UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Telecomunicació
i Aeroespacial de Castelldefels

FINAL DEGREE PROJECT

TFG TITLE: Machine learning techniques applied to dimensionality reduction for digital predistortion linearizers

DEGREE: Bachelor's degree in Aerospace Systems Engineering and Telecommunications Systems

AUTHOR: Abdoul Aziz Barry

ADVISOR: Pere Gilabert

DATE: January 13, 2021

Título: Técnicas de aprendizaje automático aplicadas a la reducción de la dimensionalidad para linealizadores de predistorsion digital
Autor: Abdoul Aziz Barry

Director: Pere Gilabert

Fecha: 13 de enero de 2021

Resumen

Durante los ultimos años, las mejoras en la tecnología de vehiculos espaciales se han producido principalmente en las áreas de la microelectrónica, los dispositivos electrónicos de alta frecuencia y los circuitos integrados para las comunicaciones y la navegación, las células solares y las baterías para la generación y el almacenamiento de energía a bordo, entre muchas otras.
A pesar de que las tecnologías de almacenamiento de energía han avanzado espectacularmente en los últimos años, el consumo de energía de las comunicaciones, los sensores y los sistemas de procesamiento de señales digitales a bordo es de suma importancia en los sistemas alimentados por baterías o por energía solar, como los pequeños satélites. Las aplicaciones que implican el uso de estos sistemas son múltiples, por ejemplo, aplicaciones de observación de la tierra, vigilancia, comunicaciones de radiodifusión, investigación científica, etc.
En las comunicaciones inalámbricas, el amplificador de potencia es un subsistema crítico en la cadena de transmisión. No sólo porque es uno de los dispositivos que más energía consume, sino también porque es la principal fuente de distorsión no lineal en el transmisor. Las señales de comunicaciones moduladas en amplitud y fase que presentan una elevada PAPR(peak-to-average power ratio) tienen un impacto negativo en la eficiencia del transmisor, porque el amplificador tiene que funcionar a niveles de retroceso de alta potencia para evitar la introducción de distorsión no lineal.
La linealización mediante predistorsión digital (DPD) es la solución más común y extendida para hacer frente a la linealidad inherente de los amplificadores de potencia (PA) frente a la compensación de la eficiencia. Cuando se consideran señales de banda ancha, los amplificadores de seguimiento de la envolvente o los transmisores de desfase, el número de parámetros necesarios en el modelo DPD para compensar tanto las no linealidades como los efectos de la memoria puede ser muy elevado. Esto tiene un impacto negativo en la extracción de ceoeficientes del DPD, porque aumenta la complejidad computacional y conduce a un exceso de ajuste e incertidumbre. Sin embargo, aplicando técnicas de reducción de la dimensionalidad podemos evitar el mal condicionamiento numérico de la estimación y reducir el número de coeficientes de la función DPD, lo que en última instancia repercute en la complejidad computacional del procesamiento de la banda base y en el consumo de energía.
En este proyecto se describirán y compararán varias técnicas de reducción de la dimensionalidad en términos de capacidad de reducción del orden de los modelos y de su rendimiento. En particular, se estudiarán algunas de las técnicas de aprendizaje automático para la reducción de la dimensionalidad.

Title : Machine learning techniques applied to dimensionality reduction for digital predistortion linearizers

Author: Abdoul Aziz Barry

Advisor: Pere Gilabert

Date: January 13, 2021

Overview

Over the past half century, the improvements in spacecraft technology have been primarily in the areas of microelectronics for on-board processing, high frequency electronic devices, and integrated circuits for communications and navigation, solar cells and batteries for on-board power generation and storage among many others.

Despite the fact that energy-storage technologies have advanced dramatically over the past years, the power consumption of on-board communications, sensors and digital signal processing systems is of paramount importance in battery or solar powered systems such as small satellites, HAPs or UAVs (drones). There is multiple applications that involves the use of these systems, e.g., Earth observation applications, surveillance, broadcast communications, scientific research, etc.

In wireless communications, the power amplifier is a critical subsystem in the transmitter chain. Not only because it is one of the most power hungry devices that accounts for most of the direct current power consumption, but also because it is the main source of nonlinear distortion in the transmitter. Amplitude and phase modulated communications signals presenting high peak-to-average power ratio have a negative impact in the transmitter's power efficiency, because the PA has to be operated at high power back-off levels to avoid introducing nonlinear distortion.

Digital predistortion (DPD) linearization is the most common and spread solution to cope with power amplifiers (PA) inherent linearity versus efficiency trade-off. When considering wide bandwidth signals, such as Doherty PAs, envelope tracking PAs or outphasing transmitters, the number of parameters required in the DPD model to compensate for both nonlinearities and memory effects can be very high. This has a negative impact in the DPD ceofficients extraction, because increases the computational complexity and drives to over-fitting and uncertainty. However, by applying dimensionality reduction techniques we can both avoid the numerical ill-conditioning of the estimation and reduce the number of coefficients of the DPD function, which ultimately impacts the baseband processing computational complexity and power consumption.

In this Project, several dimensionality reduction techniques will be described and compared in terms of model order reduction capabilities and evaluation performance. In particular, some of the machine learning techniques for dimensionality reduction will be studied.

CONTENTS

LIST OF FIGURES

TABLES INDEX

ACKNOWLEDGMENTS

Throughout the writing of this dissertation, I have received a great deal of support and assistance.

I would first like to thank my tutor, Pere Gilabert, whose expertise was invaluable in formulating the research questions and methodology. Your insightful feedback pushed me to sharpen my thinking and brought my work to a higher level.

In addition, I would like to thank my parents for their wise counsel and sympathetic ear. You are always there for me. Finally, I could not have completed this dissertation without the support of my friends, who provided stimulating discussions as well as happy distractions to rest my mind outside of my research.

CHAPTER 1. INTRODUCTION

Artificial satellites are human-built objects orbiting the Earth and other planets in the Solar System. This is different from the natural satellites, or moons, that orbit planets, dwarf planets and even asteroids. Artificial satellites are used to study the Earth, other planets, to help us communicate, and even to observe the distant Universe. Satellites can even have people in them, like the International Space Station and the Space Shuttle. Artificial satellites can have a range of missions, including scientific research, weather observation, military support, navigation, Earth imaging, and communications. Some satellites fulfill a single purpose, while others are designed to perform several functions at the same time.

