Master Thesis

Pre-Distortion Algorithms
Implemented in
Fixed-Point Arithmetic

performed for the purpose of obtaining the academic degree a graduate engineer
by Universidad Politécnica de Cartagena

under the direction of
Univ.Prof. Dipl.-Ing. Dr.techn. Markus Rupp
Projektass. Dipl.-Ing. Robert Dallinger
Institute of Telecommunications

submitted at the Vienna University of Technology
Faculty of Electrical Engineering and Information Technology

by
Beatriz Carcelén Flores

Vienna, June 2014
I hereby certify that the work reported in this thesis is my own, and the work done by other authors is appropriately cited.

Beatriz Carcelén Flores
Vienna, June, 2014
Abstract

Pre-Distortion Algorithms Implemented in Fixed-Point Arithmetic

by Beatriz Carcelén Flores

Nowadays, wireless communications systems are required to provide high data-rates with high quality. In order to achieve this, spectrally efficient transmission techniques are employed which rely on signals with large envelope fluctuations. Moreover, due to power efficiency demands power amplifiers have to work close to their saturation region. Unfortunately, their resulting nonlinear behaviour introduces nonlinear distortions. By this, on the one hand the transmitted signal is degraded, on the other hand, it causes spectral widening beyond the channel bandwidth, and consequently interference with neighbouring transmission channels.

Digital pre-distortion is a technique used to compensate the distortions introduced by the power amplifier, so that the overall system operates as a linear yet efficient amplifying stage. This solution reduces the transmission unit size and allows for cutting energy costs, especially if combined with other linearization techniques. As the pre-distorter has to predict the nonlinearity introduced by the power amplifier, pre-distortion can be considered a behavioural modeling problem.

In this thesis, we consider several pre-distortion schemes found in literature that are based on behavioural modeling. Starting with the memoryless polynomial model, we move on to the general but computationally expensive truncated Volterra series and, finally end up with the decomposed piecewise Volterra series proposed by Zhu in [1] that allow to reduce the computational complexity by selectively pruning of the truncated Volterra series. The main goal of this work is to evaluate the fixed-point implementation of the algorithms. In order to do so the algorithms are implemented in MATLAB in fixed-point arithmetic, as well as in floating-point arithmetic; where the latter is used as reference for a comparison of performance. In addition, a detailed review of the theory is presented in this work. The algorithms are evaluated with a nonlinear reference model: a saleh model for the memoryless case and a hammerstein model for the memory cases. Simulation results show that the decomposed piecewise Volterra model employing the dynamic deviation reduction-based Volterra model as sub-model outperforms the traditional models.
Acknowledgements

Before beginning with the work itself, it is time to be grateful to the people who contributed to this thesis in one or many ways.

First, I would like to thank my advisors Dr. Markus Rupp and Dipl. Ing. Robert Dallinger for giving me the opportunity to work with them and accomplish my thesis at Technical University of Wien. Their support, advice and supervision has been fundamental for the development of this thesis. I feel very lucky to meet such inspiring and talented people. This thesis has been developed with the resources and support of the Institute of Telecommunications of the Faculty of Electrical Engineering and Information Technology.

I would like to thank my hometown university, Universidad Politécnica de Cartagena, where I studied my degree and María Victoria Bueno Delgado, my Erasmus advisor, whose help was essential.

I am grateful to the people I met in Vienna, who became my family when I was far from my loved ones. Thank you for your friendship and constant support, specially Alex, Raquel, Laura and Gerardo.

I would like to thank also my fellow students and friends, because we have never walked alone through these years. We have shared a lot of moments including difficult times, now I want to share this happy moment with you.

My thanks to my best friends Paola and Nora, for believing in me when I did not, for not letting me to give up.

Finally, I would like to thank my family. My parents, Francisco and Adela, for their unconditional love and encouragement. And last but not least, thanks to my brother Álvaro for his help and for adding the touch of humour that I needed.
Contents

Abstract ii

Acknowledgements iii

List of Figures vi

List of Tables viii

Abbreviations ix

Symbols x

1 Introduction 1

1.1 Motivation ........................................ 1

1.1.1 Digital Pre-distortion .......................... 3

1.2 Outline ........................................... 4

2 Fixed-point Implementation on MATLAB 7

2.1 Basics ............................................ 7

2.2 Time Performance ................................. 9

2.2.1 Timing Improvement ............................. 12

2.2.2 Least Mean Squares Algorithm ................. 12

2.3 Word Length Selection ............................. 18

2.3.1 Recursive Least Squares Algorithm .......... 18

2.3.2 Selection of the Overall Word Length ........ 21

2.3.3 Selection of the Fraction Length ............... 24

3 Behavioural Modeling of Power Amplifiers 26

3.1 Memoryless Polynomial Model .................... 27

3.1.1 Parameter Estimation .......................... 28

3.2 Models with Memory ............................... 29

3.2.1 Memory Polynomial ............................. 29

3.2.1.1 Parameter Estimation ....................... 30

3.2.2 Model Based on Truncated Volterra Series .. 31

3.2.2.1 Properties of Volterra Series ............... 32

3.2.2.2 Baseband Representation of a Volterra System 34
## Contents

3.2.2.3 The Discrete-Time Volterra Model . . . . . . . . . . . . . 34
3.2.2.4 Parameter Estimation . . . . . . . . . . . . . . . . . . . . 37
3.2.3 Dynamic Deviation Reduction-Based Volterra Series Model . . . 39
3.2.3.1 Parameter Estimation . . . . . . . . . . . . . . . . . . . . 41
3.2.4 Decomposed Piecewise Volterra Series Model . . . . . . . . . . 42
3.2.4.1 Parameter Estimation . . . . . . . . . . . . . . . . . . . . 46

### 4 Results

4.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48
4.2 Adaptive Estimation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48
4.3 Reference Models . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 49
4.3.1 Memoryless Configuration . . . . . . . . . . . . . . . . . . . . . . . . . . 50
4.3.2 Hammerstein Configuration . . . . . . . . . . . . . . . . . . . . . . . . . 50
4.4 Memoryless Polynomial . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 51
4.5 Memory Polynomial . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54
4.6 Truncated Volterra Series . . . . . . . . . . . . . . . . . . . . . . . . . . . . 57
4.6.1 Baseband Volterra model . . . . . . . . . . . . . . . . . . . . . . . . . . 59
4.7 Dynamic Deviation Reduction-based Volterra Series . . . . . . . . . . . 62
4.8 Decomposed Piecewise Volterra Series Model . . . . . . . . . . . . . . . 65

### 5 Conclusions and Outlook

5.1 Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 68
5.2 Suggestions for Future Work . . . . . . . . . . . . . . . . . . . . . . . . . . 69

Bibliography 70
List of Figures

1.1 Characterization AM/AM and AM/PM .................................................. 2
1.2 Block diagram of the system ............................................................... 3
1.3 Desired behaviour .............................................................................. 4
1.4 Adaptive Scheme .............................................................................. 5
2.1 Time performance of indexing in fixed-point .......................................... 11
2.2 Time dependence with vector length .................................................... 11
2.3 Time performance for the LMS without timing improvement .................. 14
2.4 Time performance for LMS with optimization ....................................... 15
2.5 Results for LMS without timing improvement ....................................... 16
2.6 Results for LMS with timing improvement .......................................... 16
2.7 Results for word length selection ....................................................... 23
2.8 Results for the selection of the fraction length ..................................... 25
3.1 Vector thresholds decomposition zones ............................................... 43
3.2 Signal decomposition ........................................................................ 45
3.3 Schematic overview of the signal processing line ................................ 46
4.1 Characterization AM/AM and AM/PM of a Hammerstein model .......... 51
4.2 Parameter extraction of Memoryless polynomial ................................... 52
4.3 AM/AM and AM/PM conversion for memoryless polynomial adaptation ... 53
4.4 Error performance for memoryless polynomial adaptation ...................... 53
4.5 Parameter extraction of Memory polynomial ........................................ 55
4.6 AM/AM and AM/PM conversion for memory polynomial adaptation ....... 56
4.7 Error performance for memory polynomial adaptation .......................... 56
4.8 Parameter extraction of Passband Volterra Series .................................. 57
4.9 [AM/AM and AM/PM conversion for passband truncated Volterra series model ........................................................................ 58
4.10 Error performance for passband truncated Volterra series adaptation ....... 59
4.11 Parameter extraction of baseband Volterra Series .................................. 60
4.12 [AM/AM and AM/PM conversion for baseband truncated Volterra series model ........................................................................ 61
4.13 Error performance for baseband truncated Volterra series adaptation ...... 61
4.14 Parameter extraction of DDR model ................................................... 62
4.15 [AM/AM and AM/PM conversion for DDR model ................................. 63
4.16 Error performance for DDR Volterra Series adaptation ................................ 64
4.17 Parameter extraction of decomposed piecewise Volterra Series ............... 65
4.18 [AM/AM and AM/PM conversion for decomposed piecewise Volterra Se- ries model ........................................................................ 66
4.19 Error performance for decomposed piecewise Volterra Series adaptation . 67
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Data of indexing time dependence</td>
<td>12</td>
</tr>
<tr>
<td>2.2</td>
<td>Corresponding symbols on MATLAB</td>
<td>14</td>
</tr>
<tr>
<td>2.3</td>
<td>Data of RLS simulation</td>
<td>21</td>
</tr>
<tr>
<td>2.4</td>
<td>Fraction lengths for the word length selection</td>
<td>21</td>
</tr>
<tr>
<td>2.5</td>
<td>Corresponding symbols on MATLAB</td>
<td>22</td>
</tr>
<tr>
<td>4.1</td>
<td>Optimum Parameters for Saleh Model</td>
<td>50</td>
</tr>
<tr>
<td>4.2</td>
<td>Fi object settings</td>
<td>52</td>
</tr>
<tr>
<td>4.3</td>
<td>Data results for Memoryless polynomial</td>
<td>54</td>
</tr>
<tr>
<td>4.4</td>
<td>Configuration of the RRC filter</td>
<td>55</td>
</tr>
<tr>
<td>4.5</td>
<td>Data results for Memory polynomial</td>
<td>56</td>
</tr>
<tr>
<td>4.6</td>
<td>Data results for passband truncated Volterra Series</td>
<td>59</td>
</tr>
<tr>
<td>4.7</td>
<td>Data results for baseband truncated Volterra series</td>
<td>61</td>
</tr>
<tr>
<td>4.8</td>
<td>Data results for DDR model</td>
<td>64</td>
</tr>
<tr>
<td>4.9</td>
<td>Data results for decomposed piecewise Volterra Series</td>
<td>67</td>
</tr>
</tbody>
</table>
Abbreviations

AM/AM  Amplitude Modulation / Amplitude Modulation
AM/PM  Amplitude Modulation / Phase Modulation
BER    Bit Error Rate
BLAS   Basic Linear Algebra Subprograms
CDMA   Code Division Multiple Access
DC     Direct Current
DFT    Discrete Fourier Transform
DPD    Digital Pre-Distortion
EDGE   Enhanced Data for Global system for mobile Evolution
FIR    Finite Impulse Response
LAPACK Linear Algebra Package
LMS    Least Mean Squares
LS     Least Squares
LTI    Linear Time Invariant
MSE    Mean Square Error
OFDM   Orthogonal Frequency Division Multiplexing
PA     Power Amplifier
PAPR   Peak-to-Average Power Ratio
QAM    Quadrature Amplitude Modulation
QPSK   Quadrature Phase Shift Keying
RF     Radio Frequency
RLS    Recursive Least Squares
RRC    Root Raised Cosine
WCDMA  Wideband Code Division Multiple Access
Symbols

\((\cdot)^H\) Hermitian (Complex conjugate transpose of a vector/matrix)
\((\cdot)^*\) Conjugate of a vector/matrix
\((\cdot)^T\) Transpose of a vector/matrix
\(E\{\cdot\}\) Expectation operator
"The important thing in science is not so much to obtain new facts as to discover new ways of thinking about them." ~ William Bragg
Chapter 1

Introduction

1.1 Motivation

In wireless communication systems, power amplifiers (PAs) are an essential part of the system. The signal has to be amplified with high fidelity before being transmitted, so it can manage propagation in attenuating media.

The wireless industry is continuously growing, which involves a need of higher data rates to service a larger number of users with a limited availability of the radio-frequency (RF) spectrum. In order to achieve these requirements, spectrally efficient techniques such as Quadrature Phase Shift Keying (QPSK), Quadrature Amplitude Modulation (QAM) and Orthogonal Frequency Division Multiplexing (OFDM) are used. As a result of these techniques, the modulated signals have nonconstant envelopes (excepting QPSK) and high peak-to-average power ratio (PAPR), which stimulate the PA’s nonlinearities. PAs are the main source of nonlinear behaviour in a communication system. For power efficiency requirements, PAs are operating close to saturation. Signals whose envelopes are fluctuating inside that nonlinear region are harshly distorted depending on how extreme those variations are and this distortion is unfavourable for those signal whose amplitude carries information [2].

The nonlinearity causes spectral widening of the signal bandwidth by which the nominal channel bandwidth is exceeded, which produces interference between channels. It also leads to in-band distortion, which causes both a larger bit error rate (BER) and intersymbol interference [3]. Consequently, the design of power amplifiers entails a critical trade-off between power efficiency and linearity. The power efficiency of a PA can be defined as its ability to convert the DC power of the supply in output power. The power that is not converted into useful signal is dissipated by heat. Thus, PAs with
low efficiency have high levels of heat dissipation. As it was mentioned above, working close to this point leads to an unavoidable distortion. Also noteworthy that increasing the bandwidth of the transmission signal will make memory effects more predominant so that they will weaken the signal with additional distortion [4].

For the purpose of not degrading the performance of the whole system and ensuring signal integrity, the PA stage is required to behave as linear as possible. It means that the output signal \( y(t) \) and the input signal \( x(t) \) should ideally be related by a positive gain \( g \), according to

\[
y(t) = g \times x(t). \tag{1.1}
\]

As explained above, the behaviour when the PA is operating next to saturation becomes nonlinear and the gain starts being dependent on the input. Conventionally, memoryless nonlinearities can be characterized by the AM/AM and AM/PM responses of the PA, in which output amplitude and phase offset are given as functions of the current input amplitude [5]. In Fig 1.1, an example of this characterization is shown.

![Graph showing the transfer function of a Saleh Model](image)

**Figure 1.1:** An example of the characterization of a PA by AM/AM and AM/PM responses. The plots are generated with a Saleh Model (see Sec. 4.3.1) with the following parameters: \( \alpha_A = 1.9638 \), \( \alpha_\phi = 2.5293 \), \( \beta_A = 0.9945 \) and \( \beta_\phi = 2.8168 \)

One of the simplest techniques used for linearization is backing off the PA such that it is operated in the linear region of its operation curve. For signals with high PAPR the PA has to be backed off far from the saturation point, which turns into a low power
efficiency and therefore in a high heat dissipation. Thus, cooling costs are increased and more supply power is required since more amplifying stages are needed to achieve a specific output power level [6] [7].

Feed-forward is another linearization technique. It is based on the subtraction of the distortion from the output signal. This approach utilizes two circuits: an input signal cancellation circuit and an error cancellation circuit. The first circuit eliminates the input reference from the output of the PA, isolating the distortion component. The second stage suppresses the distortion component of the PA output, thereby leaving in theory only the linear amplified component of the output signal. Unfortunately, this also involves an increase in cost, a need of an extra power amplifier and an accurate calibration and synchronization of both circuits [8].

1.1.1 Digital Pre-distortion

Digital pre-distortion (DPD) is currently one of the preferred techniques for linearization as it allows several algorithms for adaptation and it is used for applications with up to 100 MHz bandwidth [9][10]. It consists of applying an inverse nonlinearity to the input signal before the PA in order to compensate the distortion introduced by the PA, so that the whole system behaves as a linear amplifying stage. In Fig 1.2, a block diagram of this ideally linear system is shown. An advantage of DPD is that it reduces size and cost in comparison with the other techniques. In addition, it does not need the extra high power error amplifier required in feed-forward solution [7].

![Figure 1.2: Block diagram of digital predistortion (DPD) based linear power amplifier (PA).](image)

In [6], it is estimated that Wideband Code Division Multiple Access (WCDMA) power amplifiers have an efficiency of 3-5%, whereas feed-forward can increase the efficiency to 6-8%. Nevertheless, DPD can achieve efficiencies between 8% and 10%. Even though the improvement could seem small, this difference of percentage will save 100 W/transmitter (assuming a 30 W mean output power from each PA). In more recent works, such as [11], a memoryless DPD applied to an enhanced data for global system for mobile evolution (EDGE) handset the efficiency is improved from 15,2% to 23,4%.

