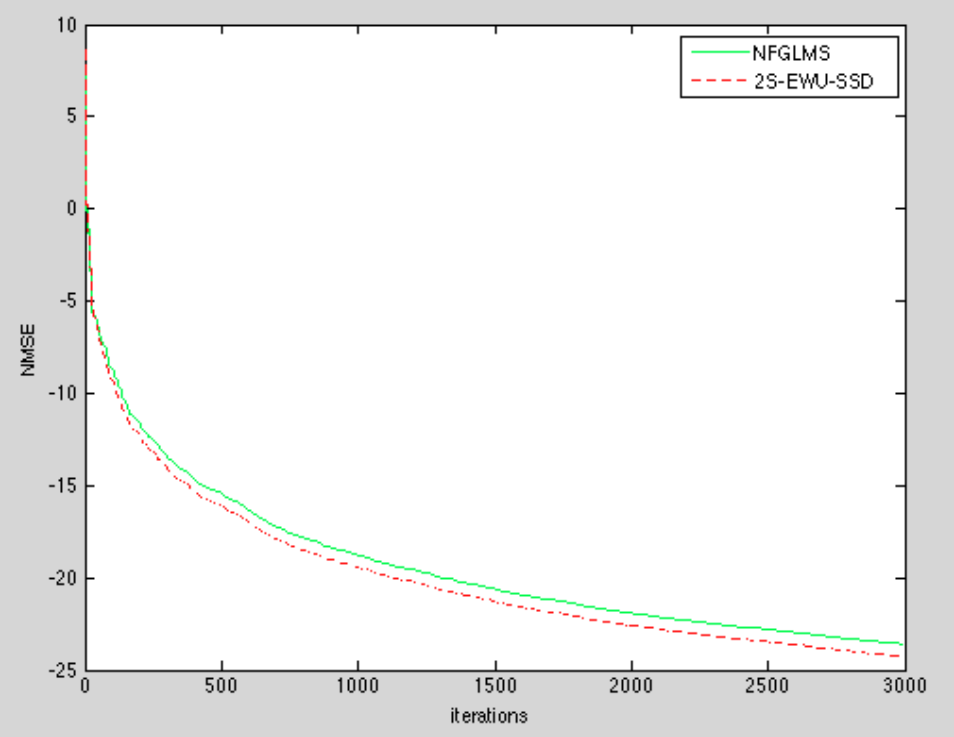

Figure 5.3: NMSE of NFGLMS(green) and EWU(red) relative to test 1

shown in figure 5.5. The NMSEs, in figure 5.6, confirm the better performance of EWU with respect to the gradient method.

5.2.3 Test 3

The third test adds complexity to the controller, assuming the same secondary path of the previous test:

$$y'(n) = \alpha y'(n-1)y(n-2) + \beta y(n) + \gamma y'(n-1)y(n) + \delta y'(n-1)y(n)^2 \quad (5.4a)$$

$$y(n) = ax(n)^2 + by(n-1)x(n-1) + cx(n) \quad (5.4b)$$

The primary path is the exact product of the SP and an ideal controller with $a = 0.4$, $b = 0.2$ and $c = 0.3$. $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$ and $\boldsymbol{\delta}$ are respectively 0.1, 0.3, 0.2 and 0.2. The input signal is again a zero mean white noise.

The results of this test tell that, even if the NMSEs (figure 5.9) are similar, the EWU is able to reach convergence before the NFGLMS (figure 5.7). The outcomes of the two systems are quite similar, but NFGLMS exhibits some noise (figure 5.8).

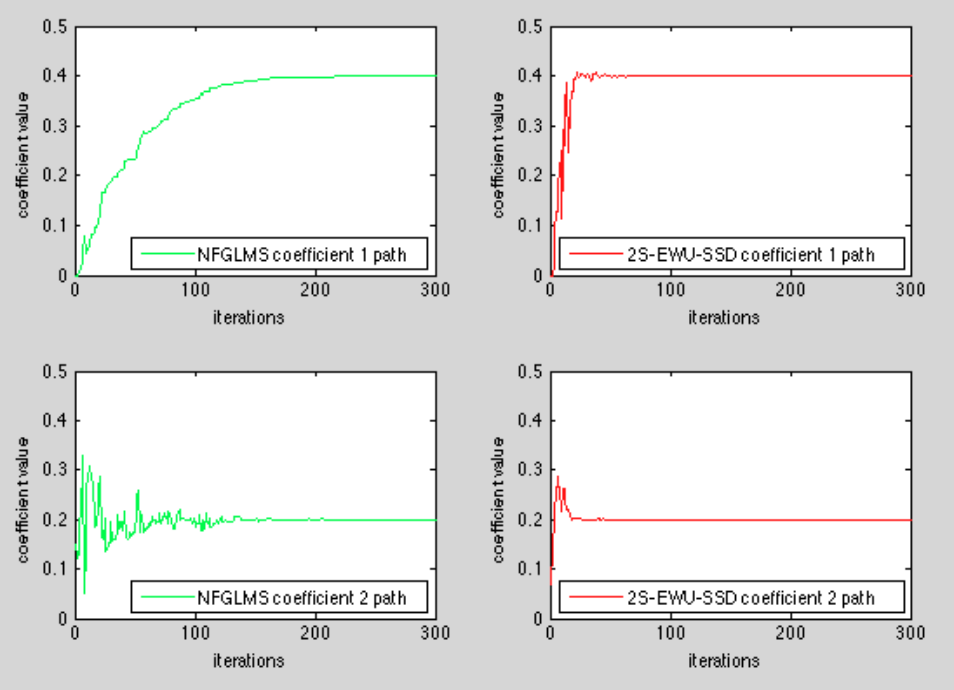

Figure 5.4: coefficients update paths for NFGLMS(left) and 2S-EWU-SSD(right) relative to test 2

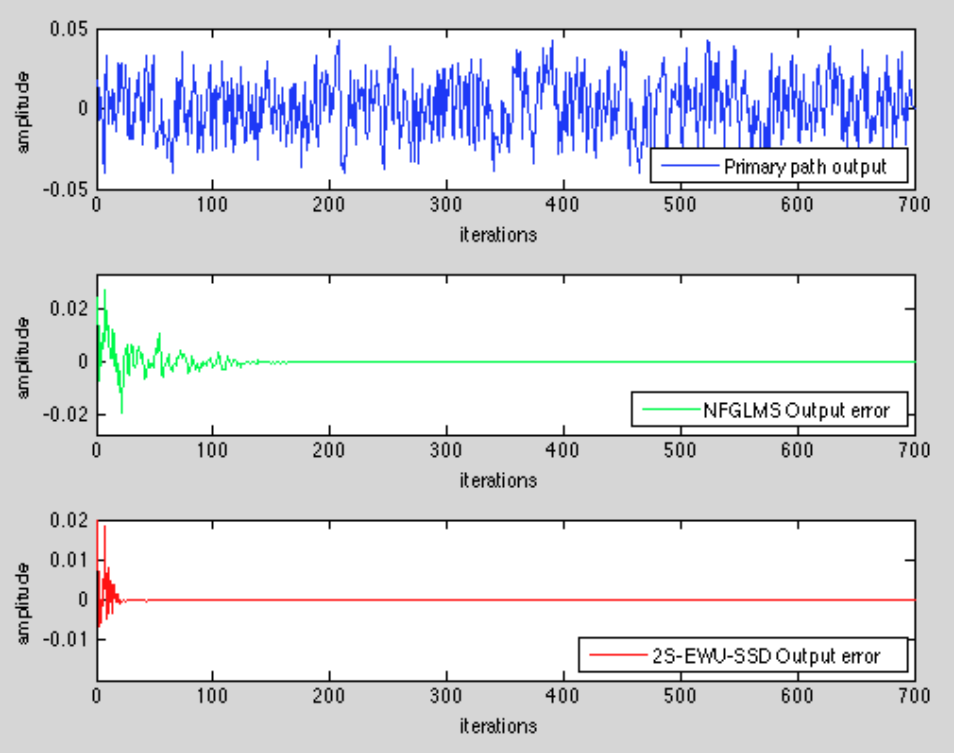

Figure 5.5: output of the system without ANC(blue), with NFGLMS (green) and with EWU(red) relative to test 2

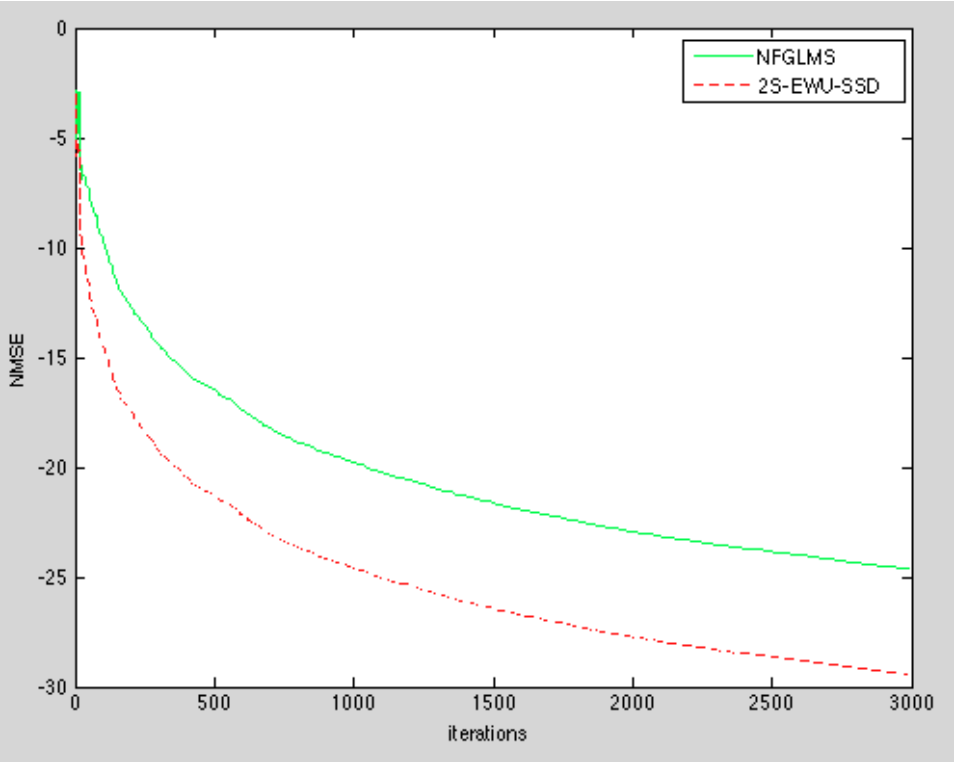

Figure 5.6: NMSE of NFGLMS(green) and EWU(red) relative to test 2

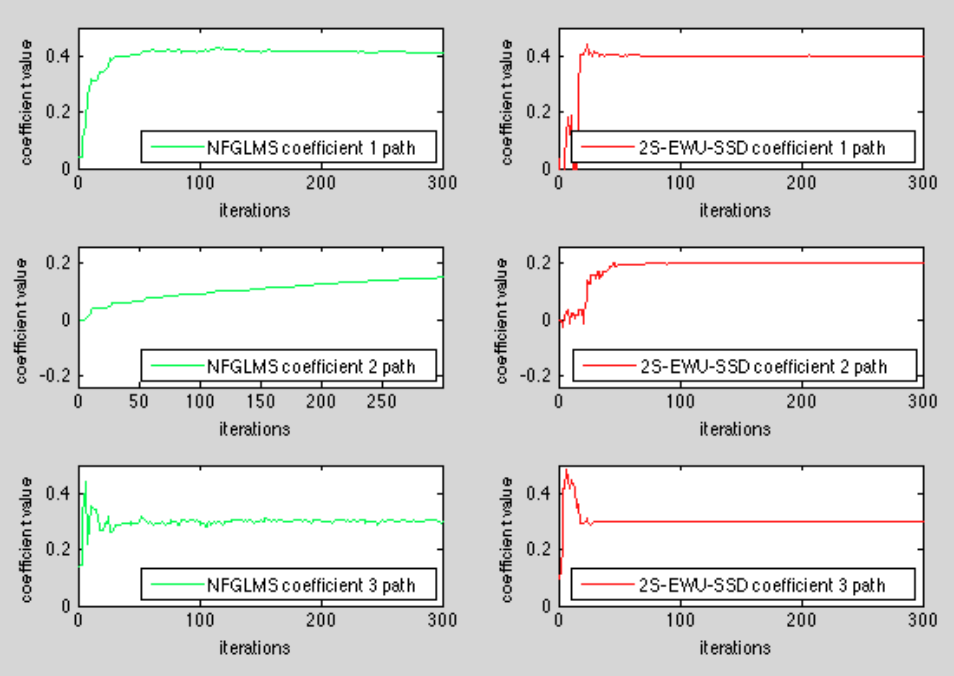

Figure 5.7: coefficients update paths for NFGLMS(left) and 2S-EWU-SSD(right) relative to test 3

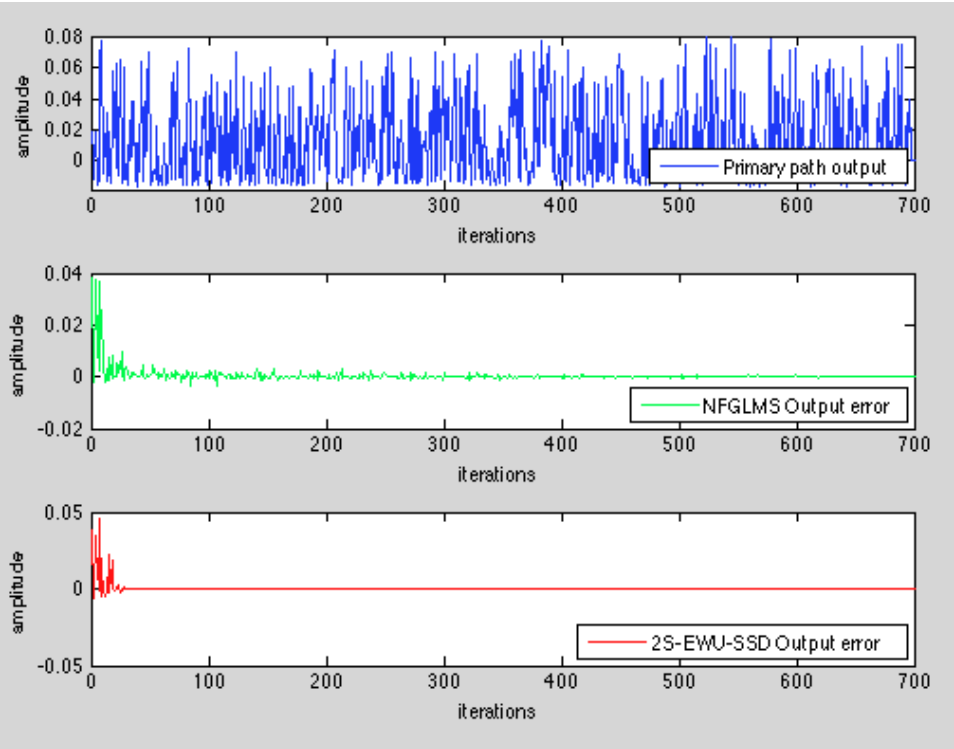

Figure 5.8: output of the system without ANC(blue), with NFGLMS (green) and with EWU(red) relative to test 3

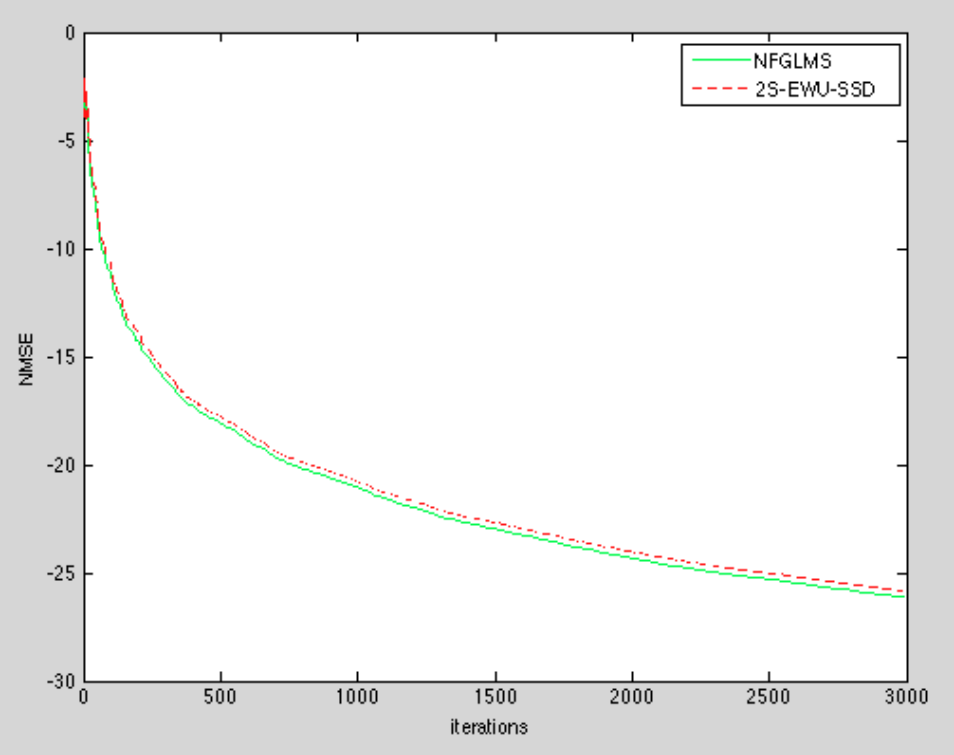

Figure 5.9: NMSE of NFGLMS(green) and EWU(red) relative to test 3

5.2.4 Test 4

The fourth test comprises a model with the same secondary path of test 3 and a more complex controller. Again, the primary path is the exact product of the SP and the ideal controller which is designed as follows:

$$\begin{aligned} y(n) = ay(n-5) + bx(n-4) + cx(n-1) + dx(n-8)y(n-3) + \\ +ex(n-3)y(n-8) + fy(n-4)y(n-6) + gx(n-6)x(n-8) + \\ +hx(n-1)y(n-8) \end{aligned} \tag{5.5}$$

The optimal parameters are $a = 0.4$, $b = 0.2$, $c = 0.3$, $d = 0.1$, $e = 0.5$, $f = 0.3$, $g = 0.4$ e $h = 0.2$.