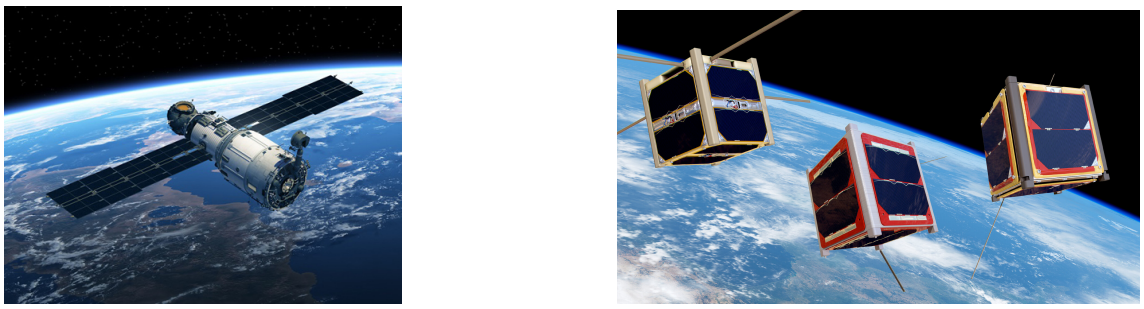

Figure 1.1: Artificial satellites

Artificial satellites can be classified according to the altitude at which they will orbit, or they can be classified according to their mass or size (see figure 1.2). Satellite platform classification based on mass is presented in [1] where satellites with a mass of more than 1000 kg are considered large satellites, satellites between 1000 and 500 kg are considered medium satellites and finally small satellites are all those below 500 kg. Small satellites are divided into minisatellites, microsatellites, nanosatellites, picosatellites and femtosatellites.

The use of small satellites (mass lower than 500 kg), especially nanosatellites and microsatellites, is growing tremendously in the space field, thanks to the development and wide use of the CubeSat standard and all related technology equipment. As an immediate consequence, small satellites have been increasingly proposed to build up distributed space systems for earth observation and science purposes. However, even though the use of this class of spacecraft presents several advantages in terms of mission costs, there are additional constraints such as power and computational capabilities.

In a satellite communication system, main limitations are down link capabilities, especially for small satellites. This is due to the limited amount of energy that can be obtained through photovoltaic cells. A typical small satellite with 50 kg can generate only as small as power of around 160 W as total and can distribute roughly 30 W. This is a power constraint for a high-data-rate communication system for small satellites. Several hundred Mbps down link system on conventional large satellite consume one or more hundreds watt. Therefore an improvement in the efficiency of the satellite components is crucial to reduce the energy consumed and thus improve the transmission rate.

All satellites have a communication system to be able to transmit collected data, therefore a great part of the energy generated by the photovoltaic cells is consumed by the transmitter elements. The main elements of a transmitter are the oscillator, the modulator and the power amplifier, being the latter the one that consumes more energy. The PA are devices

Quantity	Platform mass, kg
Large spacecraft	≥1000
Medium spacecraft	≥500 and <1000
Small spacecraft	<500
Minispacecraft	≥100 and <500
Microspacecraft	≥10 and <100
Nanospacecraft	≥1 and <10
Picospacecraft	≥0.1 and <1
Femtospacecraft	≤0.1

Figure 1.2: Satellite classification based on mass [2]

that convert a low power signal to a signal with a higher power and depending on the gain it has, it can amplify more signal or less. The ideal operation of an amplifier is such that with an input power P_{in}, you will have a power $P_{out} = P_{in} \cdot Gain$ at the output. As the input power increases, the output power increases proportionally. The reality is quite different because from an input power P_{sat}, the PA start presenting non-linear effects that distort the output signal, thus limiting the maximum input power. Therefore, the non-linear effects of the amplifier cause a worsening of the efficiency of the PA resulting in more energy consumption. Over the years, several linearization methods have appeared that try to improve the efficiency of such amplifiers and one of them is the digital predistortion (DPD). DPD is a linearization technique oriented at extending the linear dynamic range in PAs. By using mathematical models such as Volterra series or simplified versions of the Volterra series, it is possible to model the behaviour of the PA and then use it DPD linearization purposes. These parametric behavioral models consist of linear combinations of nonlinear basis functions. The number of basis functions or coefficients in a model is an indicator of its complexity and directly affects the computational load and therefore the consumption of digital processors. One of the main challenges of behavioral modeling is choosing the most relevant basis functions to achieve the right data fitting. In order to avoid over-fitting (i.e., including more coefficients than needed) that yields to an inaccurate (or ill-conditioned) estimation of the model parameters, in this project we will analyze several methods to reduce the complexity of the behavioral models by reducing the number of coefficients. This will not only have a beneficial impact by reducing the computational load and therefore the power consumption of digital signal processors, but also it will introduce regularization effects.

Therefore, this work is divided into 6 chapters and is organized as follows. In Chapter 1 a short introduction and the motivation of this work is given.

Chapter 2 introduces the non-linear behaviour in power amplifiers, the most relevant metrics to measure it and finally, the main linearization techniques oriented at compensating for these unwanted non-linear behavior in PAs.

Chapter 3 discusses some of the most relevant behavioral models in literature to characterize the PA nonlinear behavior and introduces the the extraction of the behavioral models' coefficients.

Chapter 4 presents a detailed analysis of the different model order (or dimensionality) reduction methods under analysis. Some of the specific details will be discussed and

partial results will be given.

In chapter 5 we describe the experimental data and setup used for obtaining the results. In addition, a detailed comparison of the dimensionality reduction techniques described in Chapter 4 is provided in terms of modeling accuracy versus number of required coefficients and running time of the algorithms' search.

Finally, in chapter 6 conclusions are given and some future work to do is envisaged.

CHAPTER 2. LINEARITY VERSUS POWER EFFICIENCY TRADE-OFF IN POWER AMPLIFIERS

2.1. Introduction

A radio frequency power amplifier is a tuned amplifier that amplifies high-frequency signals used in radio communications. When the signal is sent from the transmitter, the power of signal is absorbed by environment, in other words, the signal is attenuated. Therefore, if the transmitted signal is not enough at the the transmitter side, when the signal reached the receiver, it will have a very low power and the receiver may not be able to detect the signal. Therefore, the power of signal have to be increased in the beginning as much as it is possible and this is the main job of the power amplifier. Radio frequency PA are the most power consuming devices in the transmitter block diagram and one of the main sources of nonlinear distortion. Therefore, the PA's efficiency is a key feature to lower the running costs. The PA achieves highest efficiency when operated close to saturation, but also tends to be nonlinear. Therefore, in order to obtain the best efficiency possible, work will be done in the non-linear area of the amplifier. To do this, it will be necessary to understand the non-linear behaviour of the amplifiers and to find a way to quantify the RF PA parameters.