The desired behaviour of the amplifying system can be observed in Fig 1.3. The dashed lines represent a linear gain $g$ according to Eq. (1.1) and a constant phase offset. In
practical cases a zero phase offset cannot be achieved due to delays of the system, but a constant one can be considered as desired AM/PM characteristic.

DPD can be considered as a behavioural modeling problem since the nonlinearity of the PA has to be predicted. Afterwards, this nonlinearity is reversed in order to obtain the predistortion function. This prediction of the nonlinearity is achieved through two stages: modeling and identifying. It is necessary to choose an adequate mathematical model that suits the PA best. This selected formulation requires an adaptive learning algorithm to be trained and thereby reaching the optimal estimated parameters of the model as it can be observed in Fig 1.4. The key advantage of DPD is the fact that a strong knowledge about PA physics and functionality is not imperative [9].

1.2 Outline

The topic of this thesis is modeling the nonlinearities of the PA through diverse formulations. These models and algorithms will be implemented both in floating point and fixed-point arithmetic. For this purpose, MATLAB software is used. This thesis is organized as follows.

Chapter 1 - Introduction explores the need of a highly linear PA stage for spectrally efficient modulation schemes. Moreover, it describes the operating principle of Digital
Chapter 1. Introduction

pre-distortion and compares it with well-known linearization techniques such as feed-forward and backing-off.

Chapter 2 - Fixed-point Implementation in MATLAB reviews the documentation of the Toolbox used for the fixed-point implementation of the algorithms. It describes the main functions and properties employed throughout the work. Moreover, the key issue of time performance is comprised in this chapter. The proposal to optimize the simulation time is evaluated by means of the Least Mean Squares (LMS) algorithm. At the end of the chapter, the process of choosing the proper word length is presented and Recursive Least Squares (RLS) is applied to illustrate the topic.

In Chapter 3 - Behavioural Modeling of Power Amplifiers, we study the formulations proposed for modeling the behaviour of PAs. These models include both, memoryless models and models with memory. The Volterra Series are presented as the most general model for describing PAs. Different ways to prune this model lead to new models. Moreover, two models more complex and advanced are included in this chapter. The Least Squares (LS) estimator is adapted for each model in order to obtain the estimated parameters of the model.

Chapter 4 - Results presents the adaptive estimation algorithms implemented for this thesis. This chapter also explores the performance of the models proposed in Chapter 3.
through MATLAB simulations.

Finally, Chapter 5 - Conclusions and Outlook concludes this thesis and discusses future lines of research.
Chapter 2

Fixed-point Implementation on MATLAB

In this work, simulations are performed with the algorithms implemented in fixed-point arithmetics in order to evaluate their behaviour in a real world application. MATLAB provides a Fixed-Point Toolbox\(^1\) which allows to design algorithms using fixed-point data types and arithmetic. The fixed-point code can be reused in Simulink facilitating bit-true simulation, implementation and analysis. Moreover, the toolbox enables the generation of test sequences for fixed-point software and hardware verification [12].

2.1 Basics

Fixed-point numbers are a real data type representation that is characterized by its word size expressed in bits, its binary point and its signedness. A fi object is the way that MATLAB enables us to represent fixed-point data. Each fi object is defined by its numerictype properties, such as word length, fraction length, and signedness. A fi object has also attached a fimath object which defines its overflow, rounding and arithmetic properties. When representing a value in fixed-point it may occur that this value is greater in magnitude than the given fixed-point object can represent, this situation is called overflow. The ideal situation is to avoid overflows by selecting the correct data types in advance, however, if we predict that overflows may happen in our algorithm, we can set the action to take in case that overflows are detected.

The list below shows the fimath properties that are used in this thesis:

\(^1\)The version of software that is used is MATLAB R2012b
• **OverflowMode**: Action to take on overflow. There are two options: *Saturate* and *Wrap*. The first one saturates to maximum/minimum value of the range. Second one acts like two’s complement overflow.

• **RoundMode**: *Ceiling* rounds toward positive infinity, *Zero* toward zero, *Floor* toward negative infinity, *Nearest* and *Round* toward the nearest representable value. *Nearest* with ties round toward positive infinity, i.e., $1.5 \rightarrow 2 \& -1.5 \rightarrow -1$, and *Round* with ties toward negative infinity in case of negative numbers and toward positive infinity for positive numbers, i.e., $1.5 \rightarrow 2 \& -1.5 \rightarrow -2$.

• **ProductMode**: Defines how the product data type is determined. If we dont specify this property, the data type of the product is set to *Full Precision*, i.e., the word length and fraction length of the product are equal to the sum of the word lengths and fraction lengths, respectively, of both multiplicands. In our work, we need to be consistent, so we cannot have several word lengths in the same simulation. Hence, this property must be set to *Specify Precision* in this thesis. This option permits to define the data type of the product result. The word length and fraction length are determined by the properties *ProductWordLength* and *ProductFractionLength*, respectively. Using this option avoids the utilization of *quantize* (which has no direct translation into hardware). As the focus of the thesis is to study the algorithms implemented in fixed-point arithmetics, these algorithms are composed of basic operations which can be performed by an arithmetic logic unit (ALU).

• **ProductFractionLength**: Fraction length, in bits, of the product data type.

• **ProductWordLength**: Word length, in bits, of the product data type.

• **SumMode**: Defines how the sum data type is determined. As in *ProductMode*, it is possible to define the lengths of the data type with the option *Specify Precision*.

• **SumFractionLength**: Fraction length, in bits, of the sum data type.

• **SumWordLength**: Word length, in bits, of the sum data type.

The following settings will be used in the whole thesis for the fi objects (where $\mathit{wl}$ and $\mathit{fl}$, the word length and the fraction length selected for the algorithm):

```matlab
fimath(’RoundMode’,’nearest’,’OverflowMode’,’wrap’,’SumWordLength’,wl,’SumFractionLength’,fl,’SumMode’,’SpecifyPrecision’,’ProductWordLength’,wl,’ProductFractionLength’,fl,’ProductMode’,’SpecifyPrecision’);
```

In fixed-point arithmetic, addition, subtraction and multiplication can be performed normally (always choosing a correct wordlength and fractionlength in order not to overflow.
the variable). However, it is necessary to pay special attention in divisions. They are not as straightforward as the other operations. As it is not possible to define the data type of the division result, the function $\text{divide}(T,a,b)$ provided by the Fixed-Point Toolbox is needed. This function performs the division of $a$ by $b$ and $T$ is the numerictype of its result. Hence, avoiding divisions of fixed-point objects is highly recommended.

2.2 Time Performance

Time is a key problem in fixed-point implementation in MATLAB. In floating point mode, MATLAB uses BLAS (Basic Linear Algebra Subprograms) libraries, which are highly processor optimized and efficient routines that provide standard blocks for performing basic linear algebra operations with vectors and matrices [13]. Since 2000, MATLAB includes Linear Algebra Package (LAPACK) [14][15], whose routines are written so that as much as possible of the computation is performed by calls to BLAS. Operations that use these libraries are faster than a good C/C++ implementation. However, MATLAB does not use BLAS libraries in fixed-point mode. For such operations, MATLAB emulates a fixed-point processor and pays attention on scaling, rounding modes and overflows. For these reasons, fixed-point implementations in MATLAB are slower compared to the ones implemented in floating point.

There is a possibility of code acceleration in MATLAB for fixed-point arithmetic: $\text{fiaccel}$. This function translates MATLAB code to a MEX function, which accelerates fixed-point code. Nevertheless, $\text{fiaccel}$ requires conditions that our code cannot accomplish, such as the exponent of $\text{mpower}$ must be constant, i.e., its value must be known at compile time. Therefore, this option can be discarded and code optimization will be used as solution instead.

In addition, indexing fixed-point vectors and matrices is quite inefficient. The time required to reach the desired position increases with the length/size of the vector/matrix. This can be observed in the following example:

```matlab
% Fi object settings: The word length and fraction length of fixed point
% words are set as parameters.
wordlength=32;
fractionlength=30;
F = fimath('RoundMode','nearest','OverflowMode','wrap','SumWordLength',
          wordlength,'SumFractionLength',fractionlength,'SumMode','SpecifyPrecision',
          'ProductWordLength',wordlength,'ProductFractionLength',fractionlength,'ProductMode','SpecifyPrecision');
```
%Three vectors of different sizes are created and initialized randomly - the values are not important for the example. Vectors are created both in floating point and in fixed point.

%Sizes of the vectors.
M = 10^4;
M2 = 10^5;
M3 = 10^6;

%Creation of the vectors.
x = rand(M,1) + (i*rand(M,1));
x_fp = fi(x,1,wordlength,fractionlength,F);
x2 = rand(M2,1) + (i*rand(M2,1));
x2_fp = fi(x2,1,wordlength,fractionlength,F);
x3 = rand(M3,1) + (i*rand(M3,1));
x3_fp = fi(x3,1,wordlength,fractionlength,F);

m = 4; % Number of samples that we are going to take in each iteration.
N = 1000; % Number of iterations

%Creating and Initializing the matrixes.
X = zeros(m,N);
X2 = zeros(m,N);
X3 = zeros(m,N);

X_fp = fi(X,1,wordlength,fractionlength,F);
X2_fp = fi(X2,1,wordlength,fractionlength,F);
X3_fp = fi(X3,1,wordlength,fractionlength,F);

%In each iteration the matrices (X,X2,...) are filled with m samples, taken from the vector (x,x2...).
for jj = 1:N
X(1:m,jj) = x(jj:jj+m-1);
X_fp(1:m,jj) = x_fp(jj:jj+m-1);
X2(1:m,jj) = x2(jj:jj+m-1);
X2_fp(1:m,jj) = x2_fp(jj:jj+m-1);
X3(1:m,jj) = x3(jj:jj+m-1);
X3_fp(1:m,jj) = x3_fp(jj:jj+m-1);
end

In this code, the different matrices are being filled with the values of the random vectors. The same number of samples is being extracted from each vector, the only difference between these assignments is the length of the vectors. Fig 2.1 shows the results of MATLAB time profiler.
Chapter 2. Fixed-point Implementation on MATLAB

Figure 2.1: This figure shows how the time was spent in the execution of the code presented above.

As it can be observed, the execution time increases with the length and its dependence is exponential. Another simulation for vector with lengths from 1000 to 10^7 is run in order to obtain a graph of time dependence. This is shown in Fig 2.2.

<table>
<thead>
<tr>
<th>Line Number</th>
<th>Code</th>
<th>Calls</th>
<th>Total Time</th>
<th>% Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>x3_{fp}(1:m,jj) = x3_{fp}(jj;jj+m-2) ...</td>
<td>1000</td>
<td>8.322 s</td>
<td>77.3%</td>
</tr>
<tr>
<td>52</td>
<td>x2_{fp}(1:m,jj) = x2_{fp}(jj;jj+m-2) ...</td>
<td>1000</td>
<td>1.060 s</td>
<td>9.8%</td>
</tr>
<tr>
<td>33</td>
<td>x3_{fp} = fi(x3,1,wordlength,frac...)</td>
<td>1</td>
<td>0.601 s</td>
<td>5.6%</td>
</tr>
<tr>
<td>50</td>
<td>x_{fp}(1:m,jj) = x_{fp}(jj;jj+m-2) ...</td>
<td>1000</td>
<td>0.361 s</td>
<td>3.3%</td>
</tr>
<tr>
<td>32</td>
<td>x3 = rand(M3,1) + (i*rand(M3,1)) ...</td>
<td>1</td>
<td>0.081 s</td>
<td>0.8%</td>
</tr>
<tr>
<td>All other lines</td>
<td></td>
<td></td>
<td>0.346 s</td>
<td>3.2%</td>
</tr>
<tr>
<td>Totals</td>
<td></td>
<td></td>
<td>10.770 s</td>
<td>100%</td>
</tr>
</tbody>
</table>

Figure 2.2: Total time of the assignment after 1000 calls for vectors with lengths from 10^3 to 10^7. In Table 2.1 the data of the graphic is shown.
Table 2.1: Results from the simulation of the example above. With a total of 1000 calls and M from 1000 to 1000000.

<table>
<thead>
<tr>
<th>Vector length</th>
<th>$10^3$</th>
<th>$10^4$</th>
<th>$10^5$</th>
<th>$10^6$</th>
<th>$10^7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total time (s)</td>
<td>0.5841</td>
<td>0.2184</td>
<td>0.8040</td>
<td>8.0894</td>
<td>83.2738</td>
</tr>
</tbody>
</table>

2.2.1 Timing Improvement

As vectors of length in the order greater than $10^6$ will be needed for reaching the steady state of the later presented algorithms, a solution for that problem is required. Although MATLAB philosophy is to avoid nested loops to improve the code, in fixed-point it seems to be the solution. As modifications of Least Means Squares (LMS) algorithms will be used for the identifying stage, the improvement will be explained using the LMS as example.

2.2.2 Least Mean Squares Algorithm

The LMS algorithm is part of the family of stochastic gradient algorithms. The key features of this algorithm are its simplicity and low computational complexity. It has been applied to many linear filtering problems because of the fact that it requires neither matrix inversion nor estimations of correlation functions [16]. LMS approximates the Steepest Descent method for the adaptive estimation of the parameters. The operating principle of the latter is starting with an arbitrary initial guess of the solution ($\hat{w}_0$) and updates the guess towards the optimal solution iteratively [17], according to

$$\{\text{New estimate}\} = \{\text{Old estimate}\} + \{\text{Correction term}\}.$$  

Considering the input signal observation $x$ and the desired signal $d$, the standard update equation of Steepest Descent is given by

$$\hat{w}_k = \hat{w}_{k-1} + \mu_k [r_{xd} - R_{xx} \hat{w}_{k-1}], \quad k = 1, 2, ..., \quad (2.1)$$

where

- $\hat{w}_k$ $\equiv$ vector of estimated weights for the iteration $k$;
- $\mu$ $\equiv$ positive step size, which controls the influence of the correction term in the update;
- $r_{xd}$ $\equiv$ cross-correlation vector;
- $R_{xx}$ $\equiv$ autocorrelation matrix.
The Steepest Descent algorithm finds the minimum of the cost function given by the Mean Square Errors (MSE):

$$\text{MSE} = E \left\{ \left| d_k - \hat{w}_k^T x_k \right|^2 \right\}, \quad (2.2)$$

where $d_k \equiv$ is the sample $k$ for the vector $d$.

The a-priori knowledge $[r_{xd}, R_{xx}]$ can be replaced by its instantaneous estimates,

$$r_{xd} \approx x_k d_k^* \quad (2.3)$$
$$R_{xx} \approx x_k^* x_k^T. \quad (2.4)$$

The substitution of Eq. (2.3) and Eq. (2.4) in Eq. (2.1) leads to the LMS update equation:

$$\hat{w}_k = \hat{w}_{k-1} + \mu x_k^*[d_k - x_k^T \hat{w}_{k-1}], \quad k = 1, 2,... \quad (2.5)$$

It may seem that LMS will not achieve a good performance because of these instantaneous estimates. However, as LMS is recursive itself it averages over these estimates throughout the adaptation process [16].

For a constant step-size parameter $\mu$, the LMS algorithm is convergent in the mean square if and only if

$$0 < \mu < \frac{2}{\lambda_{\text{max}}}, \quad (2.6)$$

where $\lambda_{\text{max}}$ is the largest eigenvalue of the autocorrelation matrix $R_{xx}$.

After this brief introduction of the LMS, the speed improvement process is explained. First, the algorithm will be executed without optimization by Pseudocode 1. The time performance of this simulation is shown in Fig 2.3. For a length of input sequence, $M = 4 \cdot 10^4$ and a finite impulse response (FIR) filter with $m = 8$ taps, the obtained learning behaviour is depicted in Fig 2.5. As the algorithm is tested for a FIR filter of length $m$, the vector $x_k$ is a subsequence of the input vector $x$ and corresponds to the samples $k - m + 1$ to $k$ for the iteration $k$.

The behaviour is presented by the following figures of merit: The instantaneous error is represented by $10 \log_{10}(|e_k|^2)$, with $e_k = d_k - \hat{w}_k^T x_k$ and the parameter error vector is evaluated by the relative parameter mismatch $m_w(k) = 10 \log_{10}\left(\frac{||\hat{w}_k - w||_2}{||w||_2}\right)$, with the coefficient vector $w$ of the reference FIR filter.

This simple identification problem of a FIR filter does not require as many iterations as used in latter simulation. However, they are employed to illustrate the timing issue caused by the indexing of long vectors and its solution. As it can be observed, for
Figure 2.3: Distribution of the simulation time, for the LMS without improvement, obtained with MATLAB profiler.

$M = 4 \cdot 10^4$ the simulation lasts approximately 764 seconds. Considering that the identification of PA needs more than $10^6$ iterations and the execution time increases exponentially (see Fig 2.2), the simulations would possibly last days.