This test is explicitly designed to see the behavior of the EWU and the NFGLMS with a high the number of regressors. Both the algorithms reach, once again, convergence, but EWU seems to catch sooner the optimal values (figures 5.10 and 5.11). This is true especially for the sixth coefficient. Remembering that the best step-size is chosen for each algorithm, we can see from figure 5.12 that NFGLMS takes more than 3500 steps to reach convergence

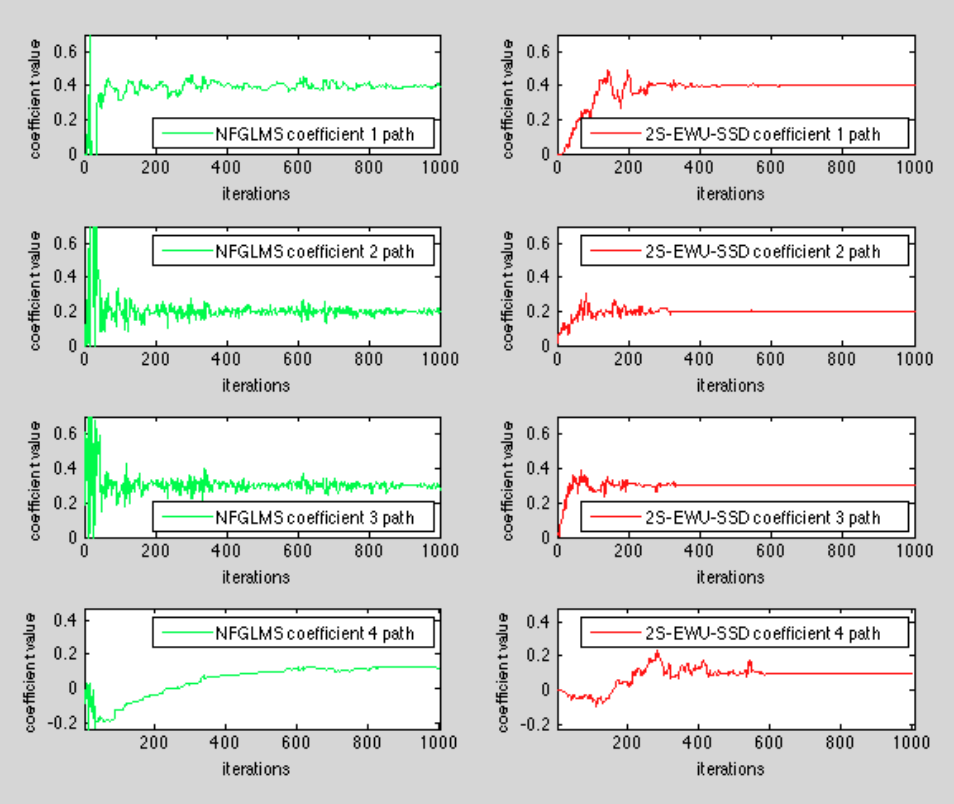

Figure 5.10: 1-4 coefficients update paths for NFGLMS(left) and 2S-EWU-SSD(right) relative to test 4

while EWU catches the right value after about 800 steps. This difference cause some uncertainty in the early outputs of NFGLMS (figure 5.13) but the NMSEs (figure 5.14) are comparable.

5.2.5 Conclusions on Convergence Tests

5.3 Robustness Tests

This section will test EWU on particular robustness features.

5.3.1 Test 5

This test is designed to observe the behavior of the algorithm in case of an incorrect controller model structure. We will feed the EWU with a wrong

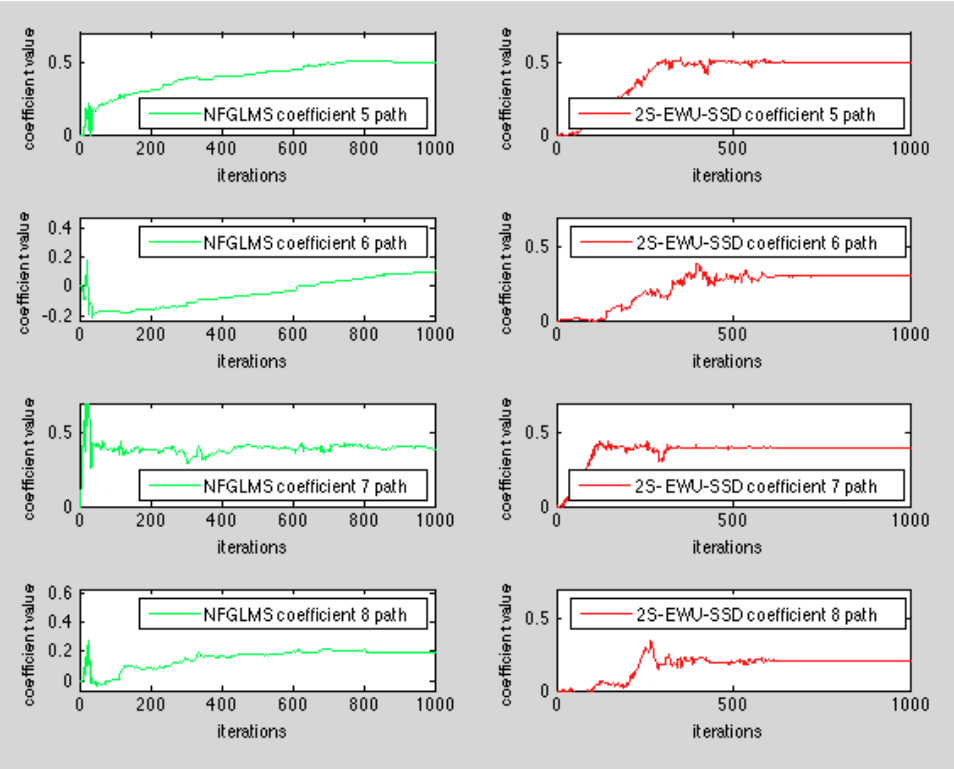

Figure 5.11: 5-8 coefficients update paths for NFGLMS(left) and 2S-EWU-SSD(right) relative to test 4

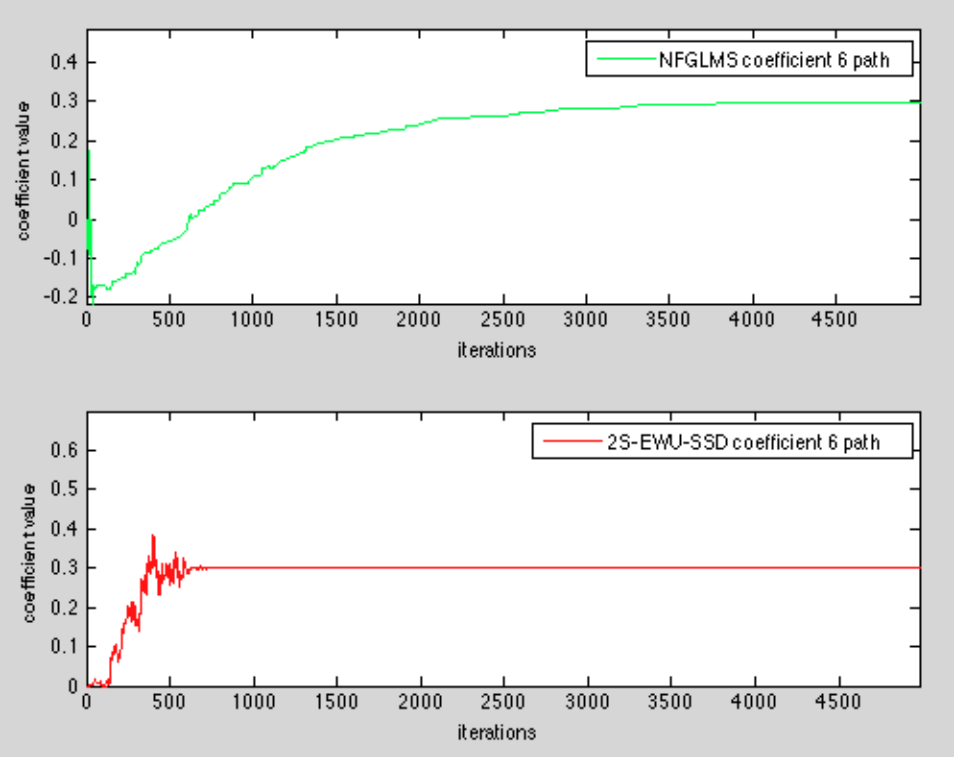

Figure 5.12: 6th coefficient update path for NFGLMS(top-green) and 2S-EWU-SSD(bottom-red) relative to test 4

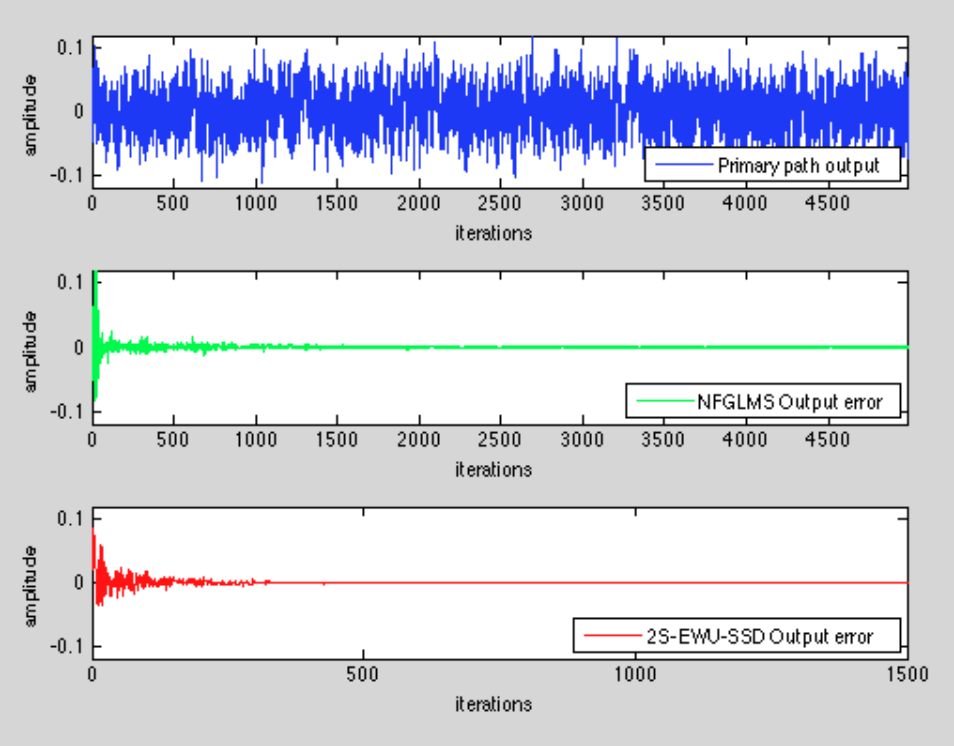

Figure 5.13: output of the system without ANC(blue), with NFGLMS (green) and with EWU(red) relative to test 4

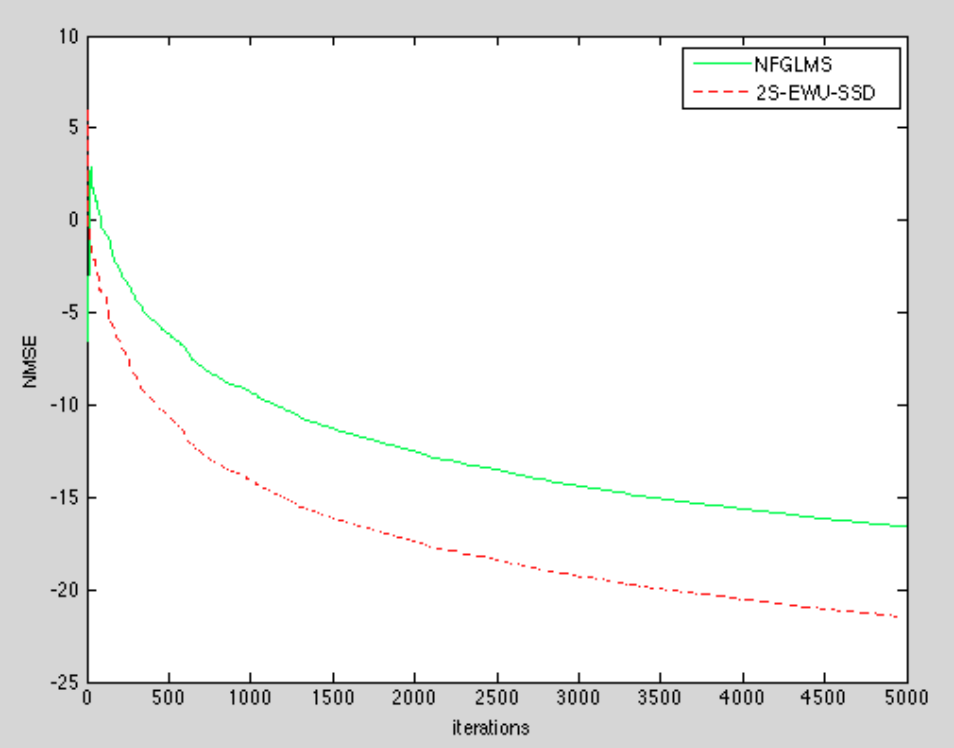

Figure 5.14: NMSE of NFGLMS(green) and EWU(red) relative to test 4

controller model, different from the ideal one used to build the primary path. The latter is equal to the one of test 3. On the other hand, the controller model fed to the EWU is the same as that of test 1.

As we can see in figure 5.16 both the algorithms reach a good attenuation. Though, the performances are way worse than in the previous cases where the exact model was chosen. The NFGLMS and the EWU react in the same way to the structural insufficiency by tuning the coefficients in the same direction (figure 5.15) showing a certain correlation between the two updating methods.