2.2. Power amplifier nonlinear behavior

Non-linearity behaviour is often a big issue, especially when power and frequency are involved. It affects particularly RF Power Amplifiers (PA) where power is high (to drive antenna) and frequencies are highly involved (modulated carriers). Figure 2.1 shows a typical RF transmitter front end in which data are modulated or mixed with a carrier. The created modulated RF signal has to be then amplified with a power amplifier in order to drive the antenna so the signal can be transmitted through the air. The objective is that the PA needs to play its amplification role as perfectly as possible without major distortion and adding noise. In other words we want $RFout$ perfectly proportional to $RFin$ with respect to the signal shape and integrity. Unfortunately all real PA's have a certain degree of non-linearity.

In figure 2.2 a non-linearity is visible when the output behavior ceases to follow the linear line. On an amplifier it can be measured by the gap between the ideal line and the real actual curve. Where the 1 dB compression point is the moment when the ideal and real behaviour differ by 1dB and $Psat$ level is the maximum the output can exhibit regardless of the quantity of input injected, both of these are giving a certain measure of non-linearity of a power amplifier.

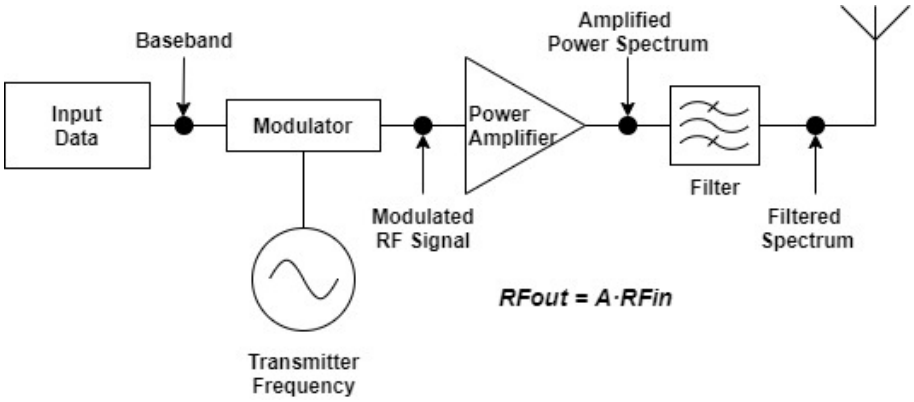

Figure 2.1: Transmitter block diagram

Mathematically a memoryless amplifier can be modeled by a polynomial expression where the output $y(t)$ is a series of terms proportional to the input $x(t)$ and higher order terms.

$$y(t) \approx \sum_{n=1}^{N} g_n \cdot x^n(t) \tag{2.1}$$

When the amplifier is perfectly linear, all the g_n terms other than $n = 1$ are zero so $y(t)$ is proportional to $x(t)$. Non-linearity will be a measure of the presence and intensity of all the g_n terms where n is different than one. When n adopts even values, it adds components at multiples of the carrier frequency (harmonics) of the input signal, this effect is known as *Harmonic Distortion (HD)* and can be mitigated by using the filters. Intermodulation products appear when n is odd, as it causes some components of the frequencies to fall very close to the input signal. These components introduced by the intermodulation products is called *InterModulation Distortion (IMD)*. Some of these components fall directly inside the signal bandwith and generate in-band distortion.

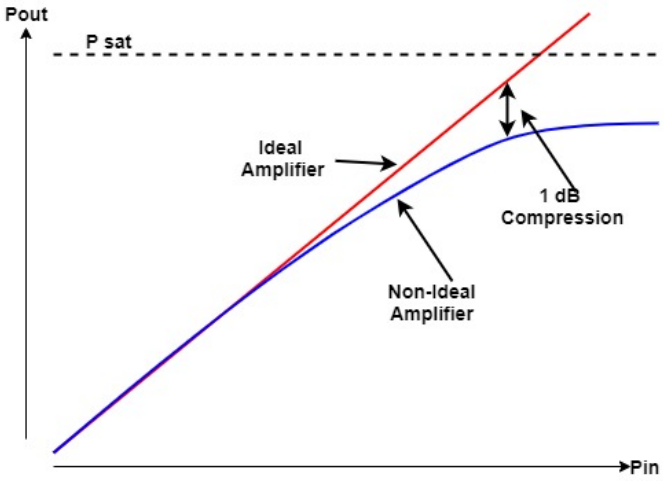

Figure 2.2: Real and ideal power amplifier

To be able to visualize and better understand the distortions caused by terms of order greater than one, a test is discussed in [5] where the PA is fed with two tones separated by Δf in frequency. The input signal is as follows:

$$x(t) = V_1\cos(\omega_1 t) + V_2\cos(\omega_2 t) \tag{2.2}$$

$$\omega_1 = 2\pi(f_c - \frac{\Delta f}{2}),\ \ \omega_2 = 2\pi(f_c + \frac{\Delta f}{2})$$

For simplicity only the first three terms in 2.1 will be considered. The output signal will be

$$\begin{aligned} v_{out}(t) &= \frac{g_2V_1^2}{2} + \frac{g_2V_2^2}{2} + \\ &+ V_1\left[g_1 + \frac{3g_3V_1^2}{4} + \frac{3g_3V_2^2}{2}\right]\cos(\omega_1 t) + \\ &+ V_2\left[g_1 + \frac{3g_3V_2^2}{4} + \frac{3g_3V_1^2}{2}\right]\cos(\omega_2 t) + \\ &+ \frac{g_2V_1^2}{2}\cos(2\omega_1 t) + \frac{g_2V_2^2}{2}\cos(2\omega_2 t) + \\ &+ g_2V_1V_2\left[\cos\left((\omega_2-\omega_1)t\right) + \cos\left((\omega_2+\omega_1)t\right)\right] + \\ &+ \frac{g_3V_1^3}{4}\cos(3\omega_1 t) + \frac{g_3V_2^3}{4}\cos(3\omega_2 t) + \\ &+ \frac{3g_3V_1^2V_2}{4}\left[\cos\left((2\omega_1+\omega_2)t\right) + \cos\left((2\omega_1-\omega_2)t\right)\right] + \\ &+ \frac{3g_3V_2^2V_1}{4}\left[\cos\left((2\omega_2+\omega_1)t\right) + \cos\left((2\omega_2-\omega_1)t\right)\right] \end{aligned} \tag{2.3}$$

It can be observed that the equation shows the effects produced due to the nonlinearity effects of the PA. The distortions caused by PA can affect the spectrum in two ways. By adding frequency components in the carrier frequency (in-band distortion)

- $\omega_1 + \omega_1 - \omega_1 = \omega_1$ and $\omega_2 + \omega_2 - \omega_2 = \omega_2$
- $\omega_1 + \omega_2 - \omega_2 = \omega_1$ and $\omega_2 + \omega_1 - \omega_1 = \omega_2$

And by adding frequency components outside the band (out-of-band distortion)

- *Harmonic distortion* which can be classified in:
  - 2^{nd} *order harmonic distortion* at $2\omega_1$ and $2\omega_2$
  - 3^{rd} *order harmonic distortion* at $3\omega_1$ and $3\omega_2$
- *Intermodulation distortion* which can be classified in:
  - 2^{nd} *order Intermodulation distortion* at $\omega_1 + \omega_2$ and $\omega_1 - \omega_2$
  - 3^{rd} *order Intermodulation distortion* at $2\omega_1 + \omega_2$, $2\omega_1 - \omega_2$ and at $2\omega_2 + \omega_1$, $2\omega_2 - \omega_1$

Figure 2.3 shows the harmonic distortion and intermodulation distortion in the output of the nonlinear PA when considering a two-tone signal. It can be seen how unwanted signals that are far away from the baseband signals are easy to eliminate with simple filters. However, unwanted signals that are very close to the baseband signals can be very difficult to eliminate, requiring complex and expensive filters.