The optimization of the code is given by splitting all long vectors, such as $\bar{e}_k$, $e$, $d$ or the matrix containing the $\hat{\mu}_k$ into auxiliar vectors/matrices of smaller size. It may seem to be a bad idea, since nested loops are supposed to slow down the code, but performing Pseudocode 2 with a reference length of $M = 10^3$, it is evidenced (see Fig 2.4) that these nested loops improve the timing performance of the code substantially. To verify that this modification does not affect the operation of the algorithm, results are displayed in Fig 2.6. In Table 2.2 the symbols of the variables are displayed along with the MATLAB variables which are referred to.

Finally, it is displayed that the time has decreased from $\approx 763$, 8 seconds to $\approx 121$, 6, i.e., the duration has been reduced by an 84, 2%, therefore our modification of the code can be considered as a solution for the timing issue.

<table>
<thead>
<tr>
<th>Line Number</th>
<th>Code</th>
<th>Calls</th>
<th>Total Time</th>
<th>% Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>57</td>
<td>$w_{fp}(:,k+1) = w_{fp}(:,k) + u_{fp}$...</td>
<td>40000</td>
<td>396.069 s</td>
<td>52.1%</td>
</tr>
<tr>
<td>54</td>
<td>$e_{fp}(k) = d_{fp}(k) - x_{shifted_{fp}}$...</td>
<td>40000</td>
<td>315.660 s</td>
<td>41.3%</td>
</tr>
<tr>
<td>51</td>
<td>$d_{fp}(k) = a_{fp}.*x_{shifted_{fp}}(n-1)$...</td>
<td>40000</td>
<td>46.368 s</td>
<td>6.1%</td>
</tr>
<tr>
<td>73</td>
<td>$plot(1:M+1,e3,'b','LineWidth',...</td>
<td>1</td>
<td>1.468 s</td>
<td>0.2%</td>
</tr>
<tr>
<td>72</td>
<td>$figure$</td>
<td>1</td>
<td>0.776 s</td>
<td>0.1%</td>
</tr>
<tr>
<td>All other lines</td>
<td></td>
<td>1.484 s</td>
<td>0.2%</td>
<td></td>
</tr>
<tr>
<td>Totals</td>
<td></td>
<td>763.830 s</td>
<td>100%</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.2: Correspondence between MATLAB symbols and variable symbols in the pseudocodes.
### Pseudocode 1: Main loop of LMS Algorithm without improvement

**Input:**

- \( x \) \( \cdots \) input sequence
- \( M \) \( \cdots \) length of \( x \)
- \( m \) \( \cdots \) number of filter taps
- \( d \) \( \cdots \) desired signal, size \( M \)
- \( \mu \) \( \cdots \) step size

**Internal:**

- \( \varepsilon \) \( \cdots \) error vector, length \( M \)
- \( \hat{W} \) \( \cdots \) matrix containing the estimated weight vectors \( \hat{w}_k \), size \( m \times M + 1 \)
- \( X \) \( \cdots \) matrix containing in its columns the vectors \( x_k \) of size \( m \), size \( m \times M \)

**Initialization:**

\[
\text{for } k = 1 \rightarrow M \text{ do}
\]

\[
e_k \leftarrow d_k - X[:,k]^T \hat{W}[:,k]
\]

%%% The notation \( e_k \) indicates that the sample \( k \) of the vector \( \varepsilon \) is selected. %%

\[
\hat{W}[:,k+1] \leftarrow \hat{W}[:,k] + \mu X[:,k] e_k
\]

end for

---

**Figure 2.4:** Distribution of the simulation time, for the LMS after improvement, obtained with MATLAB profiler.

### Lines where the most time was spent

<table>
<thead>
<tr>
<th>Line Number</th>
<th>Code</th>
<th>Calls</th>
<th>Total Time</th>
<th>% Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>( w_t(:,k+1) = w_t(:,k) + u_{fp})</td>
<td>40000</td>
<td>48.517 s</td>
<td>40.2%</td>
</tr>
<tr>
<td>66</td>
<td>( e_t(k) = d_t(k) - x_t(:,k) \cdot w_t(:,k) )</td>
<td>40000</td>
<td>40.386 s</td>
<td>33.5%</td>
</tr>
<tr>
<td>43</td>
<td>( d_t(k) = s_{fp}. \cdot x_t(:,k) )</td>
<td>40000</td>
<td>29.117 s</td>
<td>24.1%</td>
</tr>
<tr>
<td>74</td>
<td>( w_{fp}(i,(ii-1) \cdot M+2:ii \cdot M+1) = \ldots )</td>
<td>40</td>
<td>0.450 s</td>
<td>0.4%</td>
</tr>
<tr>
<td>92</td>
<td>( \text{plot}(1:M+1,w3,'b','LineWidth',...) )</td>
<td>1</td>
<td>0.252 s</td>
<td>0.2%</td>
</tr>
<tr>
<td>All other lines</td>
<td></td>
<td></td>
<td>1.871 s</td>
<td>1.6%</td>
</tr>
<tr>
<td>Totals</td>
<td></td>
<td></td>
<td>120,586 s</td>
<td>100%</td>
</tr>
</tbody>
</table>
Figure 2.5: Performance of LMS algorithm for a FIR filter of $m = 8$ taps, over $M = 4 \cdot 10^4$ iterations, stated in terms of the instantaneous error power (left) and relative parameter mismatch (right).

Figure 2.6: Performance of LMS algorithm for a FIR filter of $m = 8$ taps, over $M = 4 \cdot 10^4$ iterations and $M_1 = 10^3$, stated in terms of the instantaneous error power (left) and relative parameter mismatch (right).
Pseudocode 2 Main loop of LMS Algorithm with improvement

**Input:**

- $x \cdots$ input sequence
- $M \cdots$ length of $x$, $M$ has to be multiple of the reference length $M_1$
- $d \cdots$ desired signal, size $M$
- $m \cdots$ number of filter taps
- $\mu \cdots$ step size

**Internal:**

- $e \cdots$ error vector, length $M$
- $\hat{W} \cdots$ matrix containing the estimated weight vectors $\hat{w}_k$, size $m \times M + 1$
- $X \cdots$ matrix containing in its columns the vectors $x_k$ of size $m$, size $m \times M$
- $M_1 \cdots$ reference length for splitting
- $e_i \cdots$ auxiliar error vector, length $M_1$
- $W_t \cdots$ auxiliar estimated weight vector, size $m \times M_1 + 1$
- $d_t \cdots$ auxiliar desired vector, length $M_1$
- $X_t \cdots$ auxiliar matrix for the input matrix $X$, length $m \times M_1$

**Initialization:**

for $i = 1 \rightarrow M/M_1$ do

\[ X_t \leftarrow X[:, (i-1)M_1 + 1], \cdots, X[iM_1] \]
\[ W_t[:, 1] \leftarrow \hat{W}[:, (i-1)M_1 + 1] \]

\% The notation $X[a,b]$ indicates that the element of the row $a$ and column $b$ of matrix $X$ is selected. Also the colon symbol "\(\cdot\)" here means that all the element of the row/column are selected. \%

for $k = 1 \rightarrow M_1$ do

\[ e_{tk} \leftarrow d_{tk} - X_t[:, k]^T W_t[:, k] \]
\[ W_t[:, k + 1] \leftarrow W_t[:, k] + \mu X_t[:, k]^* e_{tk} \]

end for

$e_{(i-1)M_1 + 1}, \cdots, e_{iM_1} \leftarrow e_t$

\[ \hat{W}[:, (i-1)M_1 + 2], \cdots, \hat{W}[iM_1 + 1] \leftarrow W_t[:, 2], \cdots, W_t[:, end] \]

end for
2.3 Word Length Selection

In fixed-point implementation, the choice of the word length is a key problem. It should be large enough for the precision required by the algorithm under discussion and as short as possible in order to keep the costs as low as possible. Longer word lengths may improve the performance of the algorithm at the expense of hardware costs. Shorter ones may cause overflows or underflows that degrade the system performance. Therefore, setting the optimal word length means to find the one that maximizes application performance and minimizes hardware costs.

Since there is no formula to obtain the optimal word length, it will be searched for by a simulation-based approach. By means of several simulations and comparison of their performance, the shortest word length which satisfies the performance requirements (e.g., an established level of error in steady-state) will be considered as optimum.

The Recursive Least Squares (RLS) algorithm is chosen to perform the search for the optimum word length. The RLS is a popular adaptive algorithm known for its fast convergence. However, this feature is achieved at expense of high computational complexity. In addition, this algorithm has a drawback related to the large dynamic range of the variables, which may cause overflows in fixed-point implementation [18]. Those drawbacks make the RLS a good choice to select the optimum word length, since these are the troubles that we will find in the fixed-point implementation of the algorithms in this thesis. Hence, first the RLS algorithm will be introduced and then the selection of the word length.

2.3.1 Recursive Least Squares Algorithm

As said above, RLS algorithm applied to a FIR filter will be used for illustrating this issue. As its name indicates, this algorithm is just a recursive version of Least Squares (LS) solution and it retains its properties. The following derivations are taken from [19]. Considering N observations:

\[
d^T = [d_1, d_2, \ldots, d_N],
\]

\[
w_k^T = [w_k(1), w_k(2), \ldots, w_k(m)],
\]

\[
X_N = \begin{pmatrix}
x(1) & x(2) & \cdots & x(m) \\
x(2) & x(3) & \cdots & x(m+1) \\
\vdots & \vdots & \ddots & \vdots \\
x(N) & x(N+1) & \cdots & x(N+m-1)
\end{pmatrix}
\begin{pmatrix}
x_1^T \\
x_2^T \\
\vdots \\
x_N^T
\end{pmatrix}.
\]
The problem is to estimate the weight vector $\hat{w}$ such that the following cost function $c(\hat{w})$ is minimized
\[ c(\hat{w}) = \| d_N - X_N \hat{w} \|_2^2. \]  
(2.10)

Therefore, the solution is given by
\[ \hat{w}_{LS} = \arg \min_{\hat{w}} (c(\hat{w})). \]  
(2.11)

Since $c(\hat{w})$ is a quadratic function, differentiating from Eq. (2.10) with respect to $\hat{w}$ and setting it to zero, allows us to identify an extremum, which can be identified as the desired minimum. Hence, the following condition is obtained:
\[ \frac{\partial \| d_N - X_N \hat{w} \|_2^2}{\partial \hat{w}} = -(d_N - X_N \hat{w})^H X_N = 0, \]  
(2.12)

where $(\cdot)^H$ denotes complex conjugate transpose and thus the LS solution is found:
\[ X_N^H X_N \hat{w}_{LS} = X_N^H d_N. \]  
(2.13)

Then the LS estimator is obtained:
\[ \hat{w}_{LS} = [X_N^H X_N]^{-1} X_N^H d_N. \]  
(2.14)

The number of terms to compute $X_N^H X_N$ increases with the observations. For that reason, it is interesting to find a method that computes the result with the fewest operations as possible.

The initial value of $\hat{w}$, $\hat{w}_0$, is weighted by the parameter matrix $\Pi_0^{-1}$. If the elements of $\Pi_0^{-1}$ are set to a large value, it means that the initial values $\hat{w}_0$ are very certain, and then this first term predominates over the following ones. Thus, the cost function for the observation $N$ is obtained:
\[ c(\hat{w}_N) = (\hat{w}_N - \hat{w}_0)^H \Pi_0^{-1} (\hat{w}_N - \hat{w}_0) + (d_N - X_N \hat{w}_N)^H (d_N - X_N \hat{w}_N), \]  
(2.15)

and the corresponding normal equations are:
\[ \begin{bmatrix} (d_N - X_N \hat{w}_0)^H (d_N - X_N \hat{w}_0) & (d_N - X_N \hat{w}_0)^H X_N \\ X_N^H (d_N - X_N \hat{w}_0) & \Pi_0^{-1} + X_N^H X_N \end{bmatrix} \begin{bmatrix} 1 \\ \hat{w}_0 - \hat{w}_{LS} \end{bmatrix} = \begin{bmatrix} c(\hat{w}_{LS}) \\ 0 \end{bmatrix}. \]  
(2.16)

Considering the observation $N$, the solution at $N + 1$ can be redefined as a composition of two parts:
\[
\hat{w}_{N+1} = [\Pi_0^{-1} + X_N^H X_{N+1}]^{-1}[X_N^H d_{N+1}] = \\
\left[\Pi_0^{-1} + [X_N^H \hat{x}_{N+1}^*] \begin{bmatrix} X_N \\ \mathbb{E}_N^{T} \end{bmatrix}\right]^{-1}[X_N^H \hat{x}_{N+1}^* \begin{bmatrix} d_N \\ \mathbb{E}_{N+1} \end{bmatrix}] = \\
[\Pi_0^{-1} + X_N^H X_N + \mathbb{E}_{N+1}^* \mathbb{E}_{N+1}^T]^{-1}[X_N^H d_N + \mathbb{E}_{N+1}^* d_{N+1}].
\]

(2.17)

Defining the matrix \( P_N \) as

\[
P_{N+1} = [(\Pi_0^{-1} + X_N^H X_{N+1}]^{-1}; P_0 = \Pi_0,
\]

(2.18)

it can be reformulated as a recursion:

\[
P_{N+1}^{-1} = P_N^{-1} + \mathbb{E}_{N+1}^* \mathbb{E}_{N+1}^T; P_0 = \Pi_0.
\]

(2.19)

Finally, by inverting Eq. (2.19) we can obtain \( P_{N+1} \):

\[
P_{N+1} = P_N - \frac{P_N \mathbb{E}_{N+1}^* \mathbb{E}_{N+1}^T P_N}{1 + \mathbb{E}_{N+1}^* \mathbb{E}_{N+1}^T} P_N \mathbb{E}_{N+1}^*; P_0 = \Pi_0.
\]

(2.20)

With Eq. (2.18), the recursive formula for \( P_{N+1} \) can be inserted in Eq. (2.17) leading to the following recursive expression of \( \hat{w}_{N} \) is:

\[
\hat{w}_{N+1} = \hat{w}_N + \frac{P_N \mathbb{E}_{N+1}^*}{1 + \mathbb{E}_{N+1}^* \mathbb{E}_{N+1}^T}(d_{N+1} - \mathbb{E}_{N+1}^* \hat{w}_N).
\]

(2.21)

This formula is similar to the description of the LMS. The main difference is that the update direction depends via \( P_N \) on the past. The resulting regression vector \( k_{N+1} \) is found to be:

\[
k_{N+1} = P_N \mathbb{E}_{N+1}^* \gamma(N + 1); \quad \gamma(N + 1) = \frac{1}{1 + \mathbb{E}_{N+1}^* \mathbb{E}_{N+1}^T}.
\]

(2.22)

The scalar \( \gamma(N + 1) \) is called the conversion factor and establishes a relation between the a priori and a posteriori errors. With these new definitions of parameters, Eq. (2.20) can be reformulated as

\[
P_{N+1} = P_N - \frac{k_{N+1}^H k_{N+1}}{\gamma(N + 1)}.
\]

(2.23)

More information about RLS can be found in [16].
Table 2.3: RLS settings for the selection of the optimal word length.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>( M )</th>
<th>( \delta )</th>
<th>( m )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>( 10^3 )</td>
<td>( 10^{-2} )</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 2.4: Definition of the fraction lengths for the word length selection.

<table>
<thead>
<tr>
<th>Fraction Length 1</th>
<th>Word length - 1 bit (sign) -1 bit (magnitude)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fraction Length 2</td>
<td>Word length - 1 bit (sign) -10 bit (magnitude)</td>
</tr>
</tbody>
</table>

2.3.2 Selection of the Overall Word Length

RLS will be performed by Pseudocode 3. The simulation will have the data shown in Table 2.3 and the correspondence between the symbols and MATLAB variables is displayed in Table 2.5. Through the whole derivation, \( N \) has been used as the number of iterations, while \( M \) is the length of the input signal \( x \) and they are related as follows

\[
N = M - m + 1. \tag{2.24}
\]

In order to search for the optimal word length, the simulation will be operated with several word lengths. The word length sweep goes from 18 to 38 bits. Moreover, two fraction lengths are defined, one for signals whose magnitude is less than or equal to 1 and another for the rest of the variables. Studying the code we noticed that the critical point, i.e. the operation/variable which may lead to overflows, is located in the calculation of \( \gamma \). As seen in Eq. (2.22), the denominator of \( \gamma (N + 1) \) is \( 1 + \bar{x}_N^T P_N \bar{x}_{N+1} \). Taking into account that \( |x(n)| \leq \sqrt{2} \) in the simulation and the parameters defined in Table 2.3, the maximum value of the second term can be calculated:

\[
\begin{align*}
\bar{x}_N^T P_N \bar{x}_{N+1} &= \bar{x}_N^T \delta^{-1} I_{4 \times 4} \bar{x}_{N+1} = \\
\delta^{-1} \bar{x}_N^T I_{4 \times 4} \bar{x}_{N+1} &= \delta^{-1} \bar{x}_N^T \bar{x}_{N+1} = \\
\delta^{-1} (x(N + 1)x^*(N + 1) + \cdots + x(N + 4)x^*(N + 4)) &= \\
\delta^{-1} (|x(N + 1)|^2 + \cdots + |x(N + 4)|^2) &= \\
\delta^{-1} 4 \sqrt{2}^2 &= 800. \tag{2.25}
\end{align*}
\]

The variables have to be capable of representing the intermediate value 801. Thus, the integer part of the fi object has to be at least 10 bits long. Since we are using signed fi objects, one bit from the word length is taken for the sign. The definition of the two fraction lengths can be seen in Table 2.4. The results are averaged over 5 runs and are shown in Fig. 2.7.
Chapter 2. Fixed-point Implementation on MATLAB

<table>
<thead>
<tr>
<th>Variable symbol</th>
<th>$x$</th>
<th>$d$</th>
<th>$X$</th>
<th>$e$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable in MATLAB</td>
<td>$x_{fp}$</td>
<td>$d_{fp}$</td>
<td>$X_{fp}$</td>
<td>$e_{fp}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable symbol</th>
<th>$\hat{W}$</th>
<th>$\gamma$</th>
<th>$P$</th>
<th>$\hat{K}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable in MATLAB</td>
<td>$w_{fp}$</td>
<td>$gamma_{fp}$</td>
<td>$P_{fp}$</td>
<td>$k_{fp}$</td>
</tr>
</tbody>
</table>

Table 2.5: Correspondence between MATLAB symbols and variable symbols in Pseudocode 3.