5.3.2 Test 6

This test uses the same SP and controller model of test 4. The primary path, fed with a zero mean random signal, is again the exact product of the secondary path and the controller but with coefficients equal to: $a = 0.4$, $b = 0.2$, $c = 2$, $d = 0.1$, $e = 0.5$, $f = 0.3$, $g = 0.4$ and $h = 0.2$. The resulting output signal exhibits spikes (figure 5.17). This can be dangerous for the correct tuning of

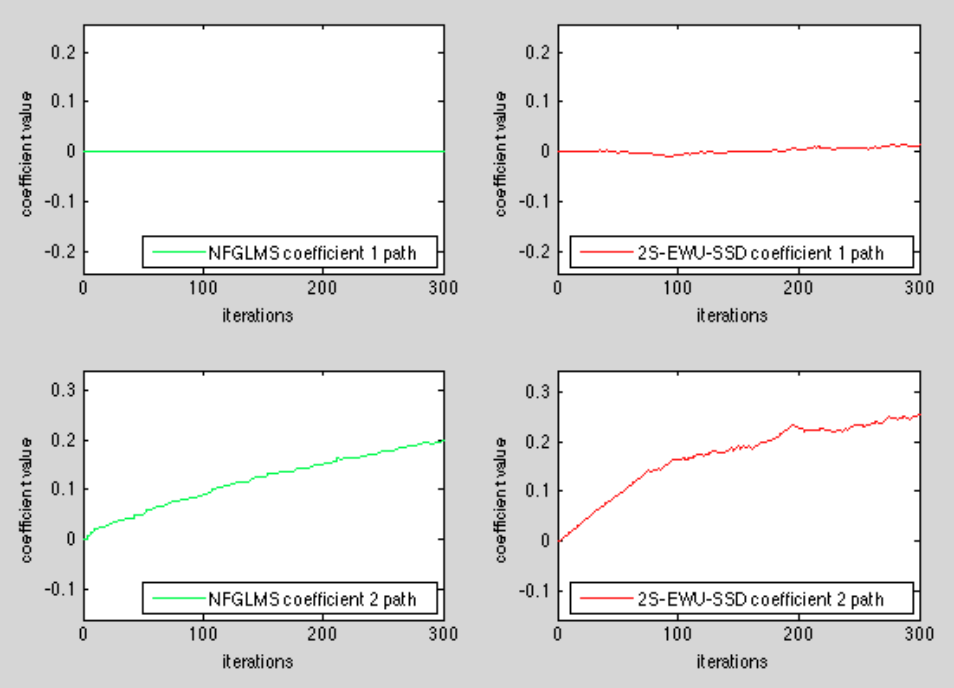

Figure 5.15: coefficients update paths for NFGLMS(left) and 2S-EWU-SSD(right) relative to test 5

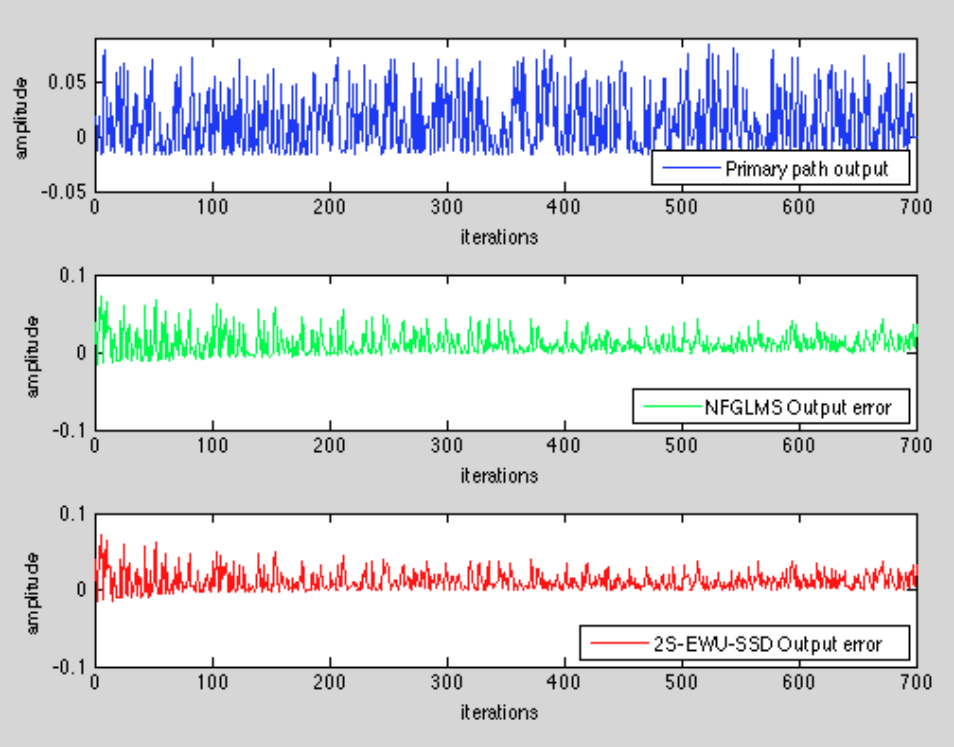

Figure 5.16: output of the system without ANC(blue), with NFGLMS (green) and with EWU(red) relative to test 5

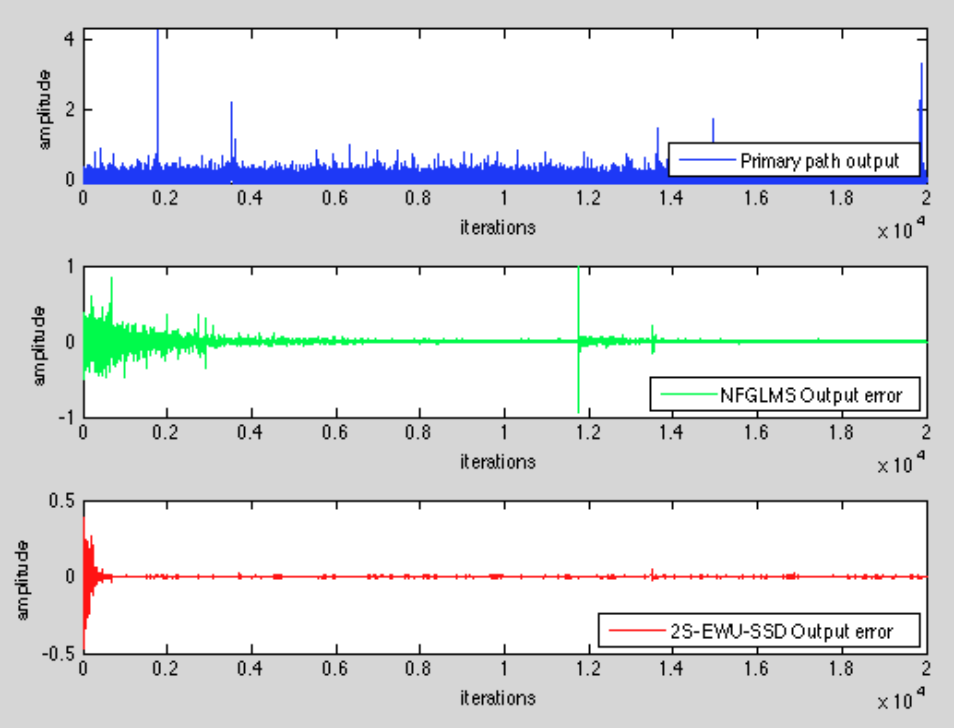

Figure 5.17: output of the system without ANC(blue), with NFGLMS (green) and with EWU(red) relative to test 6

NFGLMS step size. In fact if we use a relatively high μ NFGLMS is not able to update correctly the coefficients and the algorithm fails due to computational problems. On the other hand EWU is not affected at all by spikes and it reaches convergence without any problem. To prevent NFGLMS from failure we have to lower its step size but this affects its performance as shown in figure 5.18.

5.4 Model Selection Tests

This section will face the model selection problem. First, we will test the proposed method with simple systems in order to demonstrate that it can reach the right model. Later we will consider tests found in [30] and [18] and we will compare the results with BFxLMS and VFxLMS.

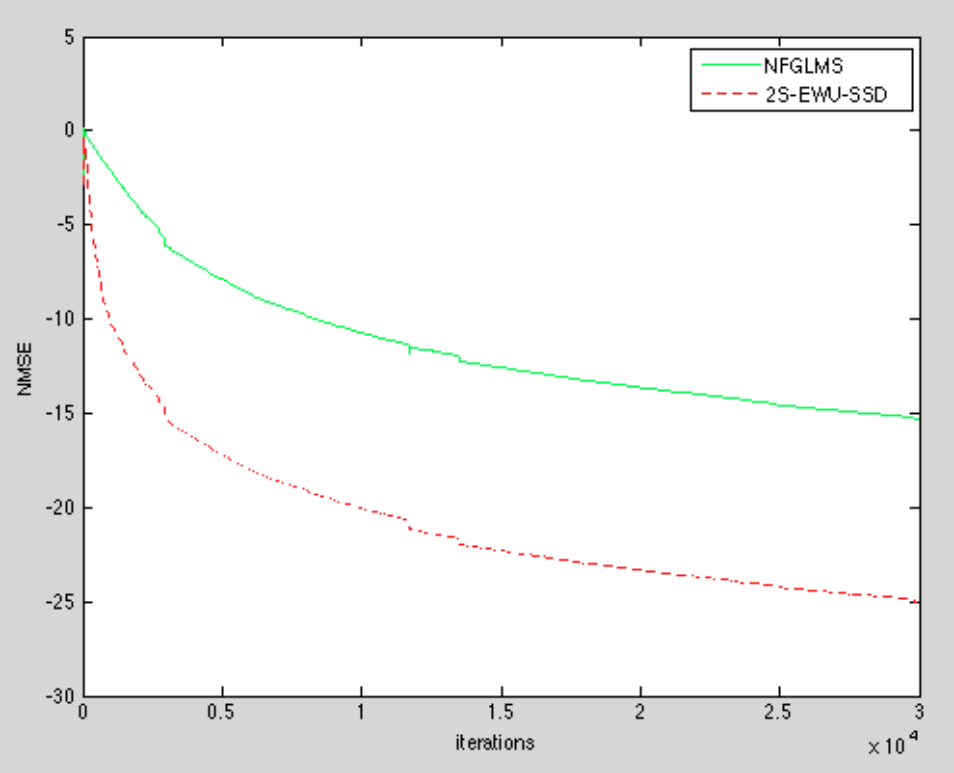

Figure 5.18: NMSE of NFGLMS(green) and EWU(red) relative to test 6

Table 5.1: optimal model test 1

Regressor	Coefficients
y(n-1)	0.4
x(n)	0.2

5.4.1 Model Selection Tests 1 and 2

Test 1 has the same configuration of the test described in 5.2.1. The optimal controller model is known to be the one in the table 5.1.

As we can see from table 5.2, EWU selection finds all the optimal regressors plus one. This redundant regressors comes from the fact that usually the more regressors you have the more you attenuate, but we can also see that its contribute is limited by a small coefficient.

Test 2 is again done with the knowledge of the best controller model. This time the system is configured as in the test reported in chapter 5.2.3 and the

Table 5.2: EWU selection model test 1

Regressor	Coefficients on the main line
$y(n-1)$	0.4
$x(n)$	0.2
$x(n-2)y(n-2)$	0.014

Table 5.3: optimal model Test 2

Regressor	Coefficients
$x^2(n)$	0.4
$x(n-1)y(n-1)$	0.2
$x(n)$	0.3

best controller is presented in table 5.3.

The algorithm adds, again, one more regressor to the model but its contribute is reduced by small coefficients on the main line. The found regressors are the ones in table 5.4.

Considering this first two tests, we can say that the EWU selection is able to reconstruct almost perfectly the model of the controller. Though, in these cases the estimated secondary path equals the real one and, consequently, $\hat{d}$ equals d.

Table 5.4: EWU selection model Test 2

Regressor	Coefficients on the main line
$x^2(n)$	0.4
$x(n-1)y(n-1)$	0.2
$x(n)$	0.3
$y(n-2)y(n-4)$	-10^{-6}

Table 5.5: Controller regressors for Zhou DeBrunner test

	regressors	coefficients on the main line
1	$x(n)$	0.9162
2	$x(n-1)$	0.4281
3	$x(n-3)$	0.3789
4	$x(n-4)$	-0.0864
5	$x(n)x(n-1)$	-0.2574
6	$x(n)x(n-2)$	0.6310
7	$x(n)x(n-5)$	-0.1718
8	$x(n)y(n-3)$	0.7190
9	$x(n-1)y(n-1)$	0.1083
10	$x(n-4)y(n-1)$	-0.0344

5.4.2 Zhou DeBrunner Test

This test is taken from [30]. It consists of a system with a non-linear primary path and secondary path described by the following equations:

$$d(n) = x(n) + x(n-1) + x(n-2) + x(n-3) + \\ + x(n)d(n-1) + x(n)d(n-2) + x(n)d(n-3)$$

$$y'(n) = y(n) + y(n-1) + y(n-2) + y(n)y'(n-1) + y(n)y'(n-2)$$

The input signal is a zero mean white noise.

Running the EWU selection procedure with test-time equal to 500 on a second order NARX with memory equal to 8 we obtain the controller model in table 5.5 and the output shown in figure 5.20. The firsts 500 samples are not attenuated because the system starts with an empty model but this problem can be easily solved using as starting point a standard FIR filter. The selection is done supposing a computation time of 50 steps. This delay is also useful to assure the selection to be general because it introduces in the update process 50 new samples. In figure 5.19 we can see which regressors took part in the selection procedure along 3000 steps.

Now we can use the selected model with 2S-EWU-SSD and compare the results with a VFxLMS with order and memory equal to 2 and 8 and a NFGLMS

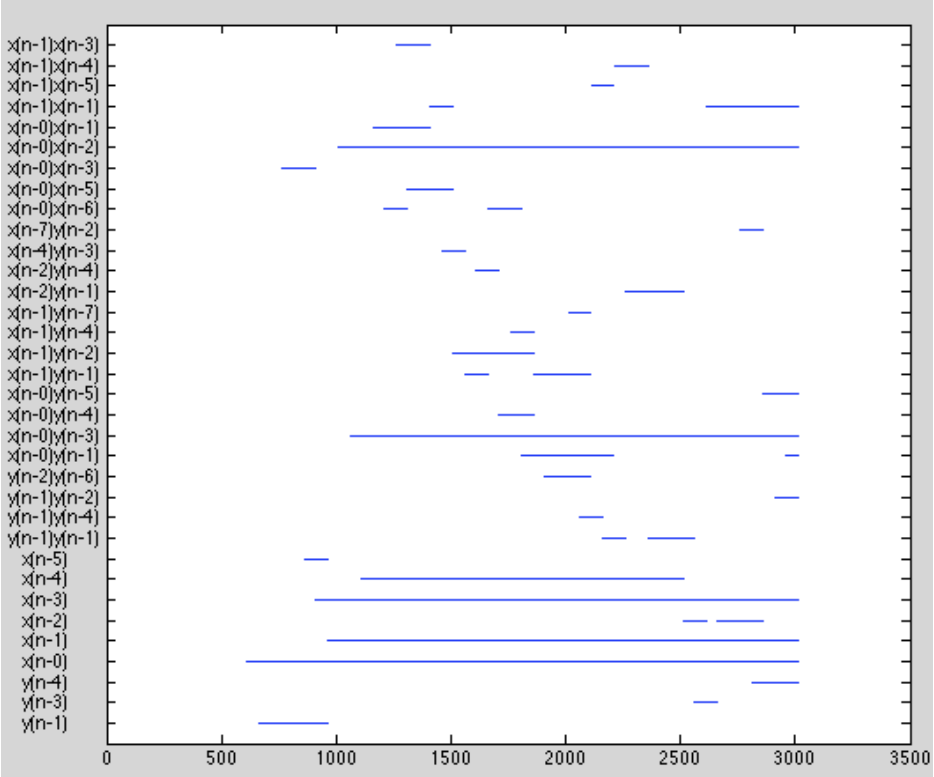

Figure 5.19: Selection history relative to DeBrunner test

used with the full second order NARX which selection was run upon. From figure 5.21 and 5.22 we can see that VFxLMS can barely reach the performance of the selected controller applied to 2S-EWU-SSD. Moreover, the NFGLMS performance shows that even if the number of the selected regressors (8) is more than an order of magnitude lower than the one of the full model (170) the EWU can still reach the same attenuation in the same time.