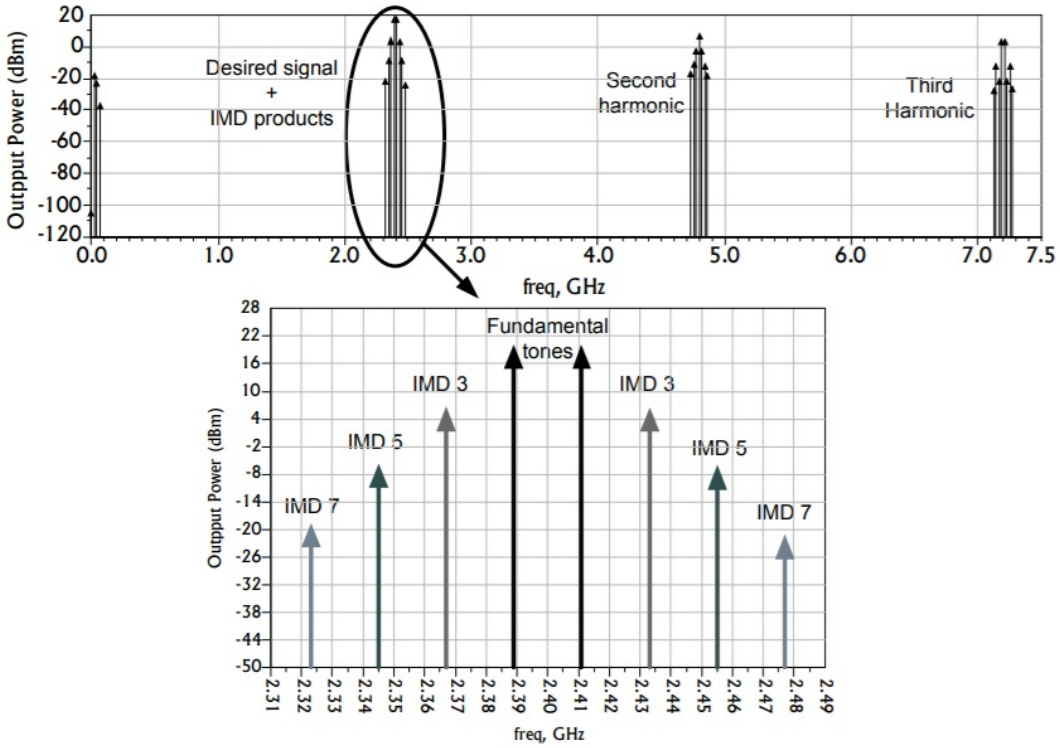

Figure 2.3: Power amplifier two-tone test output spectrum [5]

The two-tone test discussed in [5], represents an approximation to characterize the nonlinear behavior of PA. Usually, the signals that travel through an amplifier are modulated signals characterized by complex frequency spectra. In the case of complex modulated signals, nonlinearities appear in a continuous band of frequencies as a spectral regrowth. As it can be observed from figure 2.4 the output spectrum presents spectral regrowth due to the nonlinear behavior of the PA. As expected, PAs have to meet certain standards for commercial use and the way to quantify some of the characteristics of PAs is through the error vector magnitude (EVM) and the adjacent channel leakage ratio (ACLR). These two metrics allow both in-band and out-of-band quantification of PA distortion and will be explained below.

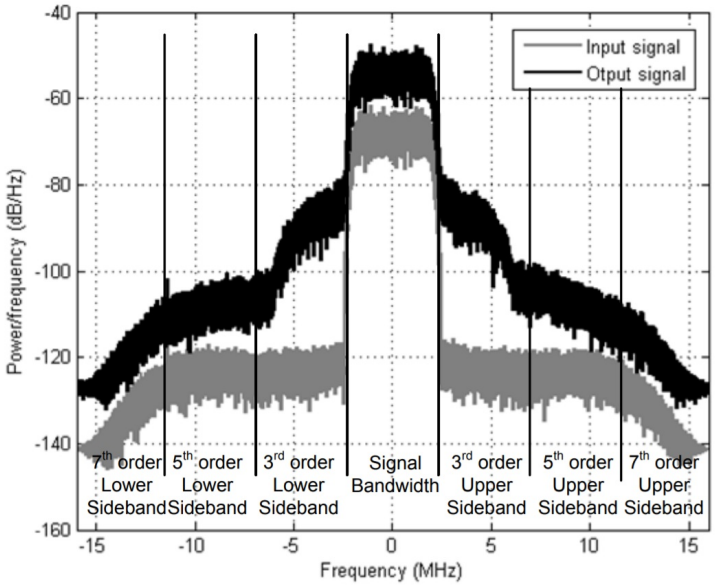

Figure 2.4: Power spectral density of the input and output signals of a nonlinear power amplifier[5]

Metrics for evaluating the linearity in Power Amplifiers.

In order to quantify the in-band distortion, the **Error Vector Magnitude** (EVM) measures the effects of the distortion on the amplitude and phase (or the In-phase and Quadrature components) of the modulated output and it is defined as the square root of the ratio of the mean error vector power to the reference power expressed as a percentage.

$$EVM = \sqrt{\frac{S_{err}}{S_{ref}}}[\%], \tag{2.4}$$

where $S_{err} = \frac{1}{N}\sum_{1}^{N}(\Delta I^2 + \Delta Q^2), S_{ref} = \frac{1}{N}\sum_{1}^{N}(I_{ref}^2 + Q_{ref}^2)$ and N is the number of samples.