**Pseudocode 3** Main loop of RLS Algorithm

**Input:**
- $x$ ... input signal
- $d$ ... desired signal, length $N$
- $M$ ... length of $x$
- $m$ ... number of filter taps
- $N$ ... number of iterations $\equiv M - m + 1$

**Internal:**
- $X$ ... matrix of input observations, size $m \times N$
- $e$ ... error vector, length $N$
- $K$ ... matrix of regression vectors, size $m \times N$
- $\hat{W}$ ... matrix of estimated weight vectors, size $m \times N$
- $\gamma$ ... vector of conversion factors, length $N$
- $\bar{P}$ ... size $m \times m$

Initialization:
- $\hat{W}[:,:1] \leftarrow 0$
- $K[:,:1] \leftarrow 0$
- $P \leftarrow \delta^{-1}I_m$
- $\gamma_1 \leftarrow 1$

**for** $n = 1 \rightarrow N - 1$ **do**

- $\gamma_{n+1} \leftarrow \frac{1}{1 + X(:,n+1)^T P X(:,n+1)^*} \gamma_n$
- $K(:,n+1) \leftarrow P X(:,n+1)^* \gamma_{n+1}$
- $\hat{W}(:,n+1) \leftarrow \hat{W}(:,n) + K(:,n+1) \left[ d_{n+1} - X(:,n+1)^T \hat{W}(:,n) \right]$
- $P \leftarrow P - K(:,n+1) K(:,n+1)^T \gamma_{n+1}$
- $e(n) \leftarrow d_{n+1} - X(:,n+1)^T \hat{W}(:,n)$

**end for**
Figure 2.7: Results for RLS simulation averaged over 5 runs. Performance represented in terms of MSE (top) and relative parameter mismatch (bottom). The legend shows the different wordlengths and their corresponding colors.
Notice that the results for word lengths 18 and 22 bits are discarded because the algorithm does not work in these cases since these configurations cause underflows which lead to a malfunction. The results for word lengths from 30 to 38 bits can be considered valid. Therefore, choosing the optimal word length is a trade-off between the performance of the algorithm and computational cost. Considering Fig. 2.7, 32 bits will be chosen as the optimal word length for this algorithm. It is clear that word lengths greater than 32 bits would allow for reaching a lower error level, but we must consider the limitation of the resources, e.g., the RAM of the computer, when we choose the word length (it may not be a problem for this algorithm, but for more complex ones could be significant).

2.3.3 Selection of the Fraction Length

Now that the word length is set, the fraction length has to be chosen. In fact, two fraction lengths will be considered, one for variables which magnitude will not exceed the unit and another one for signals and variables which need a higher rank for the integer part. The fraction length for the first type of variables is easily estimated since from the 32 bits of the whole word, one bit is required for the sign and another one for the integer part. This leaves 30 bits of the word for the fractional part. However, for the variables which assume magnitudes that exceed one, choosing the fraction length requires further considerations. Amongst them, the magnitude of the maximum value to be represented and the precision needed for calculations. This fraction length will also be used in the fimath settings (See Sec. 2.1), which has to be the same for all fi objects, in order to interoperate between them because fi operations require the fimath object of both operands to be equal. MATLAB function \texttt{reinterprecast} cannot be used to this purpose because this function converts the data type of the fi object, but it does nothing with the fimath settings, such as \texttt{SumMode} and \texttt{ProductMode}.

Considering anew Pseudocode 3 with the same settings as before (see Table 2.3) and setting the word length to 32 bits, the simulation is executed for values of the fraction length from 24 bits to 2 bits to exemplify the above discussed behaviour. As it can be observed in Fig. 2.8, there are two opposite situations. On the one hand, when the fraction length is too large (e.g., fraction lengths greater than 22 bits), the integer part is too narrow which results in overflows that cause damage in the performance. On the other hand, when the fraction length is too short, there is a loss of precision (e.g., fraction lengths from 10 bits to 2 bits) which may lead to an improper operation of the algorithm. After these observations and results, the most appropriate fraction length in the case at hand, i.e. RLS with the defined settings, seems to be 20 bits. It is likely that in other algorithms the right choice will not be the same one.
Figure 2.8: Results for RLS simulation averaged over 5 runs. Performance represented in terms of MSE (top) and relative parameter mismatch (bottom). The legend shows the different fraction lengths and their corresponding colors.
Chapter 3

Behavioural Modeling of Power Amplifiers

Several formulations have been proposed for behavioural modeling and pre-distortion of RF power amplifiers. These formulations are mathematical models which relate input and output signal and are used to represent the system properties.

Since there is no measured data available, we use a model as a replacement for a real PA thereby providing the training data. This replacement model, henceforth called numeric PA model, should be excited with the proper input signal in order to observe the behaviour of interest. These observations will allow us to extract the parameters of the numeric PA model.

In order to have a reference system which could be, in the ideal case, perfectly matched by the adapted model we have a second model called the parametrized reference model. This reference model has the same structure as the model we intend to adapt to the training data. Moreover, this reference system allows to evaluate the distance between the adapted parameters and the reference ones.

The procedure begins with the choice of the formulation and follows with the extraction of the parameters, considering the characteristics of the PA constant during the operation. By this extraction, the parametrized reference model is obtained.

Afterwards, the PA is considered time varying, therefore the estimation of the parameters should be performed adaptively. Obviously, different formulations and different training signals lead to different performances.

In this Chapter, the following models are presented:

- Memoryless polynomials
• Memory polynomials
• Model based on truncated Volterra Series
• Dynamic deviation reduction-based Volterra Series
• Decomposed piecewise Volterra Series

3.1 Memoryless Polynomial Model

Polynomial models are widely used to describe the nonlinear effects of PAs, as well as LUT approaches which are easier to implement but take longer to converge and its linear curve may introduce additional nonlinearities [20].

Considering the passband input $\tilde{x}(t)$ of a nonlinear system, e.g. a PA, and the corresponding output $\tilde{y}(t)$, the AM-AM conversion is the function mapping from the input magnitude $|\tilde{x}(t)|$ to the output magnitude $|\tilde{y}(t)|$. Similarly, the AM-PM conversion is defined as the function mapping from the input magnitude $|\tilde{x}(t)|$ to the output phase offset $\angle \tilde{y}(t) - \angle \tilde{x}(t)$. Let us describe the PA as a polynomial model in the passband,

$$\tilde{y}(t) = \sum_{p=1}^{P} \tilde{b}_p \tilde{x}^p(t),$$

with the nonlinearity order $P$ and $\tilde{b}_p$ the vector of expansion coefficients. Eq. (3.1) can be transformed into its baseband representation

$$y(t) = \sum_{p=1}^{P} b_p |x(t)|^{p-1} x(t), \ p \ odd,$$

where \[21\]

$$b_p = 2^{1-p} \left( \frac{p}{p-1} \right) \tilde{b}_p.$$  

Notice that both $b_p$ and $\tilde{b}_p$ are real valued. Ding and Zhou in [22] proposed to add even order terms to Eq. (3.2) in order to increase the modeling accuracy and lower the polynomial order, which have better numerical properties than higher order polynomials.

Up to now, this modeling of PA is purely memoryless since AM-PM conversion is constant. However, if the PA model has short-term memory effects, it can be considered quasi-memoryless and it exhibits both AM-AM and AM-PM conversions. To approximate a nonlinear PA with memory Volterra Series can be used. In passband this is given
by
\[
\tilde{y}(t) = \sum_{p=1}^{\infty} \int_{V(p)} \int \tilde{h}_p(\tau_p) \prod_{i=1}^{p} \tilde{x}(t - \tau_i) d\tau_p, \tag{3.4}
\]
where \(\tau_p = [\tau_1, \tau_2, \cdots, \tau_p]^T\), \(d\tau_p = d\tau_1 d\tau_2 \cdots d\tau_p\), \(\tilde{h}_p(\cdot)\) is the real-valued \(p\)-th-order Volterra kernel and \(V(p) = (-\infty; \infty)^p\) is the infinite \(p\)-cube over which the integration is performed. If the signal \(\tilde{x}(t - \tau_i)\) is narrow band (i.e., \(\tilde{x}(t - \tau_i) \approx \tilde{x}(t)\)) and the memory effects can be considered as short-term effects, the model represented by Eq. (3.4) is like Eq. (3.2) with \(b_p\) complex valued, according to [21]. A more detailed overview of Volterra Series will be presented on Sec. 3.2.2.

### 3.1.1 Parameter Estimation

As mentioned in the introduction of this chapter, after choosing a behavioural model to represent the PA, the parameters of the parametrized reference model should be obtained. For this estimation, it is assumed that PA characteristics remain constant throughout the process.

Since \(y(t)\) is linear in the parameters \(b_p\), these can be estimated by means of LS solution as explained in [20].

By defining the nonlinear basis function
\[
\phi_p(x) = x|x|^{p-1}, \tag{3.5}
\]
Eq. (3.2) can be reformulated as
\[
y(t) = \sum_{p=1}^{P} b_p \phi_p(x(t)). \tag{3.6}
\]

Taking \(N\) observations of \(x(t)\) and \(y(t)\), Eq. (3.6) can be rewritten in matrix notation as
\[
\underline{y} = \Phi \underline{b}, \tag{3.7}
\]
where \(\underline{x} = [x(t_1), \cdots, x(t_N)]^T\) is the \(N \times 1\) input data vector, \(\underline{y} = [y(t_1), \cdots, y(t_N)]^T\) is the \(N \times 1\) output data vector, \(\underline{b} = [b_1, \cdots, b_P]^T\) is the \(P \times 1\) parameter vector and \(\Phi = [\phi_1(\underline{x}), \cdots, \phi_P(\underline{x})]\) is a \(N \times P\) matrix with \(\phi_p(\underline{x}) = [\phi_p(x(t_1)), \cdots, \phi_p(x(t_N))]^T\).

The LS solution of Eq. (3.7) is
\[
b_{LS} = (\Phi^H \Phi)^{-1} \Phi^H \underline{y}. \tag{3.8}\]
Chapter 3. *Behavioural Modeling of Power Amplifiers*

3.2 Models with Memory

Conventionally, behavioural models for PAs have been built based on the AM-AM and AM-PM characteristics of its nonlinearity, and the complex gain has been approximated as a function of the instantaneous input power level [23].

Nevertheless, for applications with wider bandwidths, memory effects can no longer be neglected. These dynamic distortions can be either of thermal or electrical origin as shown in [24]. In the presence of memory effects, the instantaneous gain of the PA is a function of the current input sample and a finite number of the previous input samples. Despite this, the static nonlinearities keep being stronger than the memory effects.

The use of a traditional pre-distorter, which does not take into account the memory effects, for a PA with such nonlinearities causes a low quality performance. This performance is degraded as the bandwidth of the input signal increases. Therefore, the memory effects have to be incorporated in the behavioural model in order to have an accurate performance of the algorithm.

3.2.1 Memory Polynomial

In this section, the first and simplest model that involves both, memory effects and static nonlinearity, is proposed.

Volterra Series is a general nonlinear model with memory. The main problem of this model is the large number of parameters that has to be extracted from the PA. In order to relieve the complexity, there are several ways to prune Volterra Series. One of them is the memory polynomial proposed by Kim and Konstantinou in [23].

The memory polynomial model is commonly used for digital pre-distortion and modeling of PAs which presents memory effects. As mentioned before, this model is a reduction of Volterra Series (see Sec. 3.2.2). In this reduction only diagonal terms of Volterra kernels are retained.

The model represented by Eq. (3.2) has the same behaviour over the whole band of operation, thus it can be considered as a narrowband system. The behaviour of a real PA depends on the signal frequencies and the output is a function of the current input and also of the past inputs. Another model which combines both, memory effects and static nonlinearity, is described by

$$y[n] = \sum_{q=0}^{M-1} B_q(a_q, x[n-q]),$$  \hspace{1cm} (3.9)
where $M$ is the memory depth of the model and

$$B_q(x[n]) = x[n] \sum_{p=1}^{P} a_{pq} |x[n]|^{p-1}, \quad (3.10)$$

where $P$ is the order of polynomials, $a_{pq}$ are the complex valued parameters of the model and $a_q$ is the vector of parameters, so that $a_q = [a_{1q}, a_{2q}, \ldots, a_{Pq}]$.

Combining Eq. (3.9) and Eq. (3.10), the output signal of the model is obtained as

$$y[n] = \sum_{p=1}^{P} \sum_{q=0}^{M-1} a_{pq} x[n-q] |x[n-q]|^{p-1}, \quad (3.11)$$

where even and odd-order terms are included. Usually, only odd-order terms are incorporated. Following the analysis of [25], this would be represented by

$$y[n] = \left\lfloor \frac{P+1}{2} \right\rfloor \sum_{p=1}^{P} \sum_{q=0}^{M-1} a_{2p-1 q} x[n-q] |x[n-q]|^{2(p-1)}. \quad (3.12)$$

As shown in [4], spectral regrowth can be further reduced by including even-order terms in the pre-distorter. An extended report about the advantages of adding even-order terms in the baseband model represented by Eq. (3.11) can be found in [22].

### 3.2.1.1 Parameter Estimation

In this step the characteristics of the PA are considered to be almost constant. Since $y[n]$ is linear in parameters $a_{pq}$, these parameters can be estimated by means of LS.

By defining the sequence

$$u_{pq}[n] = x[n-q] |x[n-q]|^{p-1}, \quad (3.13)$$

Eq. (3.11) can be reformulated as

$$y[n] = \sum_{p=1}^{P} \sum_{q=0}^{M-1} a_{pq} u_{pq}[n]. \quad (3.14)$$

Thus, considering $N$ observations of $x[n]$ and $y[n]$, in matrix notation Eq. (3.14) becomes

$$\mathbf{y} = \mathbf{U} \mathbf{a}, \quad (3.15)$$
where,

- $M$ represents the memory depth;
- $Q = M - 1$ is the maximum delay;
- $R = M \cdot P$ is the number of parameters of the model;
- $y = [y[0], \cdots, y[N-1]]^T$ is the $N \times 1$ output data vector;
- $u_{pq} = [u_{pq}[0], u_{pq}[1], \cdots, u_{pq}[N-1]]^T$ is a $N \times 1$ vector which contains $N$ observations of the sequence $u_{pq}$;
- $U = [u_{10}, \cdots, u_{P0}, \cdots, u_{1Q}, \cdots, u_{PQ}]$ is the $N \times R$ matrix containing the $R$ sequence vectors $u_{pq}$;
- $a = [a_{10}, \cdots, a_{P0}, \cdots, a_{1Q}, \cdots, a_{PQ}]^T$ is the $R \times 1$ parameter vector;

The least-squares solution of Eq. (3.15) is

$$a_{LS} = (U^H U)^{-1} U^H y,$$  \hspace{1cm} (3.16)

### 3.2.2 Model Based on Trucated Volterra Series

The Volterra Series have their origin in the work of the Italian mathematician Vito Volterra where he was studying nonlinear functionals, integral and integro-differential equations [26],[27].