5.4.3 WU Saturation Test

The previously described tests consider only polynomial nonlinearities of an ANC system. In the next three tests we will introduce a different type of nonlinearity: saturation of the microphone. The input signal is the same in the three tests and is the sum of three sine waves at the normalized frequencies of 0.02, 0.04, and 0.08 with sampling frequency set at 8000Hz. The primary

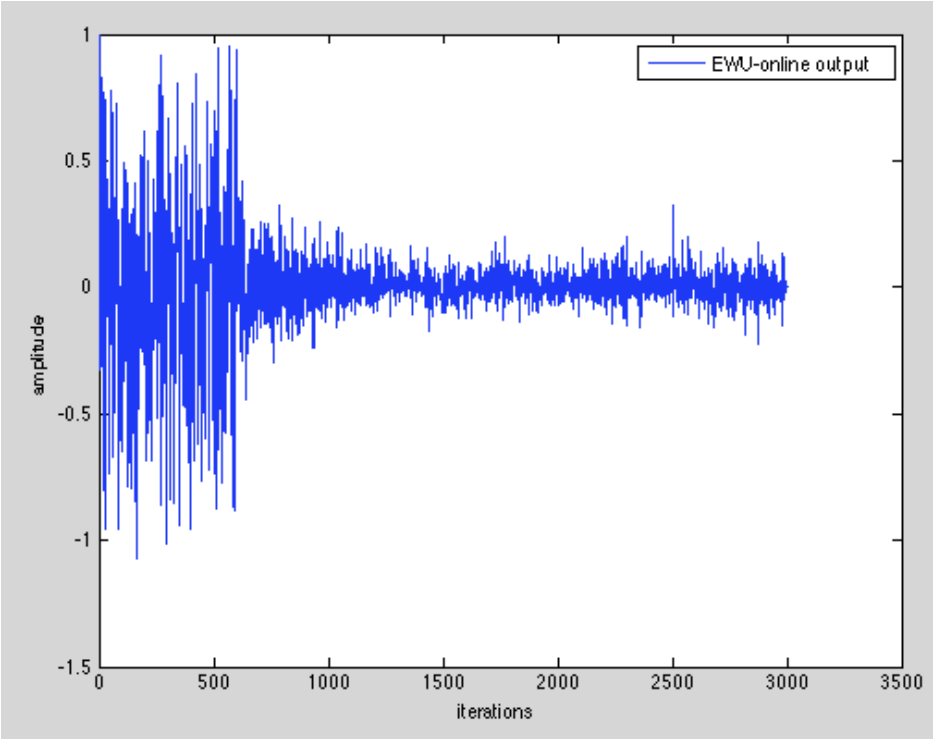

Figure 5.20: System output of EWU Selection relative to DeBrunner test

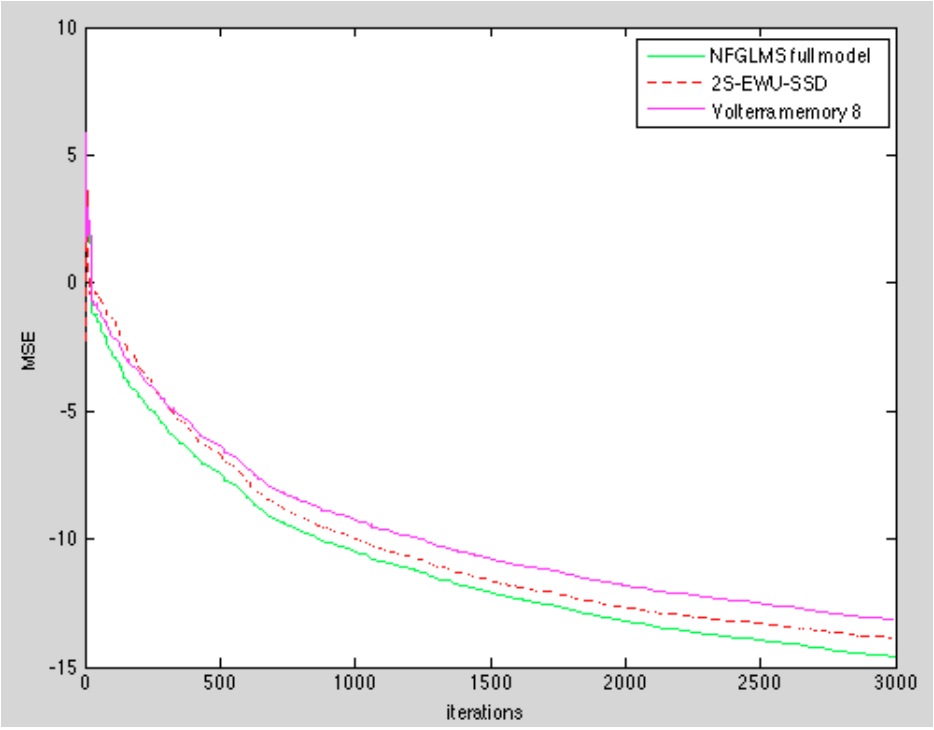

Figure 5.21: NMSE plot of NFGLMS, 2-Stage EWU SSD and VFxLMS relative to DeBrunner test

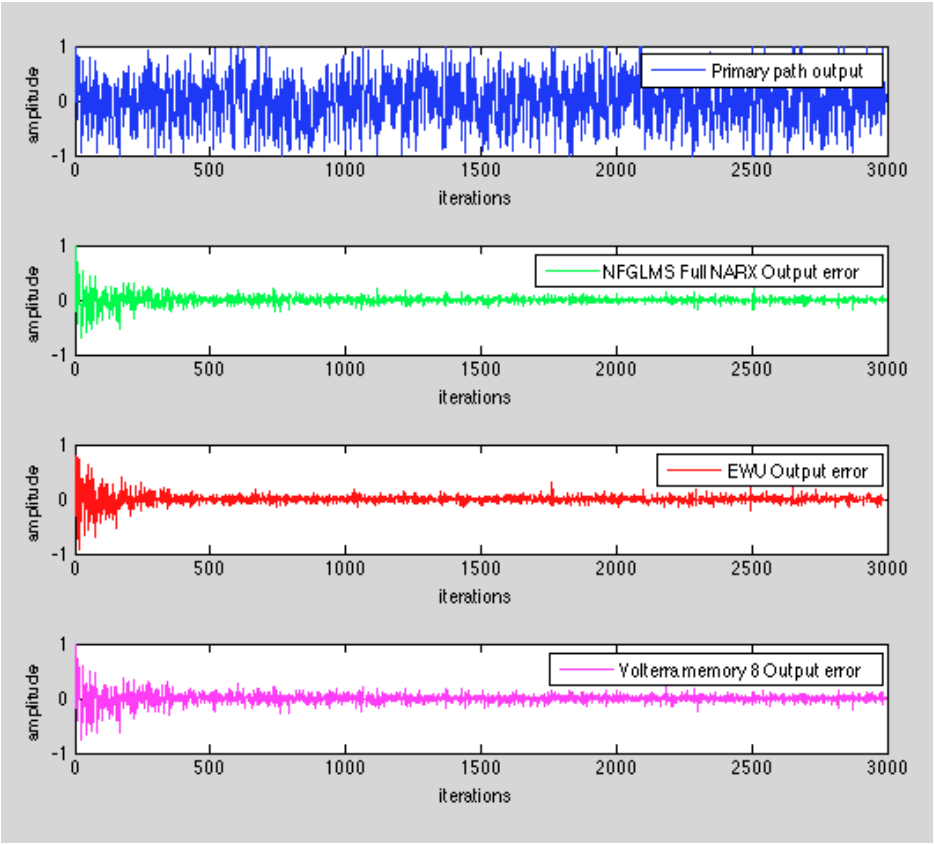

Figure 5.22: System outputs of NFGLMS, 2-Stage EWU SSD and VFxLMS relative to DeBrunner test

Table 5.6: Controller regressors for WU saturation test

	regressors	coefficients on the main line
1	$x(n-1)$	0.2686
2	$x(n-3)$	0.6036
3	$x(n-5)$	0.5735
4	$y(n-1)$	-0.4305
5	$x(n-11)x(n-11)$	0.0288
6	$x(n-12)y(n-4)0.0119$	

path is a linear FIR filter defined as:

$$\begin{aligned} d(n) = 0.0179x(n) + 0.1005x(n-1) + 0.279x(n-2) + 0.489x(n-3) + \\ +0.586x(n-4) + 0.489x(n-5) + 0.279x(n-6) + \\ +0.1005x(n-7) + 0.0179x(n-8) \end{aligned} \tag{5.6a}$$

The secondary path is a FIR filter too:

$$y'(n) = 0.7756y(n) + 0.5171y(n-1) - 0.362y(n-2); \tag{5.6b}$$

The nonlinearity introduced in this test is a weak saturation of the reference signal. It is obtained by setting a clipping threshold at 90% of the maximum input signal value.

Running the selection algorithm with memory equal to 16 and test-time equal to 500 we end up with the model in table 5.6 after 10000 steps. The EWU Selection output (figure 5.23) is still not attenuated in the first 500 steps but it suddenly reaches a good level of noise reduction. The history of the selection is shown in figure 5.24 where we can see how regressor are included (or pruned) to (or from) the model.

Figures 5.25 and 5.26 show the outputs of the system controlled by: an EWU fed with the selected model structure, a VFxLMS used together with a second order Volterra filter of memory equal to 16, a BFxLMS with a bilinear filter of memory equal to 16, and an NFGLMS fed with the full NARX model used in the selection. As we can see, the EWU reach convergence before

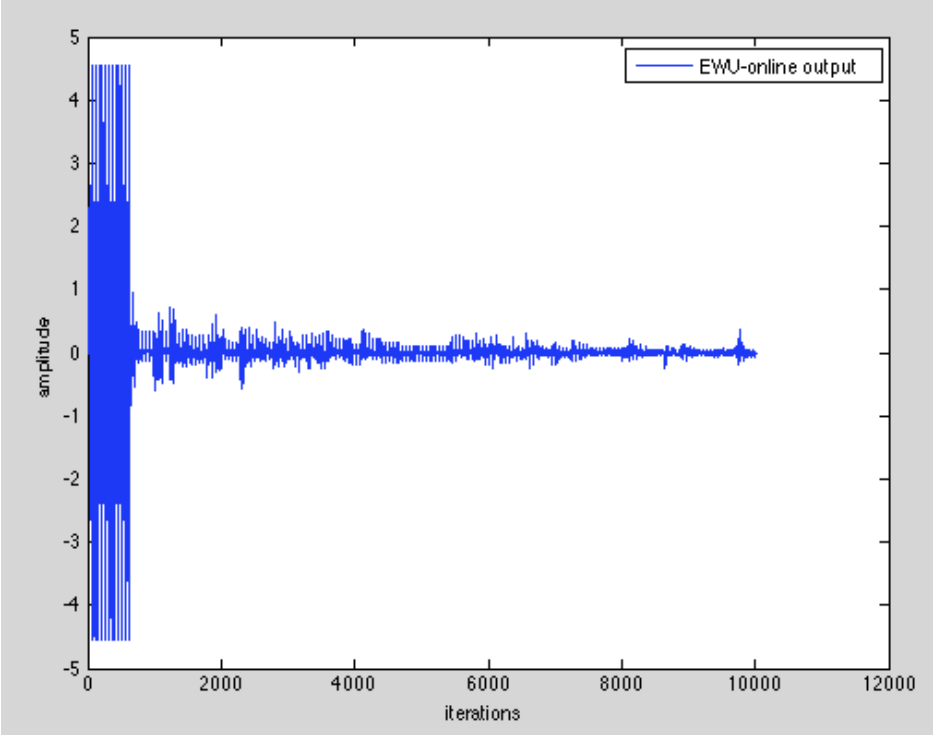

Figure 5.23: System output EWU Selection relative to WU saturation test

every other method and attenuate the error output as much as the BFxLMS which is the best. The comparison between the NFGLMS and the EWU outcomes shows that the selection procedure was able to identify the right regressors, reducing their number by more than two orders of magnitude (from 594 regressors to 6), diminishing the computational effort and improving the performance.

5.4.4 SU saturation test

In this test we hold the same configuration of the previous one but the reference signal is considered to be affected by strong saturation; the clipping factor is now the 50% of the maximum value. The controller regressors, selected with the EWU Selection after 3500 steps, are listed in table 5.7. As before figure 5.28 shows the history of the selection showing the regressors selected or removed at each step.

Even if the saturation is increased, with respect to the previous test, we can notice that the selection method still works sufficiently well, attenuating

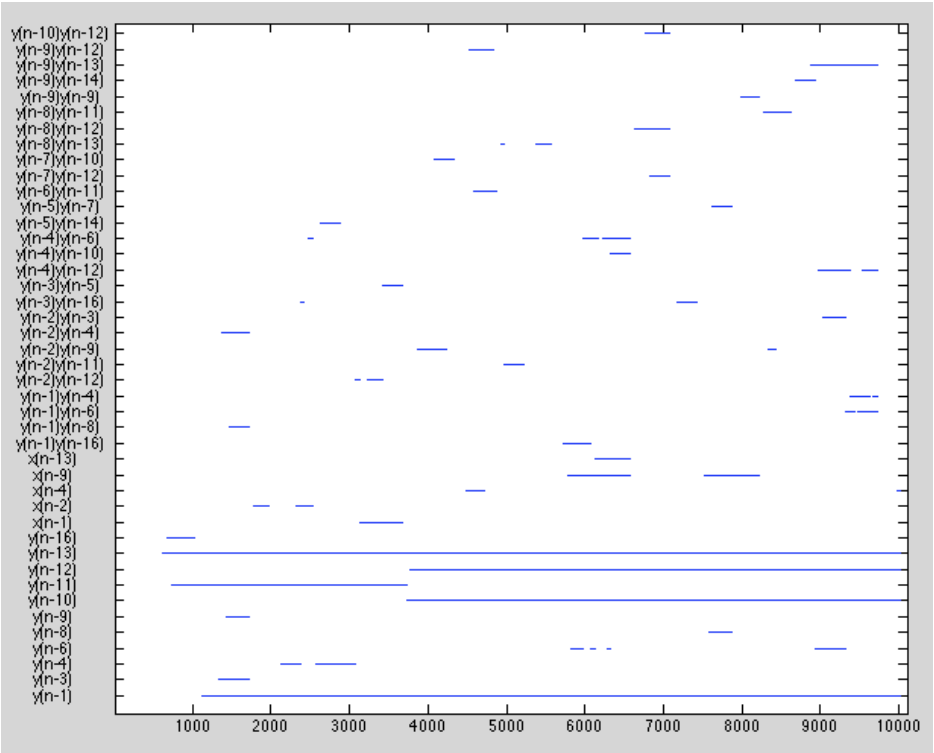

Figure 5.24: Selection history relative to WU selection test

Table 5.7: Controller regressors for SU saturation test

	regressors	coefficients on the main line
1	$x(n-1)$	0.5590
2	$x(n-4)$	1.2434
3	$x(n-6)$	0.8345
4	$x(n-1)x(n-13)$	-0.1633
5	$x(n-13)y(n-9)$	0.0965
6	$y(n-3)y(n-13)$	-0.0128

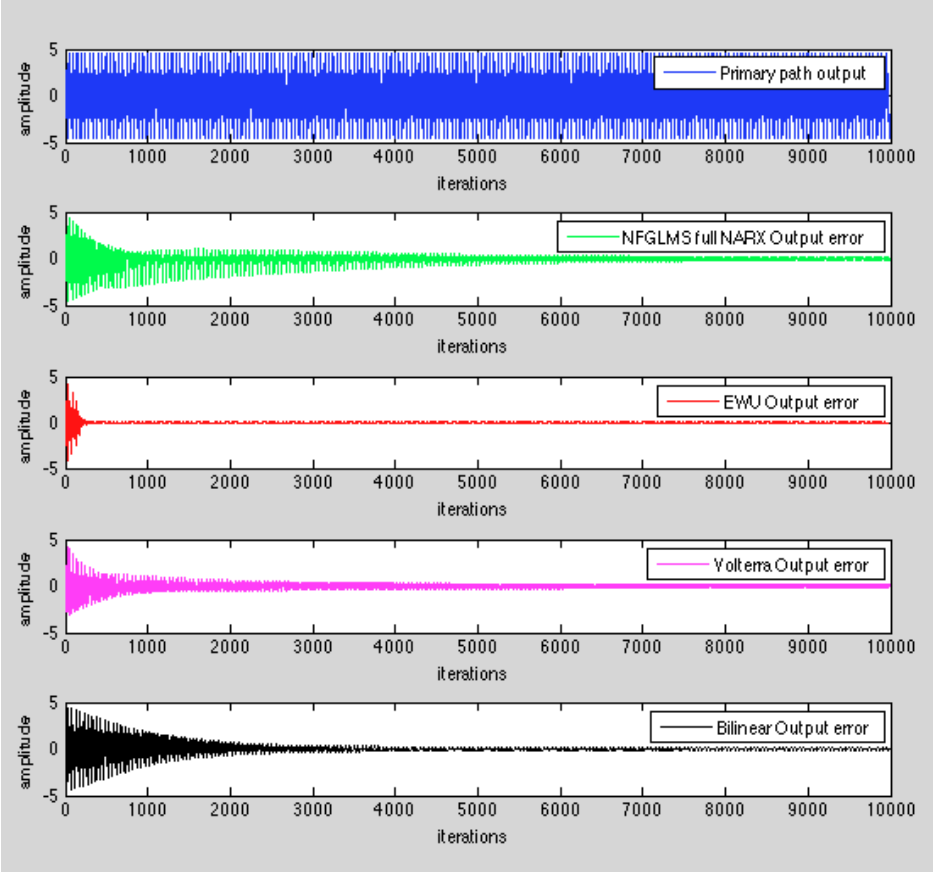

Figure 5.25: System outputs of NFGLMS with a full NARX model (green), 2-Stage EWU SSD with the selected model structure (red), VFxLMS (pink) and BFxLMS (black) with memory equal to 16 relative to WU saturation test