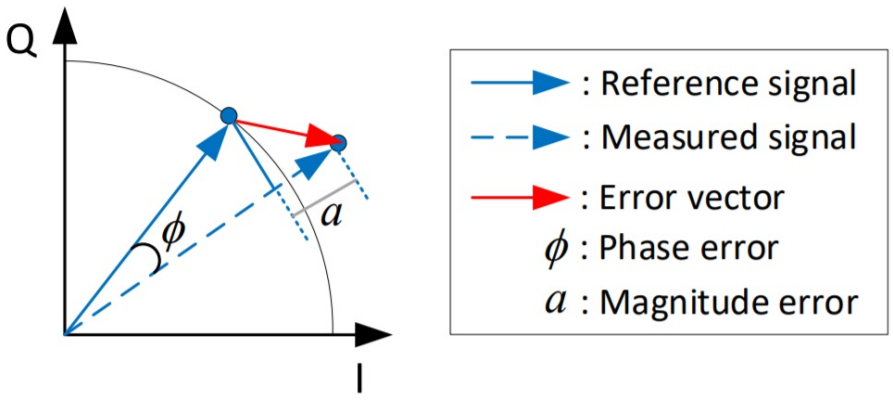

Figure 2.5: Error vector magnitude representation [13]

The **adjacent channel leakage ratio** (ACLR) is a performance metric used for DPD. It

measures the power of the distortion components that are leaked into the adjacent channel in relation to the power of the signal in the main channel (2.5). The ACLR is defined as

$$ACLR = 10\log_{10} \max_{m=1,2} \left[\frac{\int_{(adj)_m} | Y(f)}{\int_{ch} | Y(f) |^2} \right] [dB], \tag{2.5}$$

where $Y(f)$ denotes the power spectrum of the measured output signal $y(n)$. The integration in the numerator is done over the adjacent channel that presents the largest power and the integration in the denominator is done over the transmission channel. The ACLR is generally presented in dB.

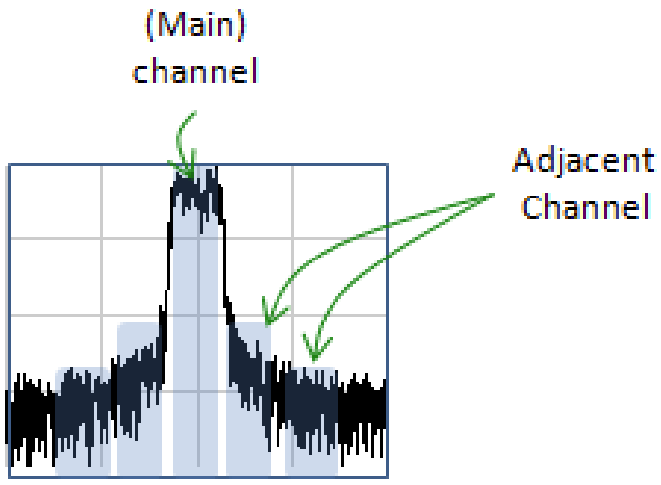

Figure 2.6: Main and adjacent channel representation

2.3. Power Efficiency in Power Amplifiers

Energy efficiency in an PA measures the relationship between signal energy at the output of the amplifier and the energy with which the device has been fed to perform the amplification of the signal. Therefore, we can determine that this efficiency marks the ability of the PA to convert continuous power (power fed into the PA) in radio frequency power (output power). Therefore, efficiency is defined as

$$\eta = \frac{P_{out}}{P_{DC}} [\%], \tag{2.6}$$

The power added efficiency (PAE) considers the presence of the input power P_{in}. It is defined as

$$PAE = \frac{P_{out} - P_{in}}{P_{DC}} [\%], \tag{2.7}$$

If the signal being transmitted consists of a signal with a variable envelope, the efficiency of the amplifier will also be variable, showing greater efficiency when operating near the

compression point. As this efficiency is time dependent, it is possible to calculate the average power as the ratio between the average output power and the average feed power.

$$\eta_{AVG} = \frac{P_{outAVG}}{P_{DCAVG}}[\%], \tag{2.8}$$

As mentioned above, working close to the compression point offers greater efficiency but we have to deal with the nonlinear effects produced by PA. It can be seen that in amplifiers, efficiency and linearity is a trade-off. The more linear the amplifier, the less efficient it is, and the more efficient the PA, the more non-linear effects can be seen. An example of linearity vs efficiency trade-off is drawn in figure 2.7.

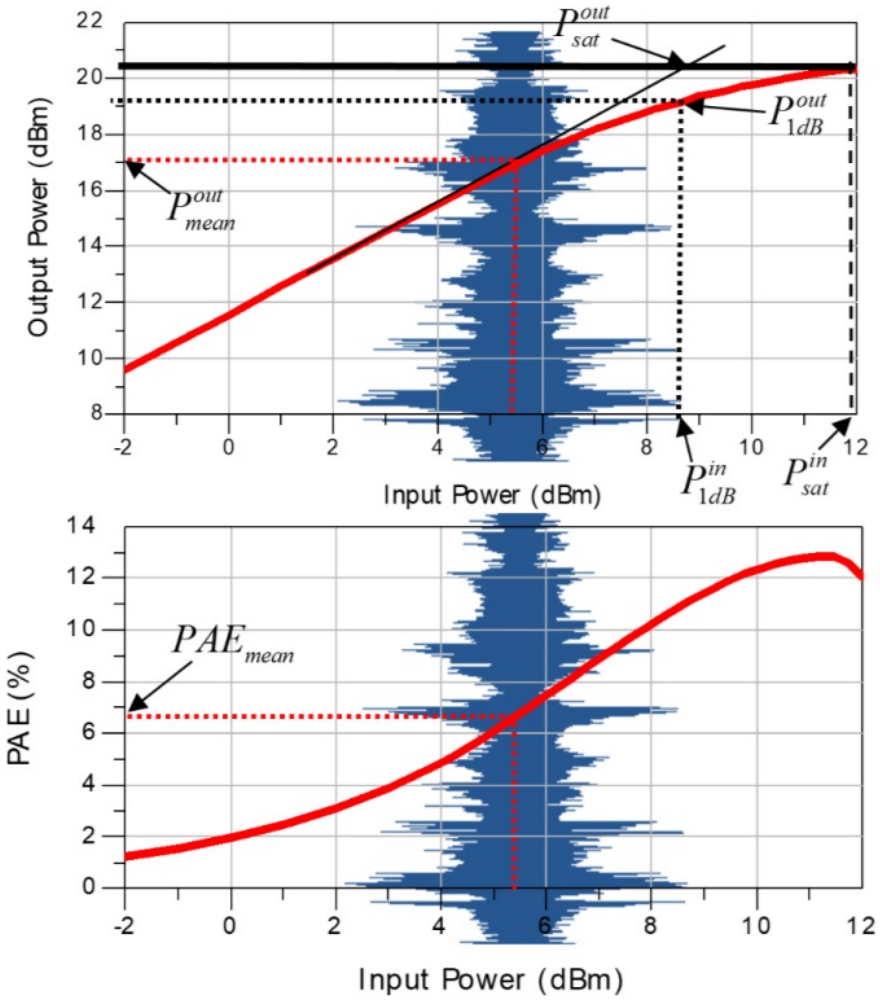

Figure 2.7: Linearity versus efficiency trade-off [13]

2.4. Linearization techniques

A linearization operation is in fact a corrective action where the principle consists on adding an opposite reaction to the non-linearity in order to compensate an to obtain a straight line. The correction can be made before or after the amplifier and the term used are predistor-

tion or post distortion. One can also make the correction at the baseband level sometimes called digital predistorsion or at the RF level called RFPAL. Linearization adds elements or functions in the signal path that compensate the nonlinearity of the elements. These techniques can be roughly classified into three groups: feedback techniques, feedforward techniques and predistortion techniques. In this project we will focus on predistortion techniques especially in digital predistortion (DPD).