In order to model nonlinear systems with memory, the Volterra Series is a multivariable polynomial series which depends on the current input signal value and previous ones, i.e.

$$y(t) = h_0 + \sum_{p=1}^{\infty} \int \cdots \int h^{(p)}(t, \tau_1, \tau_2, \cdots, \tau_p) \prod_{i=1}^{p} x(\tau_i) d\tau_1 d\tau_2 \cdots d\tau_p,$$ \hspace{1cm} (3.17)

where $h_0$ is the zero-order Volterra kernel, $y(t)$ is the real-valued output, $x(t)$ the real-valued input and $h^{(p)}(t, \tau_1, \tau_2, \cdots, \tau_p)$ is the $p$th-order Volterra kernel.

In this thesis, the kernels are considered time-independent, the nonlinearity order is truncated to the amount $P$ and the zero-order Volterra kernel is assumed to be 0. Hence, Eq. (3.17) is reduced to

$$y(t) = \sum_{p=1}^{P} \int \cdots \int h^{(p)}(\tau_1, \tau_2, \cdots, \tau_p) \prod_{i=1}^{p} x(t - \tau_i) d\tau_1 d\tau_2 \cdots d\tau_p.$$ \hspace{1cm} (3.18)
3.2.2.1 Properties of Volterra Series

Kernel Linearity

A Linear Time Invariant (LTI) system with memory can be represented by the linear convolution

$$x_{\text{out}}(t) = \int_{-\infty}^{+\infty} h(\tau) x_{\text{in}}(t - \tau) d\tau,$$  \hspace{1cm} (3.19)

where $x_{\text{in}}(t)$, $x_{\text{out}}(t)$ and $h(t)$ are the input, the output and the impulse response of the system, respectively. Comparing Eq. (3.19) with the first order Volterra functional

$$y(t) = \int_{-\infty}^{+\infty} h(\tau)x(t-\tau) d\tau,$$  \hspace{1cm} (3.20)

the equivalence of both representations is obvious.

The representation of a two dimensional LTI system can be described as

$$x_{\text{out}}(t_1, t_2) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(\tau_1, \tau_2) x_{\text{in}}(t_1 - \tau_1, t_2 - \tau_2) d\tau_1 d\tau_2$$  \hspace{1cm} (3.21)

where $t_1$ and $t_2$ are two independent parameters. Considering that the factorization of $x_{\text{in}}(t_1, t_2)$ is given by

$$x_{\text{in}}(t_1, t_2) = x_{\text{in}}(t_1) x_{\text{in}}(t_2),$$  \hspace{1cm} (3.22)

and substituting this in Eq. (3.21) and $t \equiv t_1 = t_2$, the output of the two dimensional LTI can be represented as follows

$$x_{\text{out}}(t_1, t_2) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(\tau_1, \tau_2) x_{\text{in}}(t - \tau_1) x_{\text{in}}(t - \tau_2) d\tau_1 d\tau_2$$  \hspace{1cm} (3.23)

This representation can be easily identified as the second order Volterra functional

$$y^{(2)}(t) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h^{(2)}(\tau_1, \tau_2) x(t - \tau_1) x(t - \tau_2) d\tau_1 d\tau_2.$$  \hspace{1cm} (3.24)

Similarly, a $p$th-order LTI system is described by

$$y^{(p)}(t) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} h^{(p)}(\tau_1, \tau_2, \ldots, \tau_p) \prod_{i=1}^{p} x(t - \tau_i) d\tau_1 d\tau_2 \cdots d\tau_p.$$  \hspace{1cm} (3.25)
The Truncated Volterra Series representation is obtained by adding all the Volterra functionals up to order $P$, as in

$$y(t) = \sum_{p=1}^{P} y^{(p)}(t), \quad (3.26)$$

where $y^{(p)}(t)$ is given by Eq. (3.25).

Although $y^{(p)}(t)$ is nonlinear with respect of $x(t)$ for $p > 1$, the output $y(t)$ depends linearly on the products $x(t_1)x(t_2)\cdots x(t_p)$ and $y^{(p)}(t)$ is linear in the Volterra kernel $h^{(p)}(\tau_1, \tau_2, \cdots, \tau_p)$ [28].

**Kernel Symmetry**

It can be assumed that the Volterra kernels are symmetric, i.e., in Eq. (3.25) the overall performance is not compromised by permuting the arguments in the kernel. This assumption does not lead to any loss of generality as seen in [29]. For an asymmetric kernel the order of the arguments is relevant, therefore there are $p!$ different permutations of the kernel. It is possible to obtain a symmetric kernel $h^{(p)}_{\text{sym}}(\cdot)$ by applying a "symmetrization" to the asymmetric kernel $h^{(p)}(\cdot)$ [30]

$$h^{(p)}_{\text{sym}}(t_1, t_2, \cdots, t_p) = \frac{1}{p!} \sum_{i=0}^{p!} h^{(p)}(t_{i(1)}, t_{i(2)}, \cdots, t_{i(p)}), \quad (3.27)$$

where $i(p)_i$ denotes the $p$th element in the $i$th permutation in the ordered set $\{1, 2, \cdots, p\}$.

**Fourier Transform**

It is shown in [31] that the multivariable Fourier transform $Y^{(p)}(\cdot \cdot \cdot)$ of the $p$th Volterra functional in Eq.(3.25) is given by

$$Y^{(p)}(f_1, f_2, \cdots, f_p) = H^{(p)}(f_1, f_2, \cdots, f_p) \prod_{i=1}^{p} X(f_i), \quad (3.28)$$

where $X(f)$ is the Fourier transform of the input $x(t)$ and $H^{(p)}(f_1, f_2, \cdots, f_p)$ is the Fourier transform of the Volterra kernel

$$H^{(p)}(f_1, f_2, \cdots, f_p) = \int \int \cdots \int_{\mathcal{V}^{(p)}} h^{(p)}(\tau_1, \tau_2, \cdots, \tau_p) e^{-j2\pi \sum_{i=1}^{p} f_i \tau_i} d\tau_1 d\tau_2 \cdots d\tau_p. \quad (3.29)$$
Due to the kernel symmetry property, the Fourier transform of the Volterra kernel is also symmetric. Moreover, the inverse Fourier transform is given by

\[
y^{(p)}(t) = \frac{1}{(2\pi)^p} \int \cdots \int Y^{(p)}(f_1, f_2, \ldots, f_p) e^{j2\pi \sum_{i=1}^{p} f_i t} df_1 df_2 \cdots df_p.
\] (3.30)

### 3.2.2.2 Baseband Representation of a Volterra System

The real input signal of the system of interest is centered around the carrier frequency \( f_c \) and is restricted to a bandwidth \( B_w \), such that \( f_c >> B_w \). This is a passband signal, therefore there is an equivalent baseband representation of the system.

From [28] and [32], the complex-valued baseband representation of Volterra series is extracted and is given by

\[
y(t) = \sum_{p=1}^{\lfloor \frac{p+1}{2} \rfloor} \int \cdots \int_{V(p)} h^{(2p-1)}(\tau_1, \tau_2, \ldots, \tau_p) \prod_{i=1}^{p} x(t - \tau_i) \prod_{i=p+1}^{2p-1} x^*(t - \tau_i) d\tau_1 d\tau_2 \cdots d\tau_p,
\] (3.31)

with the equivalent baseband Volterra kernels

\[
\tilde{h}^{(p)}(t_1, t_2, \ldots, t_p) = \frac{1}{\sqrt{2^{p-1}}} \left( \frac{p}{2^{p-1}} \right) h^{(p)}(t_1, t_2, \ldots, t_p) e^{j2\pi \left( \frac{\sum_{i=1}^{p+1} t_i - \sum_{i=2}^{p+1} \frac{t_i}{2}}{2} \right)}
\] (3.32)

that are, in general, complex valued and where \( h^{(p)} \) is the real valued passband Volterra kernel.

With Eq. (3.28), the Fourier transform of the baseband Volterra model is written as

\[
Y^{(p)}(f_1, f_2, \ldots, f_p) = \tilde{H}^{(p)}(f_1, f_2, \ldots, f_p) \prod_{i=1}^{\lfloor \frac{p+1}{2} \rfloor} X(f_i) \prod_{i=\frac{p+1}{2}+1}^{p} X^*(-f_i).
\] (3.33)

### 3.2.2.3 The Discrete-Time Volterra Model

The Volterra model needs to be converted to the discrete-time case, since the algorithms will work in the digital domain. The adaptation process of the Volterra model is based on the theses [28] and [32].

Applying Nyquist-Shannon's sampling theorem, a time continuous function, such as \( x(t) \), can be reconstructed without loss of information if its one-sided bandwidth \( B_w \), is
upper bounded by $\frac{1}{2T_s}$, where $T_s$ is the sampling period. Therefore, the discrete-time signal corresponding to $x(t)$ is

$$x[n] = x(nT_s), \quad (3.34)$$

with its corresponding discrete Fourier transform (DFT)

$$X[\theta] = \sum_{n=-\infty}^{+\infty} x[n] e^{-j\theta n}, \quad (3.35)$$

and its inverse relation

$$X(f) = \begin{cases} T_s X[fT_s], & -\frac{1}{2T_s} \leq f < \frac{1}{2T_s} \\ 0, & \text{otherwise} \end{cases} \quad (3.36)$$

As the input signal $x(t)$ is band-limited with bandwith $B_w$, the $p$-order Volterra kernel is only excited within the $p$-dimensional hypercube

$$c^{(p)} = [-B_w; B_w]^p, \quad (3.37)$$

whose limit is

$$\left[ -\frac{1}{2T_s}, \frac{1}{2T_s} \right]^p. \quad (3.38)$$

Hence, the $p$-order Volterra kernel $h^{(p)}(t_1, t_2, \ldots, t_p)$ can be sampled with a regular grid of dimension $p$ and spacing $T_s$. The discrete-time Volterra kernel reads

$$h^{(p)}[n_1, n_2, \ldots, n_p] = h^{(p)}(n_1T_s, n_2T_s, \ldots, n_pT_s), \quad (3.39)$$

and its $p$-dimensional DFT is

$$H^{(p)}[\theta_1, \theta_2, \ldots \theta_p] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} \ldots \sum_{n_p=-\infty}^{\infty} h^{(p)}[n_1, n_2, \ldots, n_p] e^{-j2\pi \sum_{i=1}^{p} \theta_i n_i}. \quad (3.40)$$

Moreover, the corresponding inverse relation is given by

$$H^{(P)}(f_1, f_2, \ldots, f_p) = \begin{cases} T_s^p H^{(p)}[f_1T_s, f_2T_s, \ldots, f_pT_s], & f \in c^{(p)} \\ 0, & \text{otherwise} \end{cases}, \quad (3.41)$$

where $\underline{f} = (f_1, f_2, \ldots, f_p)$. 
With the assumption of the band-limited kernel and Eq. (3.33), the Fourier transform of the $p$th baseband Volterra functional can be described as

$$y^{(p)}(t) = \frac{1}{(2\pi)^p} \int \int \cdots \int H^{(p)}(f_1, f_2, \cdots, f_p) e^{j2\pi \sum_{i=1}^{p} f_i t} \prod_{i=1}^{\frac{p+1}{2}+i} X(f_i) \prod_{i=\frac{p+1}{2}+i}^{p} X(-f_i) df_1 df_2 \cdots df_p; \ p \text{ odd.}$$  (3.42)

Substituting Eq. (3.36) and Eq. (3.41) in Eq. (3.42) and exchanging the order of the summation and the integration, the $p$th baseband Volterra functional is expressed as a function of the sampled input signal $x[n]$ and the sampled kernel $h^{(p)}[m_1, m_2, \cdots, n]

$$y^{(p)}(t) = T^p_s \sum_{m_1=-\infty}^{\infty} \sum_{m_2=-\infty}^{\infty} \cdots \sum_{m_p=-\infty}^{\infty} h^{(p)}[m_1, m_2, \cdots, m_p] \prod_{i=1}^{\frac{p+1}{2}+1} x[n - m_i] \prod_{i=\frac{p+1}{2}+1}^{p} x^*[n - m_i] \int \int \cdots \int e^{j2\pi \sum_{i=1}^{p} f_i t(m_i + n) T_s} df_1 df_2 \cdots df_p.$$  (3.43)

The integral can be solved as follows

$$\int \int \cdots \int e^{j2\pi \sum_{i=1}^{p} f_i t(m_i + n) T_s} df_1 df_2 \cdots df_p = \left( \frac{2\pi}{T_s} \right)^p \prod_{i=1}^{p} \text{sinc} \left( \pi \left( \frac{t}{T_s} - m_i - n \right) \right).$$  (3.44)

Finally, it is obtained

$$y(t) = \sum_{p=1}^{\left[ \frac{p+1}{2} \right]} \sum_{m_1=-\infty}^{\infty} \sum_{m_2=-\infty}^{\infty} \cdots \sum_{m_p=-\infty}^{\infty} h^{(2p-1)}[m_1, m_2, \cdots, m_p] \prod_{i=1}^{p} x[n - m_i] \prod_{i=p+1}^{2p-1} x^*[n - m_i] \prod_{i=1}^{p} \text{sinc} \left( \pi \left( \frac{t}{T_s} - m_{2i-1} - n \right) \right).$$  (3.45)

As $\text{sinc} (\pi r)$ for $r$ integer is a Nyquist pulse [33] and $y(t)$ is sampled at time instants $t = nT_s$, the baseband discrete-time Volterra model reads
Chapter 3. Behavioural Modeling of Power Amplifiers

\[ y[n] = \left\lfloor \frac{P+1}{2} \right\rfloor \sum_{p=1}^{\frac{P+1}{2}} \sum_{m_1=0}^{M} \sum_{m_2=0}^{M} \cdots \sum_{m_p=0}^{M} \sum_{m_{p+1}=0}^{M} \cdots \sum_{m_{2p-1}=0}^{M} \sum_{m_{2p}=0}^{M} \cdots \sum_{m_{2p-2}=0}^{M} h^{2p-1}[m_1, m_2, \ldots, m_{2p-1}] \prod_{i=1}^{p} x[n - m_i] \prod_{i=p+1}^{2p-1} x^*[n - m_i]. \] (3.46)

Applying the kernel symmetry, Eq. (3.46) is simplified to

\[ y[n] = \sum_{p=1}^{\frac{P+1}{2}} \sum_{m_1=0}^{M} \sum_{m_2=0}^{m_1} \cdots \sum_{m_p=0}^{M} \sum_{m_{p+1}=m_p}^{m_{2p-1}} \sum_{m_{2p}=m_{2p-1}}^{M} h^{2p-1}[m_1, m_2, \ldots, m_{2p-1}] \prod_{i=1}^{p} x[n - m_i] \prod_{i=p+1}^{2p-1} x^*[n - m_i], \] (3.47)

where \( P \) is the odd nonlinearity order and \( M \) is the memory depth.

Analogously, following the same process, the passband discrete-time Volterra model is given by

\[ y[n] = \sum_{p=1}^{P} \sum_{m_1=0}^{M} \sum_{m_2=0}^{m_1} \cdots \sum_{m_p=0}^{M} h^p[m_1, m_2, \ldots, m_p] \prod_{i=1}^{p} x[n - m_i]. \] (3.48)

From Eq. (3.46) it can be observed that the number of parameters of the kernel increases exponentially with the memory depth and the nonlinearity order. This drawback limits the use of Volterra Series, because of this, various techniques have been proposed such as Memory Polynomials (see Sec. 3.2.1) and Dynamic Deviation Reduction (see Sec. 3.2.3).