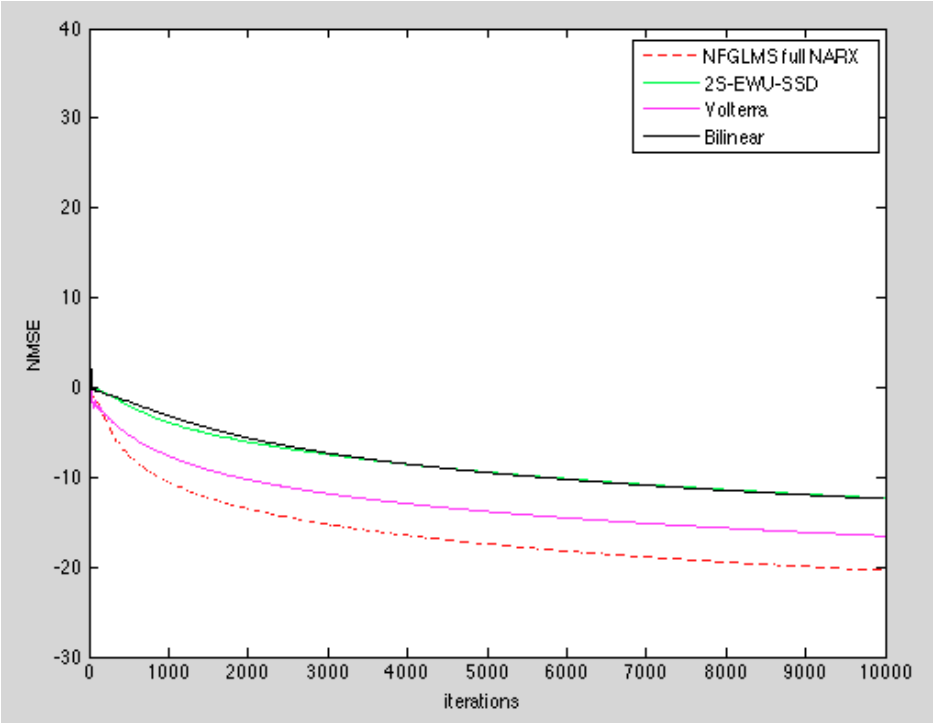

Figure 5.26: NMSEs of NFGLMS with a full NARX model (green), 2-Stage EWU SSD with the selected model structure (red), VFxLMS (pink) and BFxLMS (black) with memory equal to 16 relative to WU saturation test

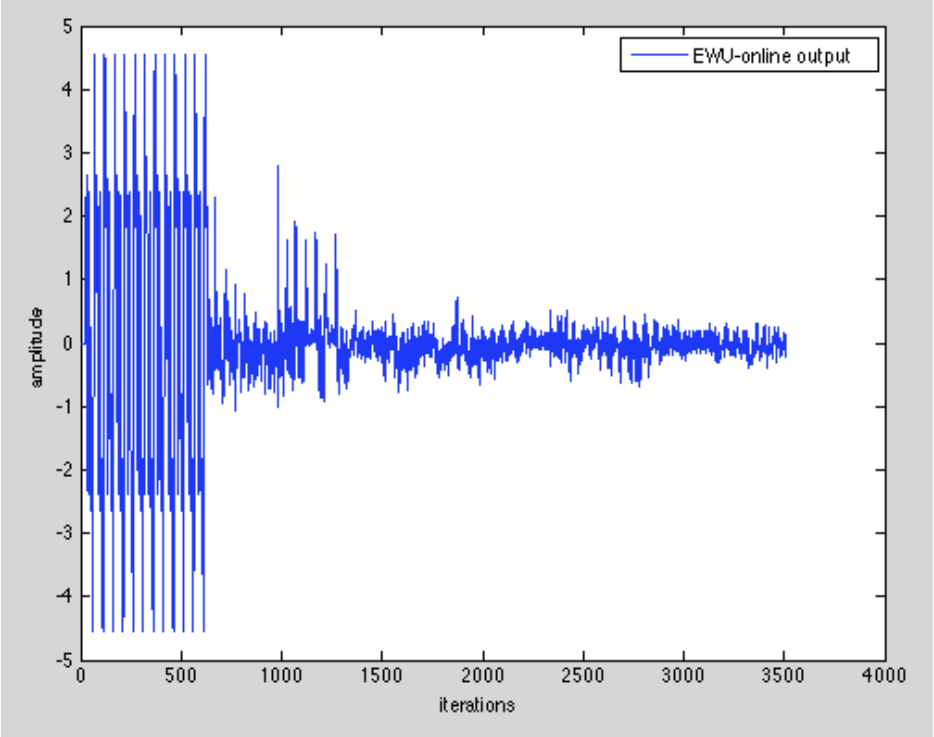

Figure 5.27: System output EWU Selection relative to SU saturation test

successfully the error (figure 5.27). Now, let's have a look at figure 5.29. Here, the outputs of the same test, feeding the 2S-EWU-SSD with the selected model, are shown together with the outputs of the NFGLMS, the VFxLMS and the BFxLMS configured as in the previous test. In this picture, and in figure 5.30, it can be appreciated that, even if the selected model has much less regressors than a second order Volterra filter with the same memory, it reaches faster the same level of attenuation. The performance of the NFGLMS (which runs a full second order NARX model with memory equal to 16) confirms that the selected model structure is a good representation of the system dynamic, in fact the EWU output converges faster and to a lower value than the former output.

5.4.5 SW saturation test

The last test on saturation introduces the same kind of nonlinearity even in the error signal, retaining the strong saturation on the reference signal. The threshold is set at 50% of the maximum error value. The controller for this

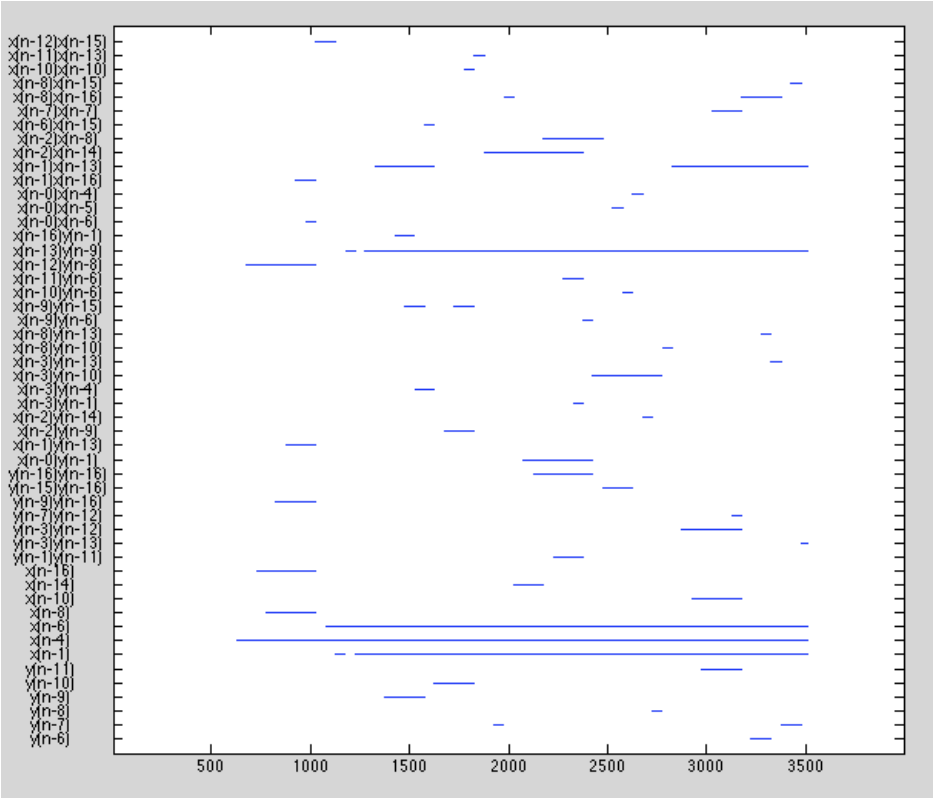

Figure 5.28: Selection history relative to SU selection test

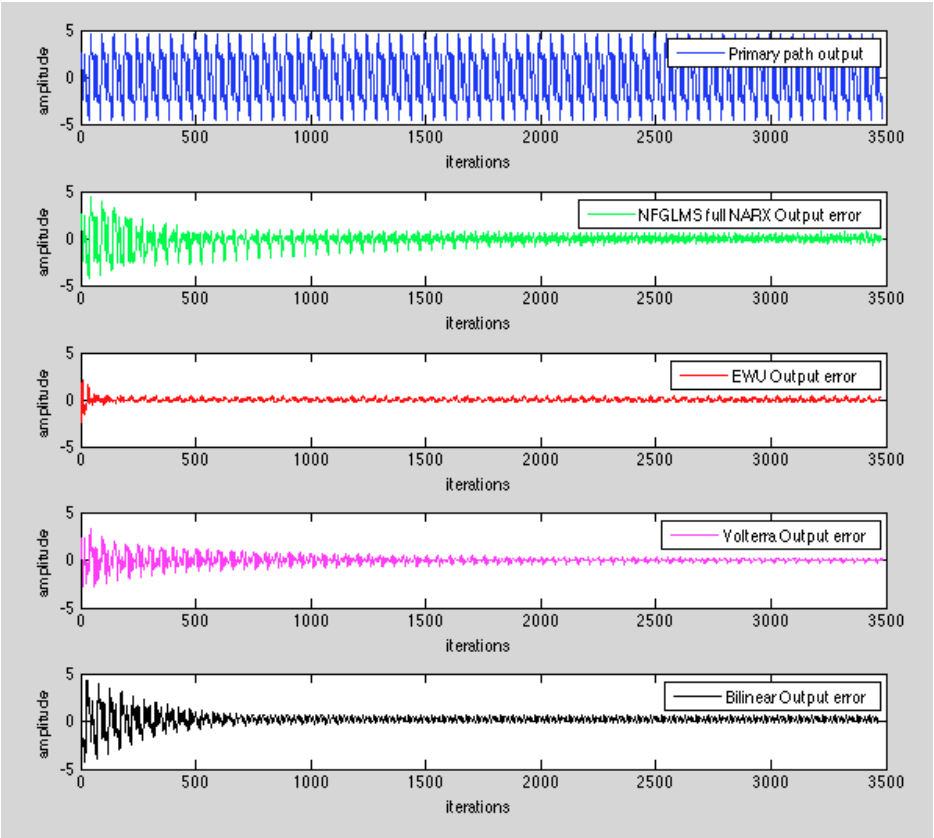

Figure 5.29: System outputs of NFGLMS with a full NARX model (green), 2-Stage EWU SSD with the selected model structure (red), VFxLMS (pink) and BFxLMS (black) with memory equal to 16 relative to SU saturation test

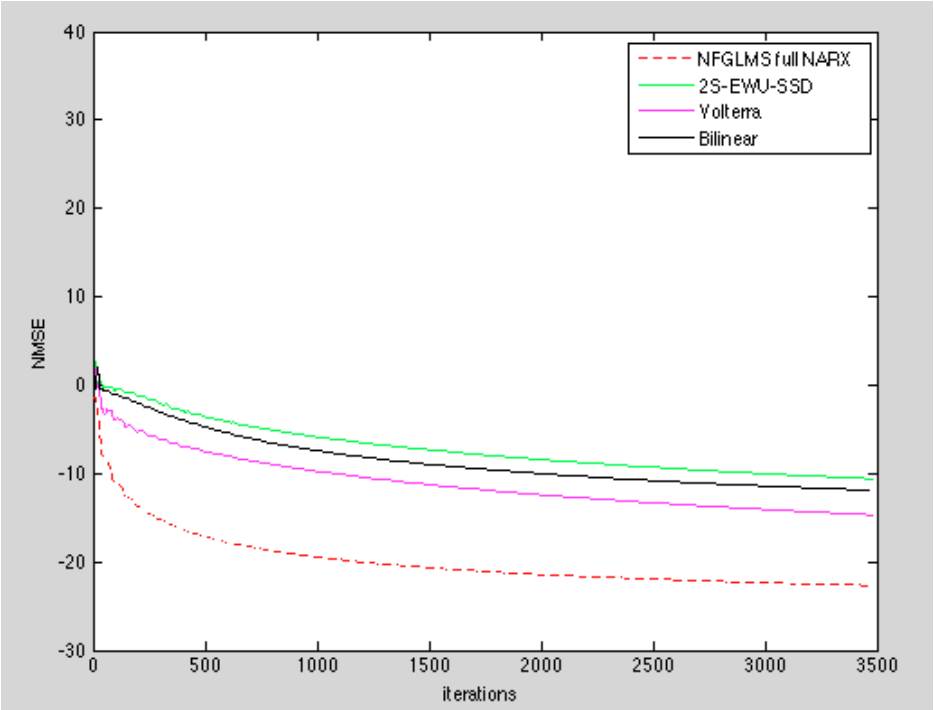

Figure 5.30: NMSEs of NFGLMS with a full NARX model (green), 2-Stage EWU SSD with the selected model structure (red), VFxLMS (pink) and BFxLMS (black) with memory equal to 16 relative to SU saturation test

Table 5.8: Controller regressors for SW saturation test

	regressors	coefficients on the main line
1	$x(n-1)$	0.3920
2	$x(n-4)$	0.9098
3	$x(n-2)x(n-13)$	-0.0314
4	$x(n-1)y(n-7)$	-0.0290
5	$x(n-2)y(n-9)$	0.0077
6	$x(n-9)y(n-13)$	-0.0217
7	$x(n-11)y(n-9)$	0.0669
8	$y(n-1)$	0.5564
9	$y(n-6)$	-0.0447
10	$y(n-1)y(n-14)$	-0.0025

test (found with EWU Selection with the same configuration as the previous tests and after 3500 steps) is designed as shown in table 5.8. As before figure 5.32 shows the history of the selection.

Despite the saturation in the error microphone, that makes the estimated secondary path intrinsically wrong, the EWU Selection algorithm is able to work properly and to reach a good level of attenuation (figure 5.31).

In order to evaluate the correctness of the selected controller model let's have a look at figure 5.33 and 5.34. Like in the previous section, we fed the 2S-EWU-SSD with it to compare their results with the NFGLMS, the VFxLMS and the BFxLMS. Once again the model found with the EWU Selection proves its rightness with respect to the proposed system.