Digital predistortion is a popular technique to compensate for PA nonlinearity. It has the advantage of unconditional stability. The bandwidth is limited by the gain and phase flatness of the predistorter. In digital predistortion the inverse of the PAs response function H, is first identified and then used for predistorting the signal before it is fed to the PA in such a way that the output signal of the PA is similar to the desired output, according to given linearization criteria. The output of the digital predistortion algorithm $y(t) = H^{-1}\{x_d(t)\}$ is fed to the PA so that the overall output can be calculated as $x(t) = H\{y(t)\} = H\{H^{-1}[x_d(t)]\}$. If the inverse function H^{-1}, is close to the inverse to the real function of the PA, the output of the PA, $x(t)$, will be close to the desired signal, $x_d(t)$.

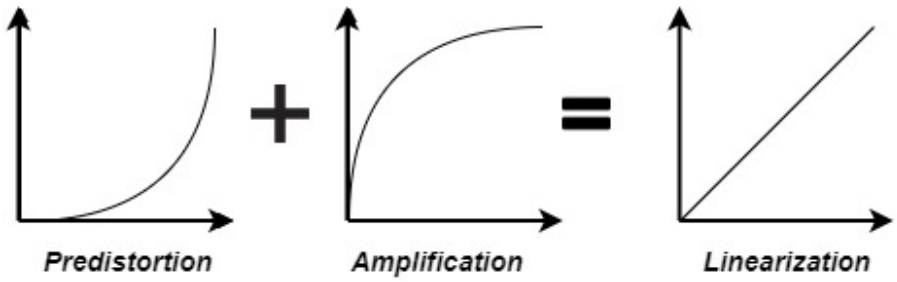

Figure 2.8: Predistortion effect

CHAPTER 3. POWER AMPLIFIER BEHAVIORAL MODELING AND DIGITAL PREDISTORTION LINEARIZATION

3.1. Introduction

Behavioral models are parametric mathematical models describing either the PA nonlinear behavior or its inverse. Selecting a proper behavioral model is key to the correct performance of the DPD linearizer. In the following some of the most popular or commonly used PA behavioral models will be discussed. Behavioral models can be divided into two main groups: behavioural models without memory and those with memory. The models without memory consist of a nonlinear mapping between the input and the output signal. On the other hand, models with memory effect are best suited to characterize the behaviour of the PA. These models take into account that the output of the amplifier not only depends on the input signal at that instant, but also depends on the past samples of the input signal. In this project, the Volterra series , the Memory Polynomial (MP) and the Generalized Memory Polynomial (GMP) behavioral models have been considered and will be explained in more detail in the following.

3.2. Most Common PA Behavioral Models

3.2.1. Volterra series

A Volterra series is a combination of linear convolution and a nonlinear power series; it provides a general way to model a nonlinear system with memory, so that it can be employed to describe the relationship between the input and the output of an amplifier with memory. However, high computational complexity makes general methods of this kind rather impractical in some real applications because the number of parameters to be estimated in the Volterra model increases exponentially with the degree of nonlinearity and with the memory length of the system.An overview of recently developed, simplified Volterra series based, behavioral modeling approaches is presented in [8].

In the discrete time domain, Volterra series can be written as follows

$$\hat{y}(n) = \sum_{p=1}^{P} \sum_{m_1=0}^{M} \cdots \sum_{m_p=0}^{M} w_p(m_1, \cdots, m_p) \prod_{j=1}^{p} x(n - m_j), \tag{3.1}$$

where $x(n)$ and $\hat{y}(n)$ represents the input and the estimated output, respectively, and $w_p(mj, \cdots, mp)$ is called the pth order Volterra kernel. The Volterra series is normally truncated to finite nonlinear order P and finite memory length M. The Volterra series in (3.1) can be directly employed to represent the nonlinear transfer function of a PA. However, in system level analysis and design, most simulators use baseband complex envelope signals to evaluate the system performance since modulation techniques are normally employed in modern wireless communication systems, where only the envelopes carry the

useful information. In order to process these carrier modulation signals, the Volterra series must be converted to a low-pass equivalent format.

$$\begin{aligned}\tilde{y}(n) &= \sum_{m=0}^{M} \tilde{w}_1(m)\, \tilde{x}(n-m) \\ &+ \sum_{m_1=0}^{M}\sum_{m_2=0}^{M}\sum_{m_3=0}^{M} \tilde{w}_3(m_1,m_2,m_3)\, \tilde{x}(n-m_1)\tilde{x}(n-m_2)\tilde{x}^*(n-m_3) \\ &+ \cdots \end{aligned} \tag{3.2}$$

where $\tilde{x}(t)$ and $\tilde{y}(t)$ is the envelope of input and output signal, respectively, $\tilde{w}_p(m_1, m_2, \cdots, m_p)$ is the pth-order complex Volterra kernel and $(\cdot)^*$ represent the conjugate transpose. In equation (3.2), we have removed the redundant items associated with kernel symmetry, and also the even-order kernels, whose effects can be omitted in band-limited modulation systems. However, the main problem of the Volterra series is that the number of coefficients grows significantly when the polynomial degree and memory depth increases. The high-order model may include unnecessary information and lead to ill-conditioned problems.

3.2.2. Memory Polynomial

As it was mentioned in the previous section Volterra series is capable of modelling any non-linear system which has memory effects . Memory polynomial (MP) [5] is a special case of Volterra series which does not have all of the terms in Volterra series. To model the memory effect and nonlinearities of a power amplifier in a digital system, truncation have to be done to original Volterra series due to its high number of coefficients. The MP can be formulated as:

$$\hat{y}(n) = \sum_{p=1}^{P}\sum_{m=0}^{M} w_{p,m} x(n-m)|x(n-m)|^{(p-1)} \tag{3.3}$$

Where x is the input, $\hat{y}$ is the estimated output, P is the maximum power order, M is the maximum memory depth, and w_p is the kernels of the system. As it can be seen from (3.3), the MP is a method that models the system by only considering the input signal and input signal's envelope for the same memory instance. It can also be thought as; MP method only considers the diagonal terms of the Volterra series. By changing P and M one can have control over the memory length and the non-linearity order of the system. This method makes a decent approximation as well as a great reduction of the coefficients.