### 3.2.2.4 Parameter Estimation

As in the previous models, the LS estimator is employed to estimate the model parameters. The model output, defined in Eq. (3.48) for the passband representation and in Eq. ( unnamed ) for the baseband model, can be expressed in matrix notation as

\[ y = U\hat{h}. \] (3.49)

where \( \hat{h} \) is the Volterra kernel vector and \( U \) is a matrix that contains the input samples of the model. For the passband case, \( \hat{h} \) contains the vectors of \( p \)th order with \( p = 1, \ldots, P \)

\[ \hat{h} = \left[ \hat{h}^{(1)T} \hat{h}^{(2)T} \cdots \hat{h}^{(P)T} \right]^T. \] (3.50)
where the vector $\vec{h}^{(p)}$ contains the coefficients of the $p$th-order Volterra kernel,

$$
\vec{h}^{(p)} = \begin{bmatrix}
  \underbrace{h^{(p)}[0, 0, \ldots, 0]}_{\text{order } p} \\
  \underbrace{h^{(p)}[0, 0, \ldots, 1]}_{\text{order } p} \\
  \vdots \\
  \underbrace{h^{(p)}[M, M, \ldots, M - 1]}_{\text{order } p} \\
  \underbrace{h^{(p)}[M, M, \ldots, M]}_{\text{order } p}
\end{bmatrix}.
$$

(3.51)

Defining the sequence $u[n]_M$ as the sequence that contains the instantaneous and the delayed samples of the input signal

$$
u[n]_M = \begin{bmatrix} x[n] & x[n-1] & \cdots & x[n-M] \end{bmatrix}^T,
$$

(3.52)

and using the Kronecker product [34] that for two matrices $A$ and $B$ is defined as

$$
A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1q}B \\
 a_{21}B & a_{22}B & \cdots & a_{2q}B \\
 \vdots & \vdots & \ddots & \vdots \\
 a_{r1}B & a_{r2}B & \cdots & a_{rq}B
\end{bmatrix}; \text{ where } A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1q} \\
 a_{21} & a_{22} & \cdots & a_{2q} \\
 \vdots & \vdots & \ddots & \vdots \\
 a_{r1} & a_{r2} & \cdots & a_{rq}
\end{bmatrix},
$$

(3.53)

the vector that contains the $p$th-order product terms of the delayed inputs $u[n]_{pM}$ reads

$$
u[n]_{pM} = \underbrace{u[n]_M \otimes u[n]_M \otimes \cdots \otimes u[n]_M}_{p \text{ times}}.
$$

(3.54)

Finally, the matrix $U$ that contains all the $u[n]_{pM}$ vector over an input signal of $N$ samples can be constructed as follows

$$
U = \begin{bmatrix} u[0] & u[1] & \cdots & u[N - 1] \end{bmatrix}^T;
$$

(3.55)

where

$$
u[n] = \begin{bmatrix} u[n]_{1M}^T & u[n]_{2M}^T & \cdots & u[n]_{pM}^T \end{bmatrix}^T.
$$

However, for the baseband case these vectors and matrices have to be redefined since only odd order terms are taking part in the model. Therefore, the vector $\vec{h}$ that contains all the Volterra kernels vectors $\vec{h}^{(p)}$ of odd order $p$ reads

$$
\vec{h} = \begin{bmatrix} h^{(1)T} & h^{(3)T} & \cdots & h^{(p)T} \end{bmatrix}^T.
$$

(3.56)
Moreover, $u[n]$ contains the product terms of the delayed inputs and it can be written as

$$u[n] = \left[ u[n]^{T}_{1M} \ u[n]^{T}_{3M} \ \cdots \ u[n]^{T}_{PM} \right]^T.$$ (3.57)

Thus, the LS estimator $h_{LS}$ for $h$ based on $N$ observations of input $x$ and output $y$ (see Eq. (3.49)) is given by

$$h_{LS} = (U^H U)^{-1} U^H y.$$ (3.58)

For the purpose of having a good estimation of the model, the number of samples that intervenes in the extraction procedure has to be large enough that the nonlinearities are properly excited.

### 3.2.3 Dynamic Deviation Reduction-Based Volterra Series Model

As mentioned in Sec. 3.2.2 the number of Volterra parameters increases exponentially as the order of nonlinearity $P$ and the memory depth $M$ grow. Therefore, the high computational complexity makes this model impractical in some real time applications [35]. This drawback leads to several techniques such as "pruning", i.e., the terms that can be neglected due to their small influence on the overall behaviour are set to zero, and the dynamic deviation reduction technique. The latter will be described in this section.

Dynamic deviation reduction was proposed by Zhu, Pedro and Brazil in [35]. The reduction is based on the modified Volterra Series developed in [36] and [37] that separates the static characteristics from the dynamic ones and extended to the discrete-time domain, i.e.,

$$y[n] = y_s[n] + y_d[n].$$ (3.59)

Due to the fact that nonlinear dynamic effects tend to fade as the nonlinearity order increases, higher order dynamics are removed from the model. Unlike the modified Volterra series proposed by Filicori [36][37], this approach retains the property of linearity in model parameters, which is essential for the extraction procedure by means of LS.

The static terms $y_s[n]$ in Eq. (3.59) can be formulated as a power series of the current input $x[n]$

$$y_s[n] = \sum_{p=1}^{P} h^{(p)}[0,\cdots,0][x^p[n]],$$ (3.60)

while $y_d[n]$ is the purely dynamic part and it can be expressed as
\[ y_d[n] = \sum_{p=1}^{P} \sum_{r=1}^{r} x^{p-r} [n] \sum_{i_1=1}^{M} \sum_{i_2=1}^{M} \cdots \sum_{i_r=r-1}^{M} h^{(p)}[0, \cdots, 0,i_1, \cdots, i_r] \prod_{j=1}^{r} x[n - i_j], \] (3.61)

where the dynamic deviation reduction order \( r \) is a variable introduced to represent the possible number of product terms of delayed inputs and \( h^{(p)}[0, \cdots, 0,i_1, \cdots, i_r] \) is the Volterra kernel with \( p \)th-order of nonlinearity and \( r \)th-order of dynamics. This number of product terms can be truncated to \( r \leq P \).

As observed in [35], in practice first order truncation, i.e., \( r=1 \), might be not enough to model memory effects of solid-state amplifiers. Therefore, higher dynamics terms are required to be added to the model. However, adding more terms to the dynamics leads to an increase in complexity of the model and computational costs. In order to reduce this complexity, the dynamic deviation reduction order \( r \) should be kept at a small value \( R \), i.e., \( 1 \leq r \leq R \).

The choice of the truncation order \( R \) depends on the characteristics of the PA and the degree of fidelity needed. In the model based on truncated Volterra Series (see Sec. 3.2.2), the truncation is given by the nonlinearity order \( P \) and memory depth \( M \), the dynamic deviation reduction provides one more truncation parameter \( R \) and allows the number of parameters to remain reasonably low while the \( P \) and \( M \) have large values.

Up to now, the reduction is only a passband representation. Hence, it needs to be transformed to its baseband form. The baseband dynamic deviation reduction can be extracted from [38] and it can be written as follows.

For \( R = 0 \), there are no dynamic terms and the zero order dynamic deviation reduction reads

\[ y[n] = \sum_{p=1}^{P+1} h^{(p)}[0, \cdots, 0] x[n] x[n]^2 (p-1), \] (3.62)

For \( R = 1 \), only one delay of the input terms can be nonzero. Therefore, the first order dynamic deviation reduction is given by

\[ y[n] = \sum_{p=1}^{P+1} \sum_{m=0}^{M} g^{(2p-1,1)}[m] x[n] x[n]^2 (p-1) x[n - m] \]
\[ + \sum_{p=2}^{P+1} \sum_{m=1}^{M} g^{(2p-1,2)}[m] x^2[n] x[n] x[n - m]^2 (p-2), \] (3.63)
where \( g^{(2j-1,i)}[\cdot] \) denotes the complex Volterra kernel of the system. This first order dynamic deviation reduction model is demonstrated to have an excellent performance with a small number of parameters when linearizing PAs [35][39].

Finally, for \( R = 2 \) the output of the equivalent model is

\[
y[n] = \sum_{p=1}^{M} \sum_{m_1=0}^{M} g^{(2p-1,1)}[m_1] |x[n]|^{2(p-1)} x[n-m_1]
\]

\[
+ \sum_{p=2}^{M} \sum_{m_1=1}^{M} \sum_{m_2=m_1}^{M} g^{(2p-1,3)}[m_1, m_2] x^*[n] |x[n]|^{2(p-2)} x[n-m_1] x[n-m_2]
\]

\[
+ \sum_{p=2}^{M} \sum_{m_1=0}^{M} \sum_{m_2=1}^{M} g^{(2p-1,4)}[m_1, m_2] |x[n]|^{2(p-2)} x[n-m_1] x^*[n-m_2]
\]

\[
+ \sum_{p=3}^{M} \sum_{m_1=1}^{M} \sum_{m_2=m_1}^{M} g^{(2p-1,5)}[m_1, m_2] x^3[n] |x[n]|^{2(p-3)} x^*[n-m_1] x^*[n-m_2].
\]

(3.64)

More details of the derivation can be found in [32]. For complexity considerations, \( R \) should be a small value, typically \( R \leq 2 \).

### 3.2.3.1 Parameter Estimation

Since this dynamic deviation reduction keeps the property of linearity in parameters of the model, LS estimation can be used for the parameter extraction under the assumption of stationarity. Therefore, the model can be written in matrix form as

\[
y = U \hat{h},
\]

(3.65)

where \( \hat{h} \) is the vector that contains all the unknown model parameters \( h^{(k)}[0, \cdots, 0, i_1, \cdots, i_r] \) and \( U \) is the matrix that include all the product terms of the model for a block of \( N \) input samples. The matrix \( U \) has the structure as in Eq. (3.55). However, \( \underline{u}[n]_{pM} \) has to be modified to match the model output. Depending on the dynamic deviation reduction order \( R \) the output of the model will be given by Eq. (3.62), Eq. (3.63) or Eq. (3.64). For instance, in the case of \( R = 1 \) Eq. (3.59) becomes Eq. (3.63), so that

\[
\underline{u}[n]_{1M} = \begin{bmatrix} x[n] & x[n-1] & \cdots & x[n-M] \end{bmatrix}^T,
\]

(3.66)
\[ u[n]_{3M} = \left[ |x[n]|^2 x[n] |x[n]|^2 x[n-1] \cdots |x[n]|^2 x[n-M] \right]^T, \]  
\[ u[n]_{5M} = \left[ |x[n]|^4 x[n] |x[n]|^4 x[n-1] \cdots |x[n]|^4 x[n-M] \right]^T. \]

As Eq.(3.67) and Eq.(3.68) already indicate, for nonlinearity order \( p \text{ odd} \geq 3 \) reads
\[ u[n]_{pM} = \left[ |x[n]|^{p-1} x[n] |x[n]|^{p-1} x[n-1] \cdots |x[n]|^{p-1} x[n-M] \right]^T. \]

Following the example of \( R = 1 \), the vector \( \mathbf{h}^{(p)} \) can be expressed as
\[ h^{(1)} = \left[ g^{(1,1)[0]} g^{(1,1)[1]} \cdots g^{(1,1)[M]} \right]^T, \]
and for nonlinearity order \( p \text{ odd} \geq 3 \) \( h^{(p)} \) is
\[ h^{(p)} = \left[ g^{(p,1)[0]} g^{(p,1)[1]} \cdots g^{(p,1)[M]} \right. \]
\[ \left. g^{(p,2)[1]} \cdots g^{(p,2)[M]} \right]^T. \]

Then, the LS solution of Eq. (3.65) is
\[ h_{LS} = (U^H U)^{-1} U^H y. \]

### 3.2.4 Decomposed Piecewise Volterra Series Model

In envelope tracking (ET) systems the PA behavior extremely depends on the input level, so that it shows very different characteristics at different levels. Therefore, it is difficult to model the behaviour of those systems by using a single function for the whole range of input levels \([40][41]\).

In [1], a new approach for the modeling of ET systems is proposed which decomposes the input signal into several sub-signals that are latter processed separately by dynamic deviation reduction based Volterra Series and finally recombined to produce the output waveform. This technique for signal decomposition developed in [1] is called vector
threshold decomposition. It is an extension of the real-valued threshold decomposition proposed in [42] [43] to the complex domain.

This approach requires a vector of a certain number $S$ of decomposition thresholds, which is defined as

$$\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_S\},$$  \hspace{1cm} (3.73)

where the $\lambda_s$, $s \in \{1, 2, \ldots, S\}$, represent the threshold magnitudes for the input. They are real-valued positive and satisfy $\lambda_1 < \lambda_2 < \cdots \lambda_S$. The input signal $x[n]$ is decomposed with respect to its magnitude, thus the threshold $\lambda_s$ defines the radius of the $s$th threshold circle on the complex I/Q plane, so that the signal space is divided into $S+1$ zones as shown in Fig. 3.1.

![Diagram](image)

**Figure 3.1:** An example for $S = 2$ in which two input samples are depicted. The values of the input signal $x[n]$ are decomposed in three sub-signals that correspond to the vector threshold decomposition zones.

The original signal $x[n]$ is divided into $S+1$ sub-signals $x_s[n]$ located in the corresponding region. Hence, the sub-signal $x_s[n]$ is given by

$$x_s[n] = \begin{cases} 
0, & |x[n]| \leq \lambda_{s-1} \\
(x[n] - \lambda_{s-1})e^{j\varphi}, & \lambda_{s-1} < |x[n]| \leq \lambda_s \\
(\lambda_s - \lambda_{s-1})e^{j\varphi}, & |x[n]| > \lambda_s 
\end{cases}$$  \hspace{1cm} (3.74)
with the assumption of $\lambda_0 = 0$ and $1 \leq s \leq S + 1$. Moreover, $\varphi$ is the phase of the input signal $x[n]$. For a more detailed look of the decomposition, a couple of illustrative examples will be considered. Let the threshold vector be given as $\Lambda = [0.3, 0.8]$. Thus, the input space is divided into three zones $\text{Zone}_1 = \{ x[n] : 0 \leq |x[n]| \leq 0.3 \}$, $\text{Zone}_2 = \{ x[n] : 0.3 < |x[n]| \leq 0.8 \}$ and $\text{Zone}_3 = \{ x[n] : 0.8 < |x[n]| < \infty \}$. Accordingly a particular signal value such as $x[1] = 1.2e^{j\frac{\pi}{6}}$ is decomposed into the three sub-samples $x_1[1] = 0.3e^{j\frac{\pi}{6}}$, $x_2[1] = 0.5e^{j\frac{\pi}{6}}$ and $x_3[1] = 0.4e^{j\frac{\pi}{6}}$. From this, it can be observed that the magnitude of the sample of the first sub-signal $x_1[1]$ is equal to $\lambda_1$, the magnitude of the sample of the second sub-signal corresponds to the radius difference between $\text{Zone}_1$ and $\text{Zone}_2$, i.e., $\lambda_2 - \lambda_1$, and finally, the magnitude of the third sub-signal is equal to the substraction of the the threshold $\lambda_2$ from the original magnitude value, i.e., $|x[1]| - \lambda_2$.

As the decomposition is performed with respect to the magnitude, the phase of the sub-signals is the same as the original phase $\varphi$. If we consider a value that does not reach the higher zone, the sub-signal corresponding to this zone is set to zero. For instance, $x[2] = 0.5e^{j\frac{\pi}{6}}$ can be decomposed into $x_1[2] = 0.3e^{j\frac{\pi}{6}}$, $x_2[2] = 0.2e^{j\frac{\pi}{6}}$ and $x_3[2] = 0$.

Obviously, the sum of the magnitude of the sub-signals have the same length as the magnitude of the original signal samples. The original sample can be recombined by adding all of sub-signals, i.e.

$$x[n] = \sum_{s=1}^{S+1} x_s[n]. \quad (3.75)$$

Fig. 3.2 gives an overview of the decomposition and recombination processes for a signal $x[n]$ and a set of two thresholds $\tau = \{\lambda_1, \lambda_2\}$.

Notice that for $N$ observations the decomposed sub-signals can be represented in matrix form as

$$X = \begin{bmatrix} x_1^T & x_2^T & \cdots & x_{S+1}^T \end{bmatrix}, \quad (3.76)$$

where $x_s, s \in \{1, 2, \ldots, S + 1\}$, is the vector of $N$ samples of the sub-signal $x_s[n]$.

After the decomposition, the sub-signals can be individually processed by the sub-models $G_s(.)$. Each sub-model characterizes the distortion and memory effects of the PA in the corresponding zone. Although different model classes can be used as sub-model in each region, in this thesis, dynamic deviation reduction based Volterra models are employed as sub-models for all of them. Then, the recombination of the overall output $y[n]$ is given by

$$y[n] = \sum_{s=1}^{S+1} G_s(x_s[n]). \quad (3.77)$$
By employing the first order dynamic deviation reduction described in Eq. (3.63), the output reads

\[
y[n] = \sum_{s=1}^{S+1} \sum_{p=1}^{\left\lfloor \frac{P_s+1}{2} \right\rfloor} \sum_{m=0}^{M} g_s^{(2p-1,1)}(m)[x_s[n]]2^{(p-1)}x_s[n-m] \\
+ \sum_{s=1}^{S+1} \sum_{p=1}^{\left\lfloor \frac{P_s+1}{2} \right\rfloor} \sum_{m=1}^{M} g_s^{(2p-1,2)}(m)x_s^2[n]x_s[n]2^{(p-2)}x_s^*[n-m].
\] (3.78)

As shown in Fig. 3.3, the signal processing line can be separated into three steps:

1. Decomposition of the input signal \(x[n]\) into sub-signals \(x_s[n]\),
2. Individual processing of each sub-signal,
3. Recombination of the overall output signal \(y[n]\)

Since the overall model is a linear combination of the sub-models, it is compulsory for the sub-models to be linear with respect of its parameters to retain the property of linearity.

Since nonlinear effects are relatively small within each magnitude zone of the input signal, the nonlinearity order of each sub-model \(P_s\) can be set to a small value. By this, the number of parameters of the overall model can be kept reasonably low, which reduces the computational complexity of the model.
3.2.4.1 Parameter Estimation

Due to the overall system retains the property of linearity with respect to its parameters, Eq. (3.78) can be rewritten as

\[ y = U h. \]  \hspace{1cm} (3.79)

where \( U \) is the matrix including all the product terms appearing in the model and \( h \) is the vector that contains the unknown parameters of the overall model. Although the model is composed by multiple sub-models, the system can be estimated by using only one LS estimator as proposed in [1].