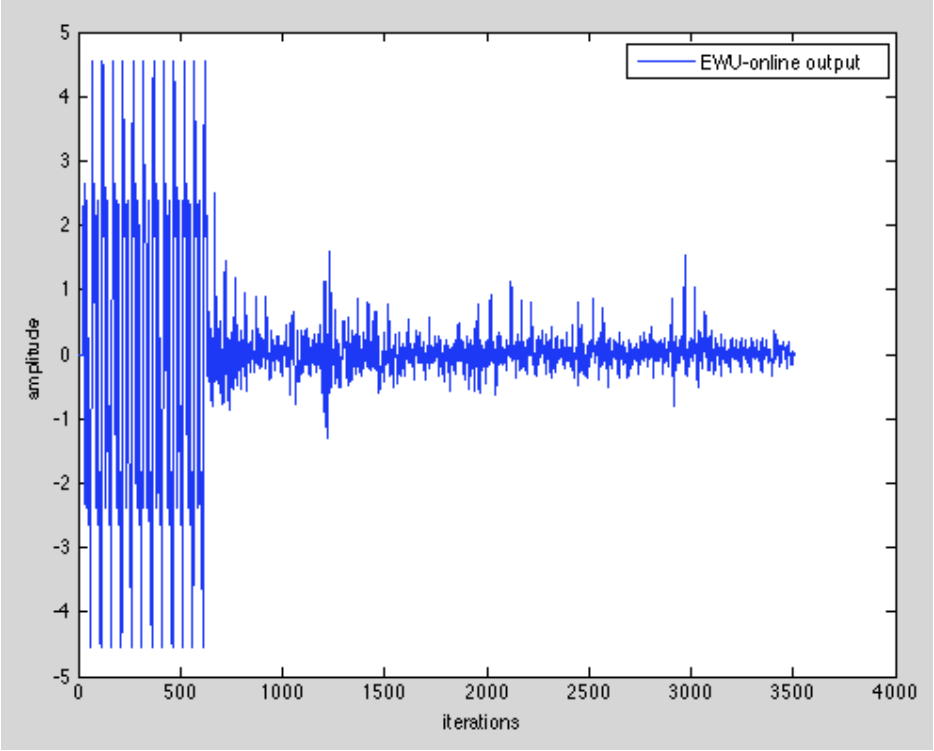

Figure 5.31: System output EWU Selection relative to SW saturation test

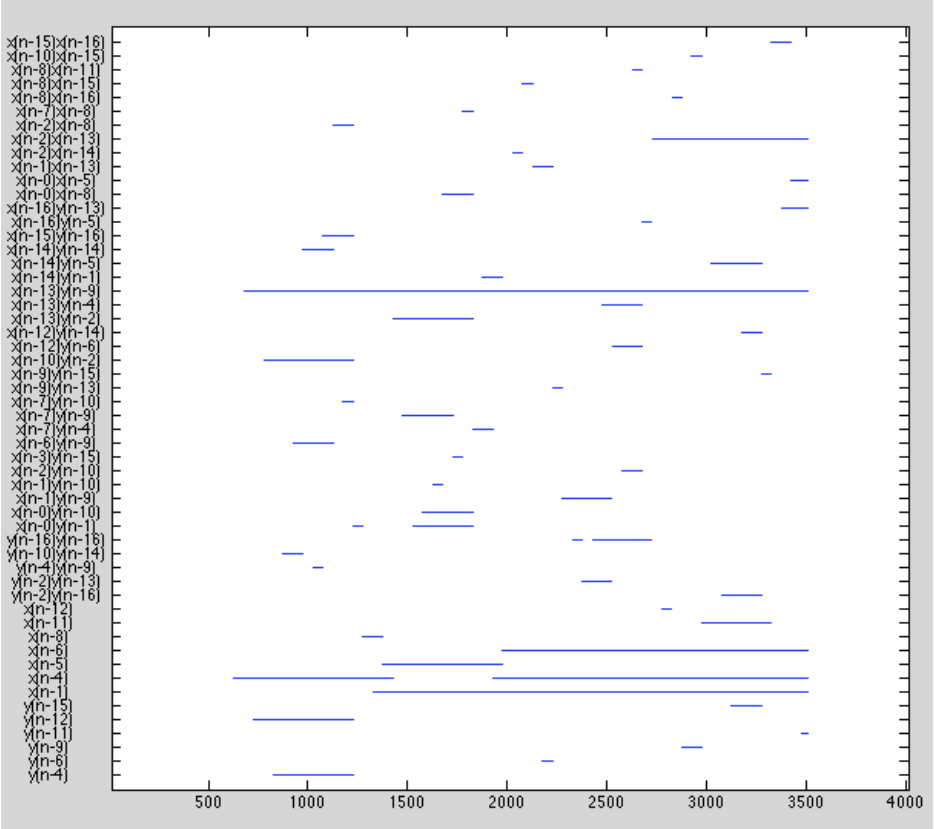

Figure 5.32: Selection history relative to SW selection test

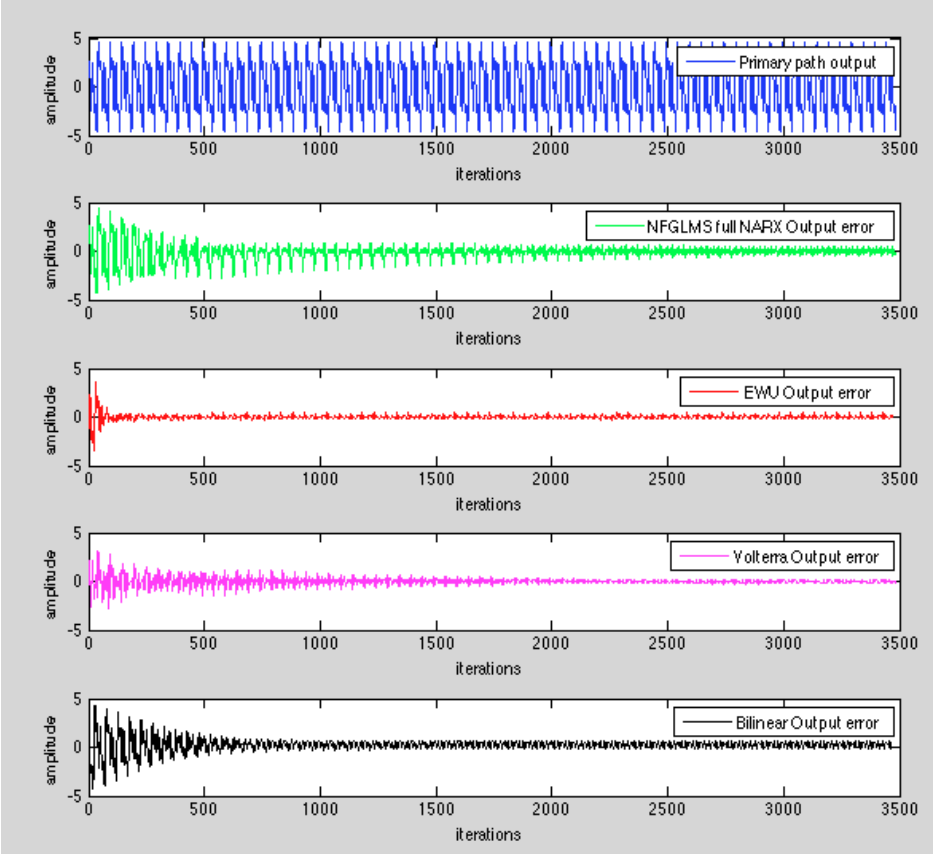

Figure 5.33: System outputs of NFGLMS with a full NARX model (green), 2-Stage EWU SSD with the selected model structure (red), VFxLMS (pink) and BFxLMS (black) with memory equal to 16 relative to SW saturation test

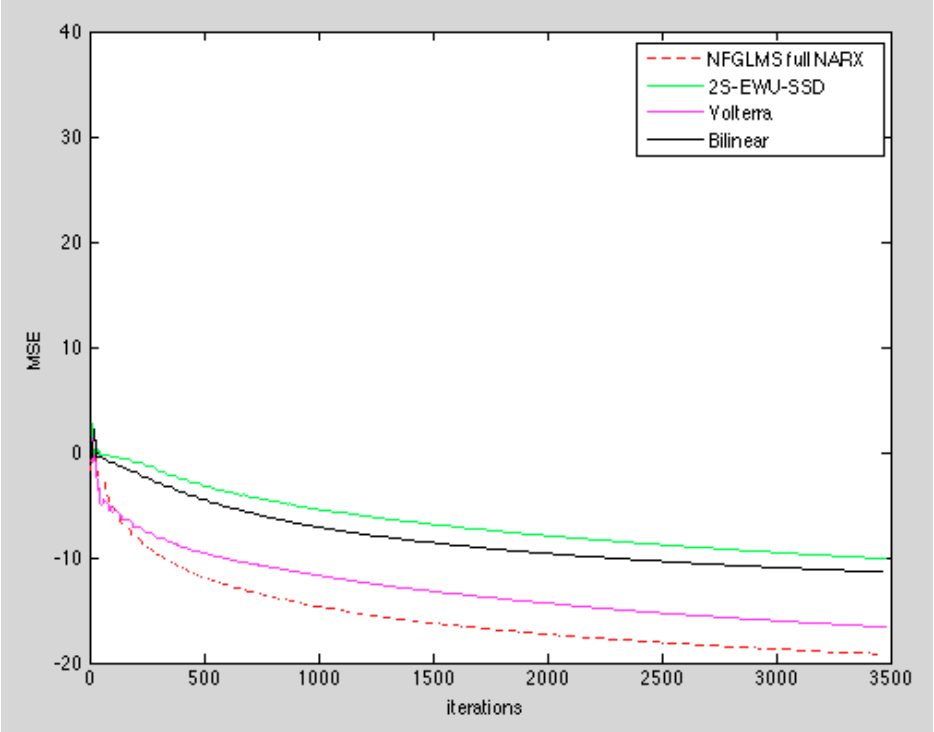

Figure 5.34: NMSEs of NFGLMS with a full NARX model (green), 2-Stage EWU SSD with the selected model structure (red), VFxLMS (pink) and BFxLMS (black) with memory equal to 16 relative to SW saturation test

Chapter 6

Complexity Analysis and Evaluation of Performance

6.1 Introduction

This chapter reports an exhaustive analysis of the computational complexity of the EWU and compared in particular with the NFGLMS, taken as a benchmark. This analysis is complemented with the execution times measured with Matlab, collected with different runs of the scripts and different input data. We also focus on the variance of single iteration execution times, which are significant in real time applications, since the inherent delay in the algorithm must be less than the lag in the primary path. It is worth noting that the EWU variant used is the 2S-EWU-SSD described in chapter 3.5.3, which is the one that leads to the best attenuation results.

6.2 EWU analysis

In this subsection an exhaustive analysis of the theoretical complexity of the EWU is carried out, taking as a reference the algorithm described in chapter 3.2. The analysis counts the total number of additions and multiplications, neglecting the costs of assignment instructions, cycles and variable declarations.

Before starting with the analysis, we must define the length of the vectors

that are used and the cost associated to build them. It is worth to recall that the algorithm relies on the NARX model structure described in chapter 2.3.2 and in order to have a fair comparison between the complexity of the EWU, the NFGLMS and other methods, we retain useful to stick with the definition used in [17], which accounts only for second order nonlinearities. The controller model is reported below.

$$\begin{aligned} y(n) = \sum_{i=0}^{L} a_i(n)x(n-i) + \sum_{j=1}^{L} b_j(n)y(n-j) + \\ \sum_{i=0}^{L}\sum_{j=1}^{L} c_{i,j}(n)x(n-i)y(n-j) + \\ \sum_{i=0}^{L}\sum_{k=0}^{L} d_{i,k}(n)x(n-i)x(n-k) + \\ \sum_{j=1}^{L}\sum_{l=1}^{L} f_{j,l}y(n-j)y(n-l) \end{aligned} \tag{6.1}$$

For the same reasons we assume a linear secondary path. Now, it is useful to count the number of regressors, in order to build two suitable vectors: the coefficient and the regressor vector, so as to calculate the output of the controller as a linear regression. By inspection of equation (6.1) is equal to the number of multiplications required to build the regressors vector. We now define a new variable, called R_C as

$$\begin{aligned} R_C = L + 1 + L + L(L+1) + (L+1)^2 + L^2 = \\ = 3L^2 + 5L + 2 \end{aligned}$$

The variable represents the lengths of the two vectors involved in the linear regression. The same reasoning should be done for the variable R_{SP}, which is the length of the vectors involved in the linear regression that calculates the secondary path output estimate, but, since we assume linear secondary path, we have that no costs are associated to build the vectors. Moreover the length of the vectors is in the worst case equal to $M+1$, when we assume a FIR filter.

$$R_{SP} = M + 1$$

The SP model can be defined as

$$y'(n) = \sum_{i=0}^{M} b_i y(n-i)$$

where b_i are the coefficients of the FIR filter.

We will now start, step by step, the analysis of the complexity of the EWU. The algorithm described in 3.2 is taken into exam and the number of operations required, considering also the optimizations described in the same chapter, are bound to each step.

1. to compute all the regressors we need to accomplish:

$$multiplications = L(L+1) + (L+1)(L+1) + L^2 = 3L^2 + 3L + 1$$

2. to compute the output $y(n)$ of the controller and send it out

$$y(n) = W^T \Phi(n) \tag{6.2}$$

where Φ and W are respectively the regressors' and the coefficients' vectors of the controller, the algorithm needs:

$$multiplications = R_C$$

$$additions = R_C - 1$$

3. to collect the current error $e_0(n)$ at the output

$$e_0(n) = d(n) - y'(n)$$

where $y'(n)$ is the secondary path output and $d(n)$ is the signal to be attenuated, the total complexity amounts to:

$$additions = 1$$

4. to compute the output of the secondary path estimate $\widehat{y'}(n)$ without perturbations on the controller coefficients

$$\widehat{y'}(n) = \Theta^T \Gamma(n) \tag{6.3}$$

where Γ and Θ are respectively the regressors and the coefficients vectors of the SP, the total number of required instructions:

$$multiplications = R_{SP}$$

$$additions = R_{SP} - 1$$

5. for each coefficient it computes the virtual errors corresponding to the two possible directions on its axis, v_{i^+} and v_{i^-}, moving with a step-size μ. Those direction vectors are computed at the initialization stage of the algorithm, and are thus not to be considered part of the main cycle complexity. Now, the algorithm calculates the virtual errors obtained reconstructing d making use of the result of the previous step. Four main operations are usually required, but in the case of a FIR filter for the SP and a second order NARX model for the controller, strong complexity simplifications can be done, following what is described in section 3.2.1. The first instruction that we need to take into consideration is

$$\widehat{d}(n) = e_0(n) - \widehat{y'}(n)$$

which costs:

$$additions = 1$$

then we should have

$$y_{i^\pm}(n) = (W + v_{i^\pm})^T \Phi(n)$$

$$\widehat{y'}_{i^\pm}(n) = \Theta^T \Gamma_{i^\pm}(n)$$

but, thanks to the optimizations involved, it is possible to calculate the output of the virtual secondary path as:

$$y'_{i^\pm}(n) = y'(n) + a((y(n) \pm \mu_i \phi_i) - y(n))$$

that becomes:

$$y'_{i^\pm}(n) = y'(n) \pm a\mu_i \phi_i$$

where a is the coefficient associated to the regressor $y(n)$ in the secondary path and ϕ_i is the coefficient corresponding to the i_{th} regressor that we are perturbating in the controller. Since the product $a\mu_i\phi_i$ is calculated once, the operation costs a total of:

$$multiplications = 2R_C$$

$$additions = 2R_C$$

The last instruction of this set is

$$\widehat{e}_{i^\pm}(n) = \widehat{d}(n) - \widehat{y'}_{i^\pm}(n)$$

which has in total:

$$additions = 2R_C$$

6. for each coefficient it choose the direction $v_{i^{min}}$ that leads to the minimum error

$$v_{i^{min}} = v_j \text{ where } j \in \{0, i^+, i^-\} \text{ such that } \hat{e}_j = \min\left(\{e_0, \hat{e}_{i^+}, \hat{e}_{i^-}\}\right)$$

$$\hat{e}_{i^{min}} = \min\left(\{e_0, \hat{e}_{i^+}, \hat{e}_{i^-}\}\right)$$

Although there are no additions and multiplications, in a real world scenario this could be a potential bottleneck, because, even if we avoid to call the $min()$ function, which for certain languages (e.g. Matlab) can be demanding, we have to estimate an execution time due to all the comparisons that the processor needs to carry out. Our choice is to reduce the calculation of the minimum between the three errors as two subtractions, the first one between e_0 and $\hat{e}_{i^+}$ and the second one between the result of the previous comparison and $\hat{e}_{i^-}$. Thus, since it has to perform this task for each controller coefficients, we can count:

$$additions = 2R_C$$

   (a) if the algorithm choses to not move in a particular direction, its correspondent step size μ_i is halved thus leading to

$$multiplications = R_C$$

   (b) if the algorithm moves a particular coefficient in the same direction for many consecutive iterations, its relative step size μ_i is doubled. This costs

$$multiplications = R_C$$

   Note that, in a real case scenario, the step size is doubled every three steps, where the same direction is chosen, thus the complexity here calculated is a bit overestimated. The two options are mutually exclusive leading to an additional load of R_C in the worst case.

7. the algorithm moves the coefficients accordingly to a linear combination of the selected directions scaling them with a factor q_i:

$$q_i = \frac{(e_0^2 - \hat{e}_{i^{min}}^2)}{\sqrt{\sum_{j=1}^{n_c}(e_0^2 - \hat{e}_{j^{min}}^2)^2}}$$

The value e_0^2 can be calculated once and the same reasoning can be done for the values $e_{k^{min}}^2$, thus leading to R_C+1 multiplications. Furthermore, we need to account for R_C additions for the calculation and the storage of the values $e_0^2-e_{k^{min}}^2$ and additional R_C multiplications for the calculation of the square value $(e_0^2 - e_{k^{min}}^2)^2$. In order to calculate the final value of the norm of the vector q we need $R_C - 1$ additions and 1 multiplication for the calculation of the radix value. Since all the values needed for the calculation of each q_i are stored, R_C divisions must be performed. We chose to count them as R_C multiplications. The total number of operations required for this step is equal to:

$$multiplications = 3R_C + 1$$

$$additions = 2R_C - 1$$

8. to perform the update of the controller's coefficients as

$$W = W + q. * v$$

where v is a vector containing μ_i or $-\mu_i$ for each i_{th} regressor in the controller and $.*$ represents an item by item product between two vectors, the associated complexity is

$$multiplications = R_C$$

$$additions = R_C$$

It is important to note that, even if the EWU variant under analysis is the 2-Stages (2S) with Step Size Doubling (SSD), the code relative to the stage that evaluates the perturbances for a longer time horizon than one, does not impact on performances, since it is run in parallel to the main algorithm and can be spread along more than one step [1].