3.2.3. Generalized Memory Polynomial

In order to improve the performance of the MP method, many different techniques have been proposed. Generalized Memory Polynomial (GMP) [6] is one of these techniques. The difference between GMP and MP is that in GMP there is an additional term used to compare terms in MP. By adding some additional terms for MP, the the modeling system can be improved. The GMP equation can be described as follows

$$\begin{aligned}\hat{y}(n) = &\sum_{p=1}^{P_a}\sum_{m=0}^{M_a} w_{p,m}x(n-m)|x(n-m)|^{(p-1)} \\ &+ \sum_{k=1}^{P_b}\sum_{m=0}^{M_b}\sum_{l=0}^{L_b} w_{p,m,l}x(n-l)|x(n-m-l)|^p \\ &+ \sum_{k=1}^{P_c}\sum_{m=0}^{M_c}\sum_{l=0}^{L_c} w_{p,m,l}x(n-l)|x(n-m+l)|^p \end{aligned} \tag{3.4}$$

Where P_a and M_a are the non-linearity order and memory depth of the diagonal terms.P_b, M_b, P_c, M_c are the nonlinearity order and memory depth of the cross terms and L_b, L_c are the values that controls the leading and lagging terms in the method. Second line of equation (3.4) is a different version of first line by lagging the envelope compare to the signal itself, and the third line is same except instead of lagging the envelope is shifted forward in time (leading). This two new lines with respect to MP, make remarkable change on the modelling performance.

3.3. Behavioral Model Parameters Estimation

The main objective of this algorithm is to find the $\hat{w}$ coefficients of the model that best fits in terms of quadratic error given the matrix of regressors and the output signal. The number of samples in the input and output is much higher than the number of regressors and this implies that it is not possible to find a solution that meets all the conditions and therefore it is an incompatible system.

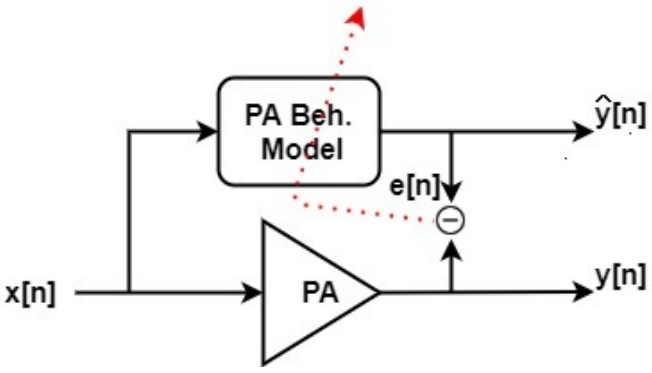

Figure 3.1: Block diagram of PA behavioral modeling

By means of the LS technique we can estimate the output signal to get as close as possible to the desired solution.

$$\hat{y} = X \cdot \hat{w}, \tag{3.5}$$

where $\hat{y}[n]$ is the estimated output with a length equal to N, $\hat{w}(\hat{w_1}, \cdots, \hat{w_i}, \cdots, \hat{w_M})^T$ where M indicate the number of basis functions and X is a $N \times M$ matrix and can be defined as:

$$X = (\varphi_x[0], \varphi_x[1], \cdots, \varphi_x[n], \cdots, \varphi_x[N-1])^T, \tag{3.6}$$

where $\varphi_x[n] = (\phi_1^x[n], \cdots, \phi_i^x[n], \cdots, \phi_M^x[n])$ is the $M \times 1$ vector of basis functions $\phi_i^x[n]$ (with $i = 1, \cdots, M$) at time n.

The estimated error will be the difference between the output signal in PA and the estimated output signal.

$$\hat{e} = y - \hat{y} = y - X \cdot \hat{w} \tag{3.7}$$

The aim of this method is to find the values of w that minimize this error taking the ℓ_2-norm squared of the error, and it is defined as follows

$$\min_{\hat{w}} \|\hat{e}\|_2^2 = \min_{\hat{w}} \|y - X \cdot \hat{w}\| \tag{3.8}$$

Therefore, the cost function to be minimized is as follows:

$$J(\hat{w}) = \|\hat{e}\|_2^2 \tag{3.9}$$

To minimize this function, we will calculate its derivative and set it equal to zero. The result obtained will be the coefficients that minimize the error and can be seen in the following equation.

$$\hat{w} = (X^H X)^{-1} X^H y \tag{3.10}$$

3.4. Adaptive predistortion

Adaptive predistortion [3] corresponds to the implementation of closed-loop DPD. In this method, there is a feedback loop that allows the modification of the DPD coefficients, therefore, DPD will adapt to the changes that the PA introduces. This time-dependent response can be due to: thermal changes, equipment aging or the implementation of automatic gain control. The implementation of the adaptive method leads to an increase in equipment cost and energy consumption. In the forward path, the input-output relationship at the DPD block can be described as

$$x[n] = u[n] - d[n] \tag{3.11}$$

where $x[n]$ is the signal at the output of the DPD block, $u[n]$ is the input signal and $d[n]$ is the distortion signal introduced by the PA and can be modeled with some of the behavioral models (Volterra, GMP or MP). Therefore, $d[n]$ can be written as follows,

$$d[n] = \varphi_u^T[n] w[n], \tag{3.12}$$

where $w = (w_1, \cdots, w_i, \cdots, w_M)^T$ is a vector of coefficients at time n with dimensions $M \times 1$, with M being the order of the behavioral model, $\varphi_u^T[n] = (\phi_1^u[n], \cdots, \phi_i^u[n], \cdots, \phi_M^u[n])$ is the vector of basis functions $\phi_i^u[n]$ (with $i = 1, \cdots, M$) at time n.

Equation 3.11 can be written in matrix notation as follows

$$x = u - U \cdot w, \tag{3.13}$$

where the U is a $N \times M$ matrix described as follows

$$U = (\varphi_u[0], \varphi_u[1], \cdots, \varphi_u[n], \cdots, \varphi_u[N-1])^T, \tag{3.14}$$

In adaptive predistortion, the extraction of the DPD coefficients can be carried out following a direct learning or an indirect learning approach.