The vector \( h \) is composed by the parameter vector of the sub-models \( h_s \) and reads

\[ h = \begin{bmatrix} h_{1T}^T & h_{2T}^T & \cdots & h_{S+1T}^T \end{bmatrix}^T, \]  \hspace{1cm} (3.80)

where \( h_s \) is the kernel vector of the first order dynamic deviation reduction of the sub-model \( s \) and is described as

\[ h_s = \begin{bmatrix} h_s^{(1)T} & h_s^{(3)T} & \cdots & h_s^{(P)T} \end{bmatrix}^T, \]  \hspace{1cm} (3.81)

and \( h_s^{(p)} \) is given by Eq. (3.70) and Eq. (3.71).

Similarly, the matrix \( U \) contains all the product terms of the \( S + 1 \) input sub-signals of the training set and reads

\[ U = \begin{bmatrix} U_1 & U_2 & \cdots & U_{S+1} \end{bmatrix}, \]  \hspace{1cm} (3.82)
where $U_s$ is the matrix corresponding to the $s$th sub-signal. As first order dynamic deviation reduction is employed as sub-model and taking $N$ input samples, $U_s$ can be written as

$$U_s = \left[ \begin{array}{c} u[0]_s \\ u[1]_s \\ \vdots \\ u[N-1]_s \end{array} \right]^T;$$

with $u[n]_s = \left[ \begin{array}{c} u[n]^{T}_{s,1M} \\ u[n]^{T}_{s,3M} \\ \vdots \\ u[n]^{T}_{s,PM} \end{array} \right]^T,$

and the vector which contains the $p$th-order product terms of the delayed inputs $u[n]_s, pM$ is described by Eq. (3.66) and Eq. (3.69).

Finally, for $N$ observations of the input signal $x[n]$ and the output signal $y[n]$, the parametrized reference model can be obtained by applying the LS estimator to Eq. (3.79)

$$h_{LS} = (U^H U)^{-1} U^H y$$

(3.84)
Chapter 4

Results

4.1 Introduction

This chapter explores the performance of the models presented in Chapter 3 through several simulations in MATLAB. The process to follow is the same in every model. First, the parameters for the reference model are extracted using floating point arithmetic. For solving an equation of the form $C = (A^H A)^{-1} A^H B$ as in Least Squares estimator (used in the parameter extraction), in MATLAB backslash operator is commonly used ($C = (A' A) \backslash (A' B)$). From the point of view of numerical accuracy, backslash operator \ is a better way to perform the solution than the inverse function ($C = \text{inv} (A' A) * (A' B)$) [44].

After obtaining these model parameters, the signal is processed by the parametrized reference model and then the fixed-point model is adaptively estimated. These last two steps are implemented in both, floating point and fixed-point arithmetic, so that the performance of the two implementations can be compared.

4.2 Adaptive Estimation

In practice the PA is expected to vary over time, therefore adaptive system identification is required. For this purpose, the so-called $\epsilon$-NLMS is employed [45]. RLS would also be a right option, but since we are interested in low complexity algorithms, the LMS algorithm is chosen. As $\epsilon$-NLMS is simply a variant of the LMS (see Sec. 2.2.2), it will be described starting from Eq. (2.5).
First of all, the step size $\mu$ is replaced by a data dependent step size $\mu(k)$ to obtain the normalized LMS (NLMS). Thus, the step-size factor $\mu$ is scaled by the reciprocal of the squared norm of the current data sample

$$\mu(k) = \frac{\mu}{\|x_k\|^2}, \quad (4.1)$$

where $x_k$ denotes the corresponding input for each model.

By this modification, the algorithm avoids that the “amount” of the update direction vector is proportional to the power of $x_k$. A second change is required in order to avoid division by zero, a small positive regularization constant $\epsilon$ is added to the normalization factor in Eq. (4.1)

$$\mu[n] = \frac{\mu}{\epsilon + \|x_k\|^2}. \quad (4.2)$$

Inserting Eq. (4.2) in Eq. (2.5) leads to the update equation of $\epsilon$-NLMS

$$\hat{w}_k = \hat{w}_{k-1} + \frac{\mu}{\epsilon + \|x_k\|^2} x_k^* [d_k - x_k^T \hat{w}_{k-1}]. \quad (4.3)$$

The convergence condition on the step-size factor $\mu$ for the $\epsilon$-NLMS becomes independent and reads

$$0 < \mu < 2. \quad (4.4)$$

### 4.3 Reference Models

In order to obtain the parameters for the reference model, the PA was considered constant throughout the process. The parameter extraction procedure estimates the model parameters based on $N$ input samples and its corresponding output samples of the PA. For performing such estimation, pairs of input/output sample are required, which can be obtained by measuring the output of a real PA. In lack of measurement data, a numeric PA model can be used to generate the required training sequences. In this thesis, the latter approach is chosen. The two configurations, with and without memory, and the numeric model used will be described in this section.
4.3.1 Memoryless Configuration

In order to evaluate the fixed-point algorithms using memoryless polynomials (see Sec. 3.1), a nonlinearity without memory effects is used as reference to represent the behavior of the PA. For this purpose, a model proposed by Saleh in [46] is employed.

The model is represented by two formulas $A(r)$ and $\phi(r)$, where $r(t)$ is the envelope of the input signal of the model. $A(r)$ is an odd function of $r$ and represents the AM-AM conversion. $\phi(r)$ is an even function of $r$ and represents the AM-PM conversion. $A(r)$ and $\phi(r)$ are given by

$$A(r) = \frac{\alpha_A r}{1 + \beta_A r^2}, \quad (4.5)$$

$$\phi(r) = \frac{\alpha_\phi r^2}{1 + \beta_\phi r^2}. \quad (4.6)$$

As showed in [46], these formulas were tested with experimental data of the Traveling Wave-Tube (TWT) amplifier from Berman-Mahle. The optimum parameters to fit this TWT amplifier are

<table>
<thead>
<tr>
<th>Parameter</th>
<th>$\alpha_A$</th>
<th>$\beta_A$</th>
<th>$\alpha_\phi$</th>
<th>$\beta_\phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>1.9638</td>
<td>0.9945</td>
<td>2.5293</td>
<td>2.8168</td>
</tr>
</tbody>
</table>

Table 4.1: Optimum parameters of Saleh Model for the TWT amplifier from Berman-Mahle.

The transfer function of the model represented by this parameters is shown in Fig. 1.1.

4.3.2 Hammerstein Configuration

For the fixed-point algorithms using a nonlinear adaptive model with memory of the cases (see Sec. 3.2), the reference model needs to behave as a nonlinearity with memory. A simple memory nonlinearity is the Hammerstein model, which consist of a nonlinearity followed by a FIR filter [7], i.e. memory effects and nonlinearity are modeled separately. The output waveform of the Hammerstein model is given by

$$y[n] = \sum_{i=0}^{M} h[i] v[n-i], \quad (4.7)$$
where $h[i]$ is the impulse response of the FIR filter and $v[n]$ is the output of the nonlinearity. In this configuration the Saleh Model with parameters from Table 4.1 will be employed for the nonlinearity function. Hence, the waveform $v[n]$ reads

$$v[n] = A(r[n]) * e^{j(\angle x[n] + \phi(r[n]))},$$  \hspace{1cm} (4.8)

with $r[n]$ being the magnitude of the input signal $x[n]$, i.e., $r[n] = |x[n]|$. Fig. 4.1 shows an example of a Hammerstein model with signal FIR filter taps from the uniform distribution: $h[0] = 0.8527 + j0.5885$, $h[1] = 0.1222 + j0.5242$, $h[2] = 0.3727 + j0.8756$ and $h[3] = 0.6516 + j0.4327$.

![Hammerstein Model](image)

**Figure 4.1:** Sample characteristics of a PA represented by a Hammerstein model with a FIR filter of four random taps and a nonlinearity represented by the parameters of Table 4.1.

### 4.4 Memoryless Polynomial

In order to evaluate the performance of the model discussed in Sec. 3.1, the training data generated by the memoryless configuration was used. For the parameter extraction, the nonlinearity order of the polynomial model is chosen to $P = 4$. The result of extracting the parameters by applying the LS estimator to $3 \times 10^4$ observations of the training data, as explained in Sec. 3.1.1, is shown in Fig. 4.2.
Although the case of interest for the thesis is the performance in fixed-point arithmetic, the algorithms have been implemented also in floating point as mentioned in the abstract. The algorithm is performed for a signal of length $N = 2 \cdot 10^6$ samples. A white noise signal is used as an input. Since the estimation of the parametrized reference model has been performed in floating point, the parameters are now quantized with the word length of 31 bits and fraction length of 20 bits. Table 4.2 contains the settings of the fixed-point objects used in the simulation. The reference length used for time improvement is $M_1 = 10^4$.

<table>
<thead>
<tr>
<th>Word Length</th>
<th>Fraction Length</th>
<th>RoundMode</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 bits</td>
<td>20 bits</td>
<td>nearest</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OverflowMode</th>
<th>SumWordLength</th>
<th>SumFractionLength</th>
</tr>
</thead>
<tbody>
<tr>
<td>wrap</td>
<td>Word Length</td>
<td>Fraction Length</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SumMode</th>
<th>ProductWordLength</th>
<th>ProductFractionLength</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpecifyPrecision</td>
<td>Word Length</td>
<td>Fraction Length</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ProductMode</th>
<th>SpecifyPrecision</th>
</tr>
</thead>
</table>

Table 4.2: Settings (see Sec.2.1) for the fi objects used in the fixed-point simulation.
In Fig. 4.3, the AM/AM and AM/PM conversions of the numeric model (Saleh model), the model adapted to the parametrized reference one and the model adapted to the numeric model are displayed. It can be observed that the model adapted to the parametrized reference model practically fits the numeric model, however, the adaptation to the numeric model is not as good as the first adaptation, as expected. Moreover, in a purely visual comparison both implementations have a similar performance. Fig. 4.4 shows the instantaneous error of both adaptations for floating point and fixed-point implementation. These figures confirm that the adaptation of the memoryless polynomial model to a real PA, i.e. the numeric model, is worse than the adaptation to the parametrized model and therefore the error that we can expect is greater. In the figures, a line representing the mean of the squared error in the steady-state of the adaptation it has been included and its value, along with the MSE, can be found in Table 4.3.

**Figure 4.3:** AM/AM and AM/PM conversions of: the numeric model (blue), model adapted to the reference model (red) and model adapted to numeric model (green). Floating point implementation (left) and fixed-point implementation (right).

**Figure 4.4:** Performance in terms of instantaneous error for floating point (left) and fixed-point (right) implementations of memoryless polynomial model adaptation to the parametrized reference model (blue) and to the numeric model (red). Dashed lines represent the mean of the squared error in steady-state.
Due to the finite precision of fixed-point, the smallest number representable by a fixed-point object is given by $2^{-\text{FractionLength}}$. With the settings of Table 4.2, this number is $9.5367 \cdot 10^{-7}$, which translates to -120 dB. Adding one bit to the fraction part halves that value, which reduces the squared error by the factor $\frac{1}{4}$. In terms of decibels, this change is -6 dB. If the precision only affected to the lower error representable, Fig. 4.4 (right) would display a fine line where the error reaches -120 dB. However, this finite precision does not only affect to the minimum error representable, but also to the correction term of the update in Eq. (2.5), with $e_k = d_k - x_k^T \hat{w}_{k-1}$. Thus, the amount of the update direction might be larger than desired in the steady-state. Beyond that, the behaviour of both implementations is similar. In Table 4.3 it is observed that for the adaptation to the real PA, the results obtained are the same. In the fixed-point implementation, with the same settings, the squared error in steady-state that we can expect for the adaptation to the numeric model is 7 orders larger than the error of the adaptation to the parametrized reference model. In general, this algorithm is a good choice for modeling memoryless or quasi-memoryless (see Sec. 3.1) PAs.

### 4.5 Memory Polynomial

The Hammerstein configuration is employed to generate the training data in order to evaluate the performance of Memory Polynomials (see sec. 3.2.1). For this configuration, the length of the FIR filter is established to $m = 3$. These filter taps are generated randomly using the uniform distribution. The polynomial order is set to $P = 3$ with memory depth $M = 3$. The parameter extraction is performed in floating point by means of LS solution (See sec. 3.2.1.1) and it result is displayed in Fig. 4.5.

<table>
<thead>
<tr>
<th>Implem.</th>
<th>Steady-state$_1$</th>
<th>Steady-state$_2$</th>
<th>MSE$_1$</th>
<th>MSE$_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed</td>
<td>-98.35 dB</td>
<td>-28.74 dB</td>
<td>-46.18 dB</td>
<td>-28.67 dB</td>
</tr>
<tr>
<td>Floating</td>
<td>-302.43 dB</td>
<td>-28.74 dB</td>
<td>-46.18 dB</td>
<td>-28.67 dB</td>
</tr>
</tbody>
</table>

Table 4.3: Data results for the memoryless polynomial model. Sub-index 1 is used for the adaptation to the reference parametrized model and sub-index 2 refers to the adaptation of the memoryless polynomial model to the numeric model.
Chapter 4. Results

55

Figure 4.5: Estimation of the parametrized reference model by means of the LS estimator applied to the training data. Memory polynomial with $P = 4$ and $M = 3$.

The algorithm is executed for a signal of length $N = 3 \cdot 10^6$ samples. A white noise signal filtered by a root-raised-cosine (RRC) filter is employed as input of the system. The parameters of the RRC filter can be found on Table 4.4. Moreover, the configuration of the objects for the fixed-point implementation is shown in Table 4.2. As explained before (see Sec. 4.4), the parameters of the parametrized reference model have to be quantized with the wordlength and fractionlength of the implementation (see Table 4.2). The reference length used for time improvement is $M_1 = 10^4$.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Order</th>
<th>Cut-off freq.</th>
<th>Sampling freq.</th>
<th>roll-off factor</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>6</td>
<td>10 Ghz</td>
<td>100 Ghz</td>
<td>0.5</td>
<td>Square root</td>
</tr>
</tbody>
</table>

Table 4.4: Configuration of the RRC filter used for generating the input.

As in memoryless polynomials, two adaptations have been performed: the adaptation using the parametrized reference model and the adaptation to the numeric PA model, i.e. Hammerstein configuration (see Sec. 4.3.2). Hence, the AM/AM and AM/PM conversions of both adaptations are shown in Fig. 4.6. It can be seen that the adaptation to the numeric model is more disperse and has a bigger offset in the AM/PM conversion. The instantaneous error of both adaptations in each implementation is displayed in Fig. 4.7. As anticipated, the error that can be expected for the adaptation to the numeric model is greater than the error for the adaptation to the reference parametrized model.
As in the previous section, a dashed line representing the mean of the squared error in steady-state is displayed also in Fig. 4.7. Moreover, the value of this mean alongside the MSE for each adaptation can be found in Table 4.5.

Table 4.5: Data results for the memory polynomial model. Sub-index 1 is used for the adaptation to the parametrized reference model and sub-index 2 refers to the adaptation of the memory polynomial model to the numeric model.

<table>
<thead>
<tr>
<th>Implem.</th>
<th>Steady-state&lt;sub&gt;1&lt;/sub&gt;</th>
<th>Steady-state&lt;sub&gt;2&lt;/sub&gt;</th>
<th>MSE&lt;sub&gt;1&lt;/sub&gt;</th>
<th>MSE&lt;sub&gt;2&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed</td>
<td>-101.69 dB</td>
<td>-42.43 dB</td>
<td>-54.77 dB</td>
<td>-42.13 dB</td>
</tr>
<tr>
<td>Floating</td>
<td>-302.63 dB</td>
<td>-42.43 dB</td>
<td>-54.77 dB</td>
<td>-42.13 dB</td>
</tr>
</tbody>
</table>

As explained in Sec. 3.2.1.1, the finite precision of the fixed-point objects has effect on the minimum error representable (-120 dB) and on the correction term of the update.
Therefore, in fixed-point implementation the length of the update direction might be larger than desired in the steady-state of the adaptation to the parametrized reference model, which leads to a fluctuation of the error in steady-state instead of being represented by a line at the lower limit. Aside from this, the behaviour is similar. By observing Fig. 4.7 and Table 4.5, it is noticed that the behaviour of the adaptation to the numeric model is the same in both implementations. With the same settings for both adaptations, in the fixed-point implementation, the squared error in steady-state that we can expect for the adaptation to the numeric PA is 5 orders larger than the error for the adaptation to parametrized model. From now on, all the models will include memory effects, so the models can be compared as they are studied.