[1]Note that in this stage the optimization seen so far is no longer available. The number of the involved operations, coming from equation 6.2 and 6.3, is:

$$multiplication = 2Tn_c(R_C + R_{SP})$$

$$addition = 2Tn_c(R_C + R_{SP} - 2)$$

Where T is the time horizon length.

The total cost of the algorithm is summarized below:

$$multiplications = 8R_C + R_{SP} + 1$$

$$additions = 8R_C + R_{SP} - 1$$

If we substitute the values R_C and R_{SP} respectively with $3L^2 + 5L + 2$ and $M + 1$, where L and M are the maximum delays admitted in the model of the controller and in the linear FIR filter of the secondary path, we obtain:

$$multiplications = 24L^2_{controller} + 40L + M + 17$$

$$additions = 24L^2_{controller} + 40L + M + 15$$

When we deal with linear SPs, the NFGLMS error gradient can be simplified as:

$$\nabla e(k) = \frac{\partial y'(k)}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta(k)}} = -\hat{s}(k) * \frac{\partial y(k)}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta(k)}}$$

that with the Feintuch's assumption becomes:

$$\begin{aligned}\nabla e(k) = -\hat{s}(k) * [&x(n) \cdots x(n-L)\ y(n-1) \cdots y(n-L) \\ &x(n)y(n-1) \cdots x(n-L)y(n-L)\ x(n)x(n) \cdots \\ &x(n-L)x(n-L)\ y(n-1)y(n-1) \cdots y(n-L)y(n-L)]\end{aligned}$$

where $\hat{s}(k)$ is the impulse response of the SP. Hence, the NFGLMS complexity can be computed distinguishing between there phases:

1. computation of the controller output $y(n)$; it has the same number of multiplications and additions of the EWU algorithm:

$$multiplications = R_C$$

$$additions = R_C - 1$$

2. computation of the error filtered gradient with respect to the weights, which in the case of a linear SP requires M multiplications and $M - 1$ additions for each controller regressor, that is to say:

$$multiplication = R_C M$$

$$additions = R_C(M - 1)$$

Table 6.1: Table representing the number of multiplications of NFGLMS and 2S-EWU-SSD in case of a liner SP

Method	Total Multiplications
NFGLMS	$(3L^2 + 5L + 2)(2 + M) + 1$
2S-EWU-SSD	$24L^2_{controller} + 40L + M + 17$

Table 6.2: Table representing the number of additions of NFGLMS and 2S-EWU-SSD in case of a liner SP

Method	Total Additions
NFGLMS	$(3L^2 + 5L + 2)(1 + M) - 1$
2S-EWU-SSD	$24L^2_{controller} + 40L + M + 15$

3. update of the controller coefficients, which in this case requires:

$$multiplications = R_C + 1$$

$$additions = R_C$$

Summing up the three steps, NFGLMS needs:

$$multiplications = R_C(2 + M) + 1 = \\ = (3L^2 + 5L + 2)(2 + M) + 1$$

$$additions = R_C(1 + M) - 1 = \\ = (3L^2 + 5L + 2)(1 + M) - 1$$

We can observe that both algorithms have a complexity that goes with the square of the maximum lag admitted for the controller, but NFGLMS has a third order term represented by the product of $L^2_{controller}$ and M. This fact does not necessarily mean that NFGLMS is slower than 2S-EWU-SSD, but that the SP memory has more impact on performance. A brief summary of the complexities is shown in table 6.1 and 6.2.

If we assume to have a full second order NARX model for the SP the variable R_{SP} becomes

$$R_{SP} = 3M^2 + 5M + 2$$

and the calculations previously done in step 4 need to be modified. In particular

$$y_{i^{\pm}}(n) = (W + v_{i^{\pm}})^T \Phi(n)$$
$$\widehat{y'}_{i^{\pm}}(n) = \Theta^T \Gamma_{i^{\pm}}(n)$$

could be simplified into

$$y'_{i^{\pm}}(n) = \Sigma_0 + \Sigma_1(y(n-1) \pm \mu_i \phi_i) + \Sigma_2(y(n-1) \pm \mu_i \phi_i)^2$$

where:

$$\Sigma_0 = \sum_{i=0}^{L} a_i(n)x(n-i) + \sum_{j=2}^{L} b_j(n)y(n-j) + \sum_{i=0}^{L}\sum_{j=2}^{L} c_{i,j}(n)x(n-i)y(n-j) + \sum_{i=0}^{L}\sum_{k=0}^{L} d_{i,k}(n)x(n-i)x(n-k) + \sum_{j=2}^{L}\sum_{l=2}^{L} f_{j,l}y(n-j)y(n-l)$$

$$\Sigma_1 = b1 + \sum_{i=0}^{L} c_{i,1}x(n-i) + \sum_{j=2}^{L} f_{j,1}y(n-j) + \sum_{l=2}^{L} f_{1,l}y(n-j)$$

$$\Sigma_2 = f_{1,1}$$

Letting M be the maximum lag of the secondary path, the calculation of Σ_0 requires

$$\begin{aligned} multiplications &= M + 1 + M - 1 \\ &\quad +2((M-1)(M+1)) \\ &\quad +2((M+1)(M+1)) \\ &\quad +2((M-1)(M-1)) \\ &= 6L^2_{secondary} + 2M + 2 \end{aligned}$$

$$\begin{aligned} additions &= 5 + M + M - 2 \\ &\quad +(M-2)M \\ &\quad +L^2_{secondary} + (M-2)^2 \\ &= 3L^2_{secondary} - 4M + 7 \end{aligned}$$

The calculation of the vector Σ_1 requires

$$multiplications = 3M - 4$$

$$additions = 3M - 1$$

and the calculation of the value Σ_2 has a null cost. Now, for each controller's coefficient we need to calculate $\mu_i\phi_1$ which costs one multiplication, $y(n-1) \pm \mu_i\phi_i$ which costs 2 additions, $(y(n-1) \pm \mu_i\phi_i)^2$ that adds other 2 multiplications and an additional cost of 4 multiplications and 4 additions in order to obtain the final values of y'_{i^+} and y'_{i^-}. The total cost to build all the possible values of $y'_{i^\pm}$ at the current step can be summarized as:

$$multiplications = 6L^2_{secondary} + 5M - 2 + 7R_C$$

$$additions = 3L^2_{secondary} + 7M + 6 + 6R_C$$

If we consider all the calculations performed for all the remaining instructions, when we took into account a linear FIR secondary path, the total cost of the algorithm, when both the controller and the secondary path are full second order NARX is obtained as:

$$multiplications = 39L^2_{controller} + 9L^2_{secondary} + 65L + 10M + 26$$

$$additions = 42L^2_{controller} + 6L^2_{secondary} + 70L + 12M + 35$$

In the nonlinear SP case NFGLMS algorithm cannot be simplified and it has to use 2.4 and 2.2 to perform the update. So the phases involved in the calculation become:

1. computation of the controller and secondary path output $y(n)$ and $y'(n)$; it has the same number of multiplication and addition of EWU:

$$multiplications = R_C + R_{SP}$$

$$additions = R_C - 1 + R_{SP} - 1$$

2. computation of $\frac{\partial y'(k)}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta(k)}}$ from 2.4 and 2.2:

$$multiplications = M(R_C + M + 2)$$

$$additions = M(R_C + M + 3)$$

Table 6.3: Table representing the number of multiplications of NFGLMS and 2S-EWU-SSD in the case of a second order NARX SP

Method	Total Multiplications
NFGLMS	$(3L^2+5L+2)(M+2)+$
	$+4L^2_{secondary}+7M+3$
2S-EWU-SSD	$36L^2_{controller}+9L^2_{secondary}+60L+10M+24$

Table 6.4: Table representing the number of additions of NFGLMS and 2S-EWU-SSD in the case of a second order NARX SP

Method	Total Additions
NFGLMS	$(3L^2+5L+2)(M+2)+$
	$4L^2_{secondary}+8M$
2S-EWU-SSD	$42L^2_{controller}+6L^2_{secondary}+70L+12M+35$

3. update of the filter coefficients, which requires:

$$multiplications = R_C + 1$$

$$additions = R_C$$

Hence, the total number of operations equals:

$$multiplications = (3L^2+5L+2)(M+2)+ \\ +4L^2_{secondary}+7M+3$$

$$additions = (3L^2+5L+2)(M+2)+ \\ +4L^2_{secondary}+8M$$

Table 6.3 and 6.4 summarize the complexity Even in the second order NARX SP case we can distinguish the third order term in the NFGLMS complexity. Though we cannot say that in general 2S-EWU-SSD is faster than the previously cited algorithm, but it is clear that the SP impacts less on the computation time. We also have to take into consideration that we are comparing EWU with the NFGLMS that uses the Feintuch's approximation. On the other hand, if the full gradient calculation is used, the computational complexity will

surely grow. In fact all the excluded approximation recursions come back in the calculation increasing the complexity on the third order term.

6.3 Measured Execution Times

A comprehensive analysis of the execution time is carried out in this section. This kind of analysis is useful to check that the theoretical values of complexity, calculated in the previous chapter, are consistent. In particular, we will investigate single cycle execution time. If it varies between the two algorithms, the one with the lower complexity is suitable for being fed with a signal sampled at higher rates, thus increasing noise reduction performances. We adopted the following precautions:

- Store different random signals (realizations of the same standard uniform distribution on the open interval $(0, 1)$), and use them as an input, feeding different algorithms with the same data, rather than generating them on the fly. This is fundamental for the repeatability of the results and for a fair comparison;
- Limit the analysis to ten different random input signals of length 30000 samples;
- Limit the analysis to the first six tests reported in chapter 5, which are the ones without saturations;
- tic-toc commands[2] are placed in such a way that only the code that constitutes the main loop is included. This is done to collect the execution time of each cycle;
- All the execution times that are measured and stored on the hard drive. An additional script takes care of the data analysis and plotting. This should in principle allow the repeatability of all the measures reported, provided the that the same input signals are used.

[2]A Matlab command used to retrieve the time of execution of a particular code

Table 6.5: Test 1, NFGLMS cycle execution statistics

	mean	variance
1	$2.12 * 10^{-4}s$	$2.52 * 10^{-10}$
2	$2.13 * 10^{-4}s$	$2.27 * 10^{-10}$
3	$2.10 * 10^{-4}s$	$0.93 * 10^{-10}$
4	$2.12 * 10^{-4}s$	$2.12 * 10^{-10}$
5	$2.10 * 10^{-4}s$	$1.89 * 10^{-10}$
6	$2.16 * 10^{-4}s$	$5.78 * 10^{-10}$
7	$2.07 * 10^{-4}s$	$0.71 * 10^{-10}$
8	$2.18 * 10^{-4}s$	$5.72 * 10^{-10}$
9	$2.17 * 10^{-4}s$	$4.65 * 10^{-10}$
10	$2.08 * 10^{-4}s$	$0.73 * 10^{-10}$

Table 6.6: Test 1, 2S-EWU-SSD cycle execution statistics

	mean	variance
1	$1.42 * 10^{-4}s$	$4.80 * 10^{-9}$
2	$1.36 * 10^{-4}s$	$2.39 * 10^{-10}$
3	$1.37 * 10^{-4}s$	$1.64 * 10^{-10}$
4	$1.43 * 10^{-4}s$	$2.90 * 10^{-10}$
5	$1.38 * 10^{-4}s$	$1.54 * 10^{-10}$
6	$1.40 * 10^{-4}s$	$2.79 * 10^{-10}$
7	$1.39 * 10^{-4}s$	$1.80 * 10^{-10}$
8	$1.44 * 10^{-4}s$	$4.53 * 10^{-10}$
9	$1.40 * 10^{-4}s$	$2.52 * 10^{-10}$
10	$1.37 * 10^{-4}s$	$1.28 * 10^{-10}$

Table 6.7: Test 2, NFGLMS cycle execution statistics

	mean	variance
1	$3.01 * 10^{-4}s$	$2.55 * 10^{-10}$
2	$2.98 * 10^{-4}s$	$2.00 * 10^{-10}$
3	$3.02 * 10^{-4}s$	$2.53 * 10^{-10}$
4	$3.00 * 10^{-4}s$	$2.70 * 10^{-10}$
5	$2.97 * 10^{-4}s$	$1.21 * 10^{-10}$
6	$2.97 * 10^{-4}s$	$1.19 * 10^{-10}$
7	$2.99 * 10^{-4}s$	$1.82 * 10^{-10}$
8	$2.99 * 10^{-4}s$	$1.77 * 10^{-10}$
9	$2.98 * 10^{-4}s$	$2.16 * 10^{-10}$
10	$3.00 * 10^{-4}s$	$2.62 * 10^{-10}$

In tables 6.5 and 6.6 we can appreciate the results collected for the execution times of the main cycle of the first test. The mean value for both algorithms doesn't exhibit noticeable variations, when the algorithm is fed with different white noise signals. The mean values for the EWU range from $1.3668 * 10^{-04}s$ to $1.4464 * 10^{-04}s$, while the NFGLMS ones range from $2.0762 * 10^{-04}$ to $2.1875 * 10^{-04}$ seconds, which means that the EWU performs a bit better in this case. On the other hand the NFGLMS exhibits overall less variance.

In the second test (see tables 6.7, 6.8), we can note that the mean values are more or less constant along different input signals and are more or less the same for the two algorithms. In fact, the measured mean times for the EWU range from $3.3890 * 10^{-04}s$ to $3.4681 * 10^{-04}s$ while the range for the NFGLMS spans from $2.9740 * 10^{-04}s$ to $3.0208 * 10^{-04}s$. As a consequence the NFGLMS seems to perform barely better in this case. Again, the NFGLMS exhibits less variance.

The results of the third test show (see tables 6.9, 6.10) that mean and variance values of the two algorithms are similar and more or less constant when we span all the ten input signals.

Table 6.8: Test 2, 2S-EWU-SSD cycle execution statistics

	mean	variance
1	$3.43 * 10^{-4}s$	$3.66 * 10^{-10}$
2	$3.45 * 10^{-4}s$	$3.07 * 10^{-10}$
3	$3.39 * 10^{-4}s$	$1.87 * 10^{-10}$
4	$3.45 * 10^{-4}s$	$1.94 * 10^{-10}$
5	$3.41 * 10^{-4}s$	$1.97 * 10^{-10}$
6	$3.45 * 10^{-4}s$	$2.68 * 10^{-10}$
7	$3.41 * 10^{-4}s$	$1.48 * 10^{-10}$
8	$3.46 * 10^{-4}s$	$2.49 * 10^{-10}$
9	$3.42 * 10^{-4}s$	$3.08 * 10^{-10}$
10	$3.41 * 10^{-4}s$	$3.89 * 10^{-10}$

Table 6.9: Test 3, NFGLMS cycle execution statistics

	mean	variance
1	$3.35 * 10^{-4}s$	$1.71 * 10^{-10}$
2	$3.36 * 10^{-4}s$	$1.86 * 10^{-10}$
3	$3.36 * 10^{-4}s$	$3.37 * 10^{-10}$
4	$3.46 * 10^{-4}s$	$1.30 * 10^{-10}$
5	$3.47 * 10^{-4}s$	$1.71 * 10^{-10}$
6	$3.45 * 10^{-4}s$	$1.60 * 10^{-10}$
7	$3.44 * 10^{-4}s$	$1.41 * 10^{-10}$
8	$3.47 * 10^{-4}s$	$1.29 * 10^{-10}$
9	$3.46 * 10^{-4}s$	$1.45 * 10^{-10}$
10	$3.45 * 10^{-4}s$	$1.81 * 10^{-10}$

Table 6.10: Test 3, 2S-EWU-SSD cycle execution statistics

	mean	variance
1	$3.65 * 10^{-4}s$	$1.98 * 10^{-10}$
2	$3.76 * 10^{-4}s$	$2.20 * 10^{-10}$
3	$1.84 * 10^{-4}s$	$0.84 * 10^{-10}$
4	$3.81 * 10^{-4}s$	$2.20 * 10^{-10}$
5	$3.80 * 10^{-4}s$	$2.26 * 10^{-10}$
6	$3.79 * 10^{-4}s$	$3.07 * 10^{-10}$
7	$3.78 * 10^{-4}s$	$2.58 * 10^{-10}$
8	$3.82 * 10^{-4}s$	$2.77 * 10^{-10}$
9	$3.88 * 10^{-4}s$	$2.45 * 10^{-10}$
10	$3.88 * 10^{-4}s$	$2.74 * 10^{-10}$

Table 6.11: Test 4, NFGLMS cycle execution statistics

	mean	variance
1	$5.12 * 10^{-4}s$	$5.10 * 10^{-10}$
2	$5.07 * 10^{-4}s$	$2.68 * 10^{-10}$
3	$5.13 * 10^{-4}s$	$2.73 * 10^{-10}$
4	$5.15 * 10^{-4}s$	$5.51 * 10^{-10}$
5	$5.10 * 10^{-4}s$	$2.66 * 10^{-10}$
6	$5.11 * 10^{-4}s$	$4.94 * 10^{-10}$
7	$5.09 * 10^{-4}s$	$2.12 * 10^{-10}$
8	$5.09 * 10^{-4}s$	$2.26 * 10^{-10}$
9	$5.09 * 10^{-4}s$	$2.15 * 10^{-10}$
10	$5.14 * 10^{-4}s$	$3.64 * 10^{-10}$

Table 6.12: Test 4, 2S-EWU-SSD cycle execution statistics

	mean	variance
1	$4.95 * 10^{-4}s$	$5.52 * 10^{-10}$
2	$5.11 * 10^{-4}s$	$8.22 * 10^{-10}$
3	$4.85 * 10^{-4}s$	$21.3 * 10^{-10}$
4	$4.96 * 10^{-4}s$	$5.94 * 10^{-10}$
5	$4.96 * 10^{-4}s$	$11.7 * 10^{-10}$
6	$4.98 * 10^{-4}s$	$12.1 * 10^{-10}$
7	$4.94 * 10^{-4}s$	$14.8 * 10^{-10}$
8	$4.94 * 10^{-4}s$	$11.1 * 10^{-10}$
9	$4.93 * 10^{-4}s$	$11.9 * 10^{-10}$
10	$4.94 * 10^{-4}s$	$18.7 * 10^{-10}$

Tables 6.11 and 6.12 show the results of the test number four. It is useful to analyze the execution mean time, as we have done so far, for all the test cases. For the NFGLMS it ranges from $5.0742 * 10^{-04}s$ to $5.1515 * 10^{-04}$ while for the EWU the range spans the interval $4.8515 * 10^{-04}s$ - $5.1111 * 10^{-04}s$. In this case, the EWU wins by a very small margin. The EWU shows again a bigger variance in the collected measures.

The results of test five, which is the one that uses a non optimal model for the controller, are shown in tables 6.13 and 6.14 are quite similar to the ones done before. The mean values range from $3.0041 * 10^{-04}s$ to $3.1208 * 10^{-04}s$ for the NFGLMS, while for the EWU the span is $1.5748 * 10^{-04}s$ - $3.5919 * 10^{-04}s$. The variances are quite similar for both algorithms.

The sixth test (see tables 6.15 and 6.16) is the most complicated one and it is the last case that we analyze. All the other tests described in 5 are referred to saturation problems, which are not useful for the performance analysis of the algorithms, when only the time is taken into consideration as a performance measure. The mean execution times range from value $5.0832 * 10^{-04}s$ to $5.2451 * 10^{-04}s$ for the NFGLMS and from $3.0262 * 10^{-04}s$ to $4.9766 * 10^{-04}s$ for the EWU. Even if the EWU performs better from this point of view, its variances are again greater.

Table 6.13: Test 5, NFGLMS cycle execution statistics

	mean	variance
1	$3.12 * 10^{-4}s$	$1.57 * 10^{-10}$
2	$3.12 * 10^{-4}s$	$1.15 * 10^{-10}$
3	$3.08 * 10^{-4}s$	$1.09 * 10^{-10}$
4	$3.07 * 10^{-4}s$	$0.78 * 10^{-10}$
5	$3.07 * 10^{-4}s$	$0.92 * 10^{-10}$
6	$3.12 * 10^{-4}s$	$1.13 * 10^{-10}$
7	$3.04 * 10^{-4}s$	$1.90 * 10^{-10}$
8	$3.00 * 10^{-4}s$	$1.39 * 10^{-10}$
9	$3.03 * 10^{-4}s$	$1.54 * 10^{-10}$
10	$3.03 * 10^{-4}s$	$2.00 * 10^{-10}$

Table 6.14: Test 5, 2S-EWU-SSD cycle execution statistics

	mean	variance
1	$3.55 * 10^{-4}s$	$2.57 * 10^{-10}$
2	$3.59 * 10^{-4}s$	$2.20 * 10^{-10}$
3	$3.53 * 10^{-4}s$	$2.90 * 10^{-10}$
4	$3.52 * 10^{-4}s$	$1.88 * 10^{-10}$
5	$3.56 * 10^{-4}s$	$1.57 * 10^{-10}$
6	$3.45 * 10^{-4}s$	$3.26 * 10^{-10}$
7	$3.42 * 10^{-4}s$	$2.18 * 10^{-10}$
8	$1.57 * 10^{-4}s$	$0.55 * 10^{-10}$
9	$3.50 * 10^{-4}s$	$2.81 * 10^{-10}$
10	$1.63 * 10^{-4}s$	$0.98 * 10^{-10}$

Table 6.15: Test 6, NFGLMS cycle execution statistics

	mean	variance
1	$5.09 * 10^{-4}s$	$2.27 * 10^{-10}$
2	$5.09 * 10^{-4}s$	$1.95 * 10^{-10}$
3	$5.08 * 10^{-4}s$	$2.11 * 10^{-10}$
4	$5.10 * 10^{-4}s$	$2.93 * 10^{-10}$
5	$5.21 * 10^{-4}s$	$4.00 * 10^{-10}$
6	$5.17 * 10^{-4}s$	$4.05 * 10^{-10}$
7	$5.14 * 10^{-4}s$	$3.68 * 10^{-10}$
8	$5.15 * 10^{-4}s$	$3.38 * 10^{-10}$
9	$5.25 * 10^{-4}s$	$2.12 * 10^{-10}$
10	$5.22 * 10^{-4}s$	$3.16 * 10^{-10}$

Table 6.16: Test 6, 2S-EWU-SSD cycle execution statistics

	mean	variance
1	$3.73 * 10^{-4}s$	$99.1 * 10^{-10}$
2	$3.27 * 10^{-4}s$	$45.9 * 10^{-10}$
3	$3.84 * 10^{-4}s$	$97.6 * 10^{-10}$
4	$3.03 * 10^{-4}s$	$1.98 * 10^{-10}$
5	$4.96 * 10^{-4}s$	$13.3 * 10^{-10}$
6	$4.96 * 10^{-4}s$	$12.4 * 10^{-10}$
7	$4.98 * 10^{-4}s$	$10.8 * 10^{-10}$
8	$4.51 * 10^{-4}s$	$84.1 * 10^{-10}$
9	$4.45 * 10^{-4}s$	$97.0 * 10^{-10}$
10	$4.67 * 10^{-4}s$	$70.7 * 10^{-10}$

In general, we can state that the measured execution times between the two algorithms are always comparable. Moreover, they are consistent with the complexity values found in the theoretical analysis. In all the performed tests the variance of the EWU is always bigger with respect to the NFGLMS one. This is due to the various optimizations added to the EWU in order to build it's best variant called 2S-EWU-SSD. This variant, has many if cycles and therefore the execution time changes greatly depending on which condition is satisfied and in particular in which of the two stages is the algorithm. On the other hand, the NFGLMS does, at each iteration, the same type of instructions, consequently leading to more regular measures. Though the collected values are similar between the two algorithms, it is worth noticing that the main advantage of the EWU with respect to the NFGLMS resides in its fast convergence, as showed in chapter 5. Consequently, if a convergence threshold is defined in order to stop the coefficients adaptation, EWU total execution time should be less the the NFGLMS one. As a concluding remark, we can state that, looking at the mean values of each algorithm, the distance from the speaker to the reference microphone should not represent a problem in a real scenario. For example, if the worst mean value among all (NFGLMS - test 6 - input 9) is considered, the critical distance is calculated as $343[m/s] * 5.25 * 10^{-4}[s] = 0.1801[m]$, which is reasonable for the majority of the applications. Moreover, it is important to note that a DSP implementation of the algorithms should further improve the cycle execution delays measured through Matlab, hence the distance from the noise microphone to the speaker is likely to be reduced.

Chapter 7

Conclusions and Future Work

A gradient-free optimization method was employed in a general NANC setting to avoid the computational complexity and stability issues related to gradient-based approaches in the case of recursive nonlinear models. Contrary to other adaptations of SA algorithms to the ANC context previously presented in the literature, the availability of a SP estimate has been assumed, making it possible to evaluate the effect of a given parameter perturbation virtually, without affecting the controller performance and improving the convergence properties. The EWU algorithm compares favorably with competitor NANC methods, such as the NFGLMS or the VFXLMS, both in terms of estimation accuracy and convergence speed. An EWU-based model selection procedure has also been developed and, as shown in chapter 5, it finds reasonable and compact model structures. The basic version of the algorithm relies on the instantaneous measure of the error to evaluate a given controller parametrization. A possible solution, that needs to be further investigated, is to protract the evaluation for a longer time horizon, without computing at every step all the filters' outputs.

The usage of the SP estimate to recover a primary path output estimate allows, in particular, the parallel adaptation of multiple filter instances, subject to different perturbations. This approach to the problem can be used together with SA methods as PSO, SPSA and FDSA. In this way they could earn in speed of convergence and could collect all the information coming from different parametrizations without affecting the system output.

Appendix A

Notes on the Matlab Implementation of the Algorithm

A.1 Introduction

In this section we will face some of the main issues tied up with the implementation in Matlab of the algorithm and methods used.

A.2 NARX Representation

To speed up the simulation process in Matlab we build a comfortable structure for the NARX model. This structure was created to let Matlab work as much as possible with matrices. Every model is represented by a 3-dimensional matrix. Each layer of the matrix represent one regressor and is composed, for a SISO system, by 2 columns, one for the input and one for the output. The number of rows is equal to the order of the model. In every cell is stored the delay of each term of the current regressor and -1 if the cell is not used. For example, if we have the following NARX model

$$y(n) = ax(n) + bx^2(n) + cx(n-1)y(n-2) + dy^2(n-2)$$

where the order is 2 the slices representing every single regressor are:

$$\underbrace{\begin{bmatrix} 0 & -1 \\ -1 & -1 \end{bmatrix}}_{\text{regressor 1}} \underbrace{\begin{bmatrix} 0 & -1 \\ 0 & -1 \end{bmatrix}}_{\text{regressor 2}} \underbrace{\begin{bmatrix} 1 & 2 \\ -1 & -1 \end{bmatrix}}_{\text{regressor 3}} \underbrace{\begin{bmatrix} -1 & 2 \\ -1 & 2 \end{bmatrix}}_{\text{regressor 4}}$$

A.3 Gradient Implementation

The computation of the gradient is one of the main issues of NFGLMS. In fact its implementation represents the most of the computational complexity. Following the data structure described in the previous section, and the recursive method described in 2.3.2, we came up with the following procedure:

for $i = 0 \rightarrow$M = memory of the SP model **do**
  for $j = 0 \rightarrow$ns = number of SP coefficients **do**
    out(i) $\leftarrow$ *out(i) + product of all the terms of the SP regressor but that with the delay equals to i;*
  end for
end for
for $i = 0 \rightarrow$nc = number of the controller coefficients **do**
  $yTheta(k, i) \leftarrow$*product of all the terms of the controller regressor;*
end for
$grad(k) \leftarrow$ *scalarProduct(out,yTheta);*

Bibliography

[1] Luis Antonio Aguirre and Stephen A. Billings. Dynamical effects of overparametrization in nonlinear models. *Physica D*, 8(1-2):26–40, January 1995.

[2] S. A. Billings, S. Chen, and M. J. Korenberg. Identification of mimo nonlinear systems using a forward-regression orthogonal estimator. *International Journal of Control*, 49:2157–2189, 1989.

[3] Elias Bjarnason. Analysis of the filtered-x lms algorithm. *IEEE Transactions on Speech and Audio Processing*, 3(6):504–514, November 1995.

[4] C. Cantelmo and L. Piroddi. Adaptive model selection for polynomial narx models. *IET Control Theory Appl.*, 4(12):2693–2706, 2010.

[5] Debi Prasad Das and Ganapati Panda. Active mitigation of nonlinear noise processes using a novel filtered-s lms algorithm. *IEEE Transactions on Speech and Audio Processing*, 12(3):313–322, May 2004.

[6] Debi Prasad Das, S. R. Mohopatra, A. Routray, and T. K. Basu. Filtered-s lms algorithm for multichannel active control of nonlinear noise processes. *IEEE TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING*, 14(5):1875–1880, September 2006.

[7] Debi Prasad Das, Ganapati Panda, and K. D. Nayak. Development of frequency domain block filtered-s lms (fbfslms) algorithm for active noise control system. *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, pages 289–292, 2006.

[8] F. Alton Everest. *The Master Handbook of Acoustics*. McGraw-Hill, 2001.

[9] Kiefler and Wolfowitz. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, 23:462–466, 1951.

[10] Sen Maw Kuo and Dennis R. Morgan. *Active Noise Control Systems - Algorithms and DPS Implementations*. Wiley-Interscience, 1996.

[11] Sen Maw Kuo and Hsien-Tsai Wu. Nonlinear adaptive bilinear filters for active noise control systems. *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS*, 52(3):617– 624, March 2005.

[12] Sen Maw Kuo, Hsien-Tsai Wu, Fu-Kun Chen, and Madhu R. Gunnala. Saturation effects in active noise control systems. *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS*, 51(6):1163– 1171, June 2004.

[13] I. J. Leontaritis and S. A. Billings. Input-output parametric models for nonlinear systems—part ii: Stochastic nonlinear systems. *International Journal of Control*, pages 329–344, 1985.

[14] K. Li, J.-X. Peng, and G. W. Irwin. A fast nonlinear model identification method. *IEEE Trans. Autom. Control*, 50(8):1211–1216, 2005.

[15] Yutaka MAEDA and Takao YOSHIDA. An active noise control without estimation of secondary-path an active noise control without estimation of secondary-path–anc using simultaneous perturbation. *Active99*, 1999.

[16] Tetsuya Matsuura, Takehiko Hiei, Hiroyuki Itoh, and Kanikazu Torikoshi. Active noise control by using prediction of time series with a neural network. *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, 3:2070–2075, 1995.

[17] Roberto Napoli. Nonlinear active noise control with narx models. Master's thesis, Politecnico di Milano, 2006-2007.

[18] Roberto Napoli and Luigi Piroddi. Nonlinear active noise control with narx models. *IEEE TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING*, 18(2):286–295, February 2010.

[19] R.E. Perez and K. Behdinann. Particle swarm approach for structural design optimization. *Computers and Structures*, pages 1579–1588, 2007.

[20] Luigi Piroddi and W. Spinelli. An identification algorithm for polynomial narx models based on simulation error minimization. *International Journal of Control*, 76(17):1767–1781, 2003.

[21] Nirmal Kumar Rout, Debi Prasad Das, and Ganapati Panda. Particle swarm optimization based active noise particle swarm optimization based active noise particle swarm optimization based active noise control algorithm without secondary path identification. *IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT*, 61(02), February 2012.

[22] G. L. Sicuranza and A. Carini. Piecewise-linear expansions for nonlinear active noise control. *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, pages 209–212, 2006.

[23] S. D. Snyder and N. Tanaka. Active control of vibration using a neural network. *IEEE Transactions On Neural Networks*, 6(4):819–828, July 1995.

[24] James C. Spall. An overview of the simultaneous perturbation method for efficient optimization. *An Overview of the Simultaneous Perturbation Method for Efficient Optimization*, 19(4):482–492, 1998.

[25] Paul Strauch and Bernard Mulgrew. Nonlinear active noise control in a liner duct. *IEEE TRANSACTIONS ON SIGNAL PROCESSING*, 46(9): 2404–2412, September 1998.

[26] Li Tan and Jean Jiang. Adaptive volterra filters for active control of nonlinear noise processes. *IEEE TRANSACTIONS ON SIGNAL PROCESSING*, 49(8):1667–1676, August 2001.

[27] K.S. Tang, K.F. Man, S. Kwong, and Q. He. Genetic algorithms and their applications. *IEEE SIGNAL PROCESSING MAGAZINE*, pages 22–37, November 1996.

[28] Jerome Friedman Trevor Hastie, Robert Tibshirani. *The Elements of Statistical Learning*. 2 edition, 2008.

[29] Bernard Widrow, John R. Glover, and John M. McCool. Adaptive noise cancelling: Principles and applications. *Proceedings of the IEEE*, 63(12): 1692–1716, 1975.

[30] Dayong Zhou and Victor DeBrunner. Efficient adaptive nonlinear filters for nonlinear active noise control. *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS*, 54(3):669–681, March 2007.

[31] Ya-Li Zhou, Qi-Zhi Zhang, Xiao-Dong Li, and Woon-Seng Gan. On the use of an spsa-based model-free feedback controller in active noise control for periodic disturbances in a duct. *JOURNAL OF SOUND AND VIBRATION*, May 2008.

List of Figures