4.6 Truncated Volterra Series

In order to assess the Volterra Series model (see Sec. 3.2.2), Hammerstein configuration is considered as reference model. As in the previous model, in the Hammerstein configuration the length of the FIR filter is set to $m = 3$. The settings of the model, which have been chosen by means of experimentation, are nonlinearity order $P = 2$ and memory depth $M = 3$. The parameter extraction is performed by LS estimator as explained in Sec. 3.2.2.4. For the passband case, the result of the extraction is shown in Fig. 4.8.

![Figure 4.8: Estimation of the parametrized reference model by means of the LS estimator applied to the training data. Passband Volterra Series with $P = 2$ and $M = 3$.]
A RRC filtered white noise signal of length $N = 3 \cdot 10^6$ samples is used as input of the system. The configuration of the RRC filter is the same as in Sec. 4.5 and it is displayed on Table 4.4. The settings for the fixed-point objects are shown in Table 4.2. The coefficients of the parametrized reference model have to be quantized in order to employ them in the fixed-point implementation. The wordlength and fractionlength used for the quantization is the same than the rest of fixed-point objects (see Table 4.2). The reference length used in fixed-point implementation is $M_1 = 10^4$.

The AM/AM and AM/PM conversions of the numeric PA model (Hammerstein model), the adaptation to the reference parametrized model and the adaptation to numeric PA model for both implementations are shown in Fig. 4.9. It can be seen that the adaptation to the numeric model is more disperse and has a bigger offset in the AM/PM conversion. Fig. 4.10 displays the instantaneous errors of both adaptations in each implementation. As in previous models, the adaptation to the numeric model has a greater error to be expected in both implementations. Table 4.6 shows the value of the squared error in steady-state and the MSE for each adaptation in both implementations. The finite precision of fi objects not only affects to the minimum number representable (-120 dB), but it also does to the update term (see Eq. (2.5)). Hence, it may happen that this term is larger than needed in the steady-state stage, which entails an oscillation of the instantaneous error in this stage. Besides, the behaviour in both implementations is similar as seen in Fig. 4.9. It can be inferred from Table 4.6 that the adaptation to a numeric PA model behaves the same in both implementations. Moreover, in fixed-point the squared error in steady-state that can be expected for the adaptation to a numeric PA model is 7 orders larger than the error for the adaptation to the parametrized reference model.

Figure 4.9: AM/AM and AM/PM conversions of: the numeric model (blue), model adapted to the reference model (red) and model adapted to numeric model (green). Floating point implementation (left) and fixed-point implementation (right).
Figure 4.10: Performance in terms of instantaneous error for floating point (left) and fixed-point (right) implementations of passband truncated Volterra Series adaptation to the parametrized reference model (blue) and to the numeric model (red). Dashed lines represent the mean of the squared error in steady-state.

<table>
<thead>
<tr>
<th>Implem.</th>
<th>Steady-state$_1$</th>
<th>Steady-state$_2$</th>
<th>MSE$_1$</th>
<th>MSE$_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed</td>
<td>-96.06 dB</td>
<td>-22.96 dB</td>
<td>-76.13 dB</td>
<td>-22.94 dB</td>
</tr>
<tr>
<td>Floating</td>
<td>-301.63 dB</td>
<td>-22.96 dB</td>
<td>-76.56 dB</td>
<td>-22.94 dB</td>
</tr>
</tbody>
</table>

Table 4.6: Data results for passband truncated Volterra Series implementation. Sub-index 1 is used for the adaptation to the parametrized reference model and sub-index 2 refers to the adaptation of the passband truncated Volterra Series model to the numeric model.

4.6.1 Baseband Volterra model

For the baseband case, with the same Hammerstein configuration like for the passband model, the nonlinearity order of the model is set to $P = 3$ and the memory depth is set to $M = 2$. These settings have been chosen by a simulation-based approach. The results of parameter extraction are shown in Fig. 4.11.
Chapter 4. Results

Figure 4.11: Estimation of the parametrized reference model by means of the LS estimator applied to the training data. Baseband Volterra Series with $P = 3$ and $M = 2$.

For the input signal, a root-raised cosine filtered white noise signal of length $N = 3 \cdot 10^6$ samples is employed. The configuration of the RRC filter is displayed on Table 4.4. The settings of the fixed-point objects for the fixed-point implementation are shown in Table 4.2. The extraction of the parameters of the reference model has been performed in floating point, therefore the parameters have to be quantized with the wordlength and fraction length of the fixed-point objects (see Table 4.2) in order to use them in the fixed-point implementation. The reference length employed for time improvement in fixed-point implementation is $M_1 = 10^4$.

Two adaptations have been performed in both implementations: the adaptation to the reference parametrized model and the adaptation to numeric PA model (Hammerstein model). The AM/AM and AM/PM conversions of both adaptations for each implementation are shown in Fig. 4.12. It can be observed that the adaptation to the numeric model is not as good as the adaptation to the parametrized reference model. Moreover, Fig. 4.13 displays the instantaneous error corresponding to each adaptation. As predicted by observing Fig. 4.12, the adaptation to the numeric model has a greater error to be expected in both implementations. In addition, the behaviour of the adaptation to the reference model is similar in both implementations. The drawback introduced by the finite precision of the fixed-point implementation is that the correction term of the update (Eq. (2.5)) may be larger than needed in the steady-state situation, so that
the error fluctuates in this stage instead of being a straight line coincident with the lower bound of the finite precision (-120 dB). Table 4.7 lists the value of the mean of the squared error in the steady-state stage and the MSE for each adaptation in both implementations. It shows that the squared error of the adaptation to the Hammerstein model is 8 orders larger than the error of the adaptation to the parametrized reference model.

<table>
<thead>
<tr>
<th>Implem.</th>
<th>Steady-state(_1)</th>
<th>Steady-state(_2)</th>
<th>MSE(_1)</th>
<th>MSE(_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed</td>
<td>-103.25 dB</td>
<td>-18.55 dB</td>
<td>-54.34 dB</td>
<td>-18.14 dB</td>
</tr>
<tr>
<td>Floating</td>
<td>-303.86 dB</td>
<td>-18.55 dB</td>
<td>-54.46 dB</td>
<td>-18.14 dB</td>
</tr>
</tbody>
</table>

Table 4.7: Data results for baseband truncated Volterra series implementation. Sub-index 1 is used for the adaptation to the parametrized reference model and sub-index 2 refers to the adaptation of the baseband Volterra Series model to the numeric model.

Figure 4.12: AM/AM and AM/PM conversions of: the numeric model (blue), model adapted to the reference model (red) and model adapted to numeric model (green). Floating point implementation (left) and fixed-point implementation (right).

Figure 4.13: Performance in terms of instantaneous error for floating point (left) and fixed-point (right) implementations of memory polynomial model adaptation to the parametrized reference model (blue) and to the numeric model (red). Dashed lines represent the mean of the squared error in steady-state.
4.7 Dynamic Deviation Reduction-based Volterra Series

The configuration used for testing this model (see Sec. 3.2.3) is the Hammerstein configuration. The length of the FIR filter is set to $m = 3$. For the parameter extraction, the nonlinearity order is set to $P = 3$, the memory depth to $M = 3$ and the dynamic truncation order to $R = 1$. As in the previous model, LS estimator is employed for extracting the parameters of the model and the result of this process is displayed in Fig. 4.14.

![Numeric model and Reference parametrised model](image)

**Figure 4.14:** Estimation of the parametrized reference model by means of the LS estimator applied to the training data. First order dynamic deviation reduction model with $P = 3$ and $M = 3$.

The simulation is performed for a white noise input signal filtered by a RRC filter (configuration displayed in Table 4.4) of length $N = 3 \cdot 10^6$ samples. In order to perform the fixed-point implementation, the parameters for the fi objects of the system have to be chosen. Table 4.2 contains the settings of the fi object. Since the parameter extraction by LS estimator has been performed in floating-point, the parameters have to be quantized with the word length and fraction length of the fi objects (31 bits and 20 bits, respectively). As commented in Sec. 2.2.1, for fixed-point implementation a reference length is needed. The reference length used is $M_1 = 10^4$.

In Fig. 4.15, the AM/AM and AM/PM conversions of the numeric model (Hammerstein configuration), the model adapted to the parametrized reference model and the
model adapted to the numeric model. The figure clearly shows that the adaptation to the numeric model has a worse performance than the adaptation to the parametrized reference model. In addition, it is observed that the adaptation to the parametrized reference model behaves similar in both implementations. Fig. 4.16 shows the squared error of both adaptations in each implementation. As observed before, the error of the adaptation to the numeric PA model is greater than the error of the adaptation to the parametrized reference model. In reference to the adaptation to the parametrized reference model, it may seem that the steady-state level is reached earlier in fixed-point, but, actually, the error in fixed-point has a lower bound imposed by the finite precision of the fi objects. The value of this level, as developed in Sec. 4.4, is -120 dB. Moreover, this limit of the lower number representable also affects to the correction term of the weight update equation (Eq. (2.5)). The value of the correction term may be larger than desired caused by that limitation. The dashed lines displayed in Fig. 4.16 represent the mean of the squared error in steady-state for each adaptation. The value of these means, together with the value of the MSE, is contained in Table 4.8. It shows that the squared error of the adaptation to the numeric model is 8 orders larger than the error of the adaptation to the parametrized reference model.

![Figure 4.15: AM/AM and AM/PM conversions of: the numeric model (blue), model adapted to the reference model (red) and model adapted to numeric model (green). Floating point implementation (left) and fixed-point implementation (right).](image-url)
Chapter 4. Results

Figure 4.16: Performance in terms of instantaneous error for floating point (left) and fixed-point (right) implementations of DDR model adaptation to the parametrized reference model (blue) and to the numeric model (red). Dashed lines represent the mean of the squared error in steady-state.

Table 4.8: Data results for DDR implementation. Sub-index 1 is used for the adaptation to the parametrized reference model and sub-index 2 refers to the adaptation of the DDR model to the numeric model.
Chapter 4. Results

4.8 Decomposed Piecewise Volterra Series Model

Hammerstein configuration is used to evaluate the Decomposed Piecewise Volterra series model (see Sec. 3.2.4). As in the previous models, in the Hammerstein configuration the length of the FIR filter is set to \( m = 3 \). Let \( \tau \) be the vector threshold \( \tau = 0.3 \). The settings of the first order dynamic deviation reduction sub-models are nonlinearity order \( P = 3 \) and memory depth \( M = 2 \). The parameter extraction is performed by LS estimator as explained in Sec. 3.2.4.1 and its result its displayed in Fig. 4.17.

![Figures showing numeric model and reference parametrised model](image)

**Figure 4.17:** Estimation of the parametrized reference model by means of the LS estimator applied to the training data. Decomposed piecewise Volterra series model with first order dynamic deviation reduction sub-models of parameters \( P = 3 \) and \( M = 3 \).

The floating point simulation is performed for an input signal of length \( N = 15 \cdot 10^7 \) samples. A white noise signal filtered with a RRC filter, whose configuration can be found in Table 4.4, is employed as input for the system. However, for the fixed-point simulation only \( N = 6 \cdot 10^7 \) samples has been used, as they were enough to represent the steady-state. The settings of the \( \text{fi} \) object used for the fixed-point implementation can be seen in Table 4.2. The parameters of the parametrized reference model have to be quantized since the extraction was performed in floating point. The quantization is performed with the word length and fraction length of the fixed-point implementation (Table 4.2). The reference length employed for the time improvement (see Sec.2.2.1) is \( M_1 = 10^4 \).
For each implementation (floating point and fixed-point), two adaptations have been performed: the adaptation to the parametrized reference model and the adaptation to the numeric PA model, i.e., Hammerstein configuration. The AM/AM and AM/PM conversions of the numeric PA model and the two adaptations for both implementations are shown in Fig. 4.18. By looking at the AM/PM conversion, it can be observed that the adaptation to the Hammerstein configuration has an increment of phase offset for larger magnitudes, whereas in the adaptation to the parametrized reference model it does not happen. In addition, Fig. 4.19 displays the instantaneous error of each adaptation. As observed in Fig. 4.18, the error that can be expected in steady-state for the adaptation to the numeric PA model is greater than the error expected for the adaptation to the parametrized reference model. The behaviour of both floating point and fixed-point implementation is similar. The difference is the finite precision introduced by the fixed-point arithmetic. It only has a visible effect in the adaptation to the parametrized reference model because its error reaches the smallest number representable by the fi objects (see Sec. 4.4). This limitation may force the correction term of the LMS update to be larger than required, which leads to a fluctuation of the error in the steady-state stage. Notice that Fig. 4.18 also shows a couple of dashed lines. They represent the mean of the squared error in the steady-state stage for each adaptation. Their values are listed in Table 4.9. It shows that the squared error of the adaptation to the Hammerstein model is 8 orders larger than the error of the adaptation to the parametrized reference model.

**Figure 4.18:** AM/AM and AM/PM conversions of: the numeric model (blue), model adapted to the reference model (red) and model adapted to numeric model (green). Floating point implementation (left) and fixed-point implementation (right).
Figure 4.19: Performance in terms of instantaneous error for floating point (left) and fixed-point (right) implementations of decomposed piecewise Volterra model adaptation to the parametrized reference model (blue) and to the numeric model (red). Dashed lines represent the mean of the squared error in steady-state.

<table>
<thead>
<tr>
<th>Implem.</th>
<th>Steady-state₁</th>
<th>Steady-state₂</th>
<th>MSE₁</th>
<th>MSE₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed</td>
<td>-106.25 dB</td>
<td>-21.33 dB</td>
<td>-60.46 dB</td>
<td>-21.34 dB</td>
</tr>
<tr>
<td>Floating</td>
<td>-304.80 dB</td>
<td>-21.40 dB</td>
<td>-61.83 dB</td>
<td>-21.38 dB</td>
</tr>
</tbody>
</table>

Table 4.9: Data results for decomposed piecewise Volterra Series implementation. Sub-index 1 is used for the adaptation to the parametrized reference model and sub-index 2 refers to the adaptation of the decomposed piecewise Volterra Series model to the numeric model.
Chapter 5

Conclusions and Outlook

5.1 Conclusions

This work considered behavioural modelling algorithms to represent power amplifiers both with and without memory effects. These models can be used to linearize PA through digital pre-distortion. The behavioural models used in this thesis are the following ones:

- Memoryless polynomial
- Memory polynomial
- Model based on truncated Volterra Series - Passband and Baseband representations.
- Dynamic deviation reduction-based Volterra model
- Decomposed piecewise Volterra Series

The main goal of this work was to study the behaviour of these algorithms in fixed-point arithmetic. For this purpose, these in literature proposed DPD algorithms were implemented in MATLAB using the Fixed-Point Toolbox. During this thesis, the Fixed-Point Toolbox provided by MATLAB was studied in order to obtain a correct performance of the simulations. The adaptive estimation of the models were performed by the well-known $\epsilon-NLMS$. The models were tested with a white noise input signal generated by a random input sequence filtered by a root-raised-cosine filter. Since there is no available measured data, a numeric PA model is used to provide training data for testing the algorithms. Two different numeric models are used: a saleh model for the memoryless
configuration and a hammerstein model, composed by a saleh nonlinearity and a FIR filter, for the memory configuration.

For these fixed-point implementations was necessary to find adequate word lengths and fraction lengths which allow the algorithms to have a proper behaviour with respect to the floating point implementations.

The decomposed piecewise Volterra model has a better performance than the no-decomposed models, with a small number of parameters. Whereas the decomposed piecewise Volterra model employs a first order dynamic deviation reduction-based Volterra model with nonlinearity order $P = 3$ and memory depth $M = 2$ as submodel, which makes a total number of 10 parameters (5 parameters each), the first order DDR model with $P = 3$ and $M = 3$ employs 8 parameters itself obtaining a worse adaptation.

Moreover, it is observed (see Table 4.5, Table 4.7 and Table 4.8) that the dynamic deviation reduction-based Volterra model is, in general, more accurate than memory polynomials and truncated Volterra series.

Regarding the complexity, the decomposed piecewise Volterra model presents a higher complexity than the other models. Being $S$ the number of decomposition thresholds, the complexity of this model is $S + 1$ times the complexity of the sub-model, plus the decomposition and recomposition of the signals.

### 5.2 Suggestions for Future Work

This work can be extended with some of the following questions:

- Implementing the Vector-Switched Model proposed by Afsardoost in [47] using the models studied in this thesis as submodels and comparing the performance of the Vector-Switched model with the decomposed piecewise Volterra model.

- Performing tests with different types of commercial power amplifiers and finding the optimum parameters for the decomposed piecewise Volterra.

- Implementing the schemes in hardware and comparing the simulation results with real-time measures, since this thesis has been based exclusively in simulations.

- Synthesizing the DPD, through direct and/or indirect learning architecture [48], and measuring the linearization of a PA with the models presented on this work.
Bibliography