In Indirect learning, the PA behavioral model is post-inversed by a postdistorter and the coefficients of the postdistorter are applied to the predistorter. As mentioned in [4], indirect learning assume that the coefficients of the postdistortion are equal to the coefficients of the predistortion. Consequently, the input of the postdistortion can be noisy, making the postdistorter coefficients estimation converge to biased values. The block diagram of a closed-loop adaptive DPD architecture following an indirect learning approach is shown in figure 3.2. In the direct learning method, the DPD coefficients are continually adjusted by comparing the PA output y to the input signal u. This approach is also named closed-loop since the predistorter is inside the feedback loop.

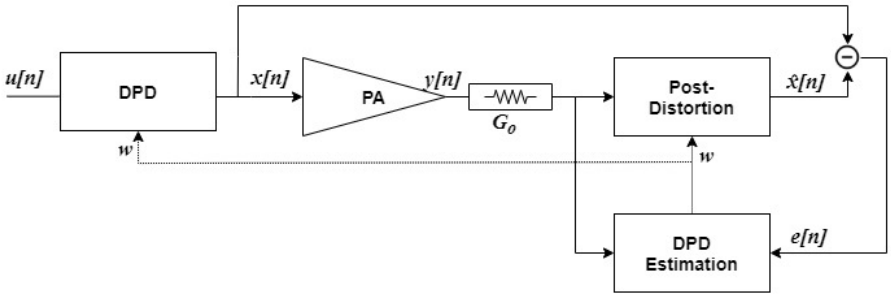

Figure 3.2: Indirect learning method for DPD

As explained in [4], the main advantage in direct learning is that we gain robustness against noisy PA output observations and avoid the offset of the coefficient vector from its optimal value. The block diagram of a closed-loop adaptive DPD architecture following an direct learning approach is shown in figure 3.3.

Finally, the DPD coefficients can be updated iteratively as follows

$$w_{i+1} = w_i + \mu \Delta w_i \tag{3.15}$$

with $\mu(0 \geq \mu \leq 1)$ being a weighting factor and Δw can be estimated by the following LS solution

$$\Delta \hat{w} = (U^H U)^{-1} U^H e, \tag{3.16}$$

where e is the $N \times 1$ vector of the identification error defined as

$$e = \frac{y}{G_o} - u \tag{3.17}$$

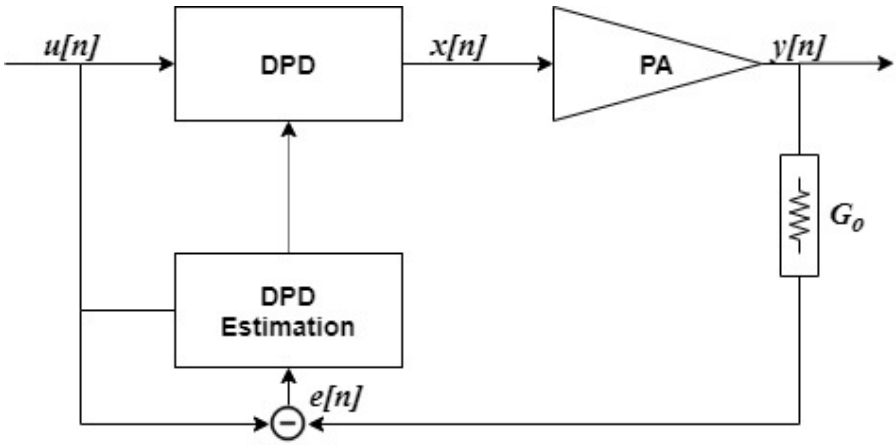

Figure 3.3: Direct learning method for DPD

CHAPTER 4. DIMENSIONALITY REDUCTION TECHNIQUES

4.1. Feature selection techniques

In the field of DPD linearization, dimensionality reduction techniques are used with a double objective: ensure a proper, well-conditioned parameter identification and reducing the number of estimated parameters (thus reducing the computational complexity and memory requirements of a hardware implementation).

As described in [3], some of the proposed solutions for dimensionality reduction of DPD linearizers are based on feature selection techniques. The objective of these techniques is to enforce the sparsity constraint on the vector of coefficients by minimizing the number of regressors (i.e., ℓ_0-norm) subject to a constraint on the ℓ_2-norm squared of the identification error. For example, particularizing for the identification of the PA behavioral model coefficients described in Section 3.3., the optimization problem can be described as

$$\begin{aligned} &\min_{w} \|w\|_0 \\ &\text{subject to} \quad \|y - Xw\|_2^2 \leq \varepsilon \end{aligned} \tag{4.1}$$

Unfortunately, this is a non-deterministic polynomial-time hard (NP-hard) combinatorial search problem. Therefore, in the field of DPD linearization, several sub-optimal approaches based on greedy and global optimization (or heuristic search) algorithms have been proposed targeting both robust identification and model order reduction. In the following sections some selected greedy and heuristic search algorithms will be presented and discussed for a later comparison in Chapter 5 of their dimensonality reduction capabilities, in terms of accuracy and computational time.

4.2. Greedy algorithms

Greedy algorithms allow the reduction of regressors (or basis functions) from a single model by analysing the importance of each regressor and thus, reducing the number of estimated coefficients. When a behavioral model such as GMP defined in (3.4) is considered, we do not have a priori knowledge which is the best configuration of the polynomial order, memory depth or memory of the cross-products. This leads to the use of regressors that increase the complexity of the model without improving its accuracy and eventually will derive in an ill-conditioned estimation. Some of the most relevant greedy algorithms proposed in literature oriented at addressing the optimal configuration of PA and DPD behavioral models will be explained in the following. The specific algorithms that will be addressed are: least absolute shrinkage and selection operator (LASSO), Ridge regression, orthogonal matching pursuit (OMP), doubly orthogonal matching pursuit (DOMP), Subspace Pursuit and Random Forrest.

In order to evaluate the greedy algorithms, the original GMP configuration, taking into account the notation in (3.4), is:

- $p_a = 9;\ p_b = 3;\ p_c = 3$
- $m_a = [0:1:10];\ m_b = [-1:1:3];\ m_c = [-1:1:3]$
- $l_b = [1:1:2];\ l_c = [1:1:2]$

With this initial configuration the GMP behavioral model presents a number of 139 parameters.

4.2.1. Regularization techniques applied to dimensionality reduction

One of the challenges of behavioral modeling is choosing the necessary features to characterize the system, thus avoiding underfitting or overfitting problems. An underfitting model is one that does not have the essential coefficients required to properly characterize the system while, on the other hand, an overfitting model is one that has more coefficients than the necessary. Least absolute shrinkage and selection operator (LASSO) proposed in [40] and Ridge regression proposed in [39] are adjustment techniques where a regularization term (or constraint) is added to the cost function to avoid overfitting.

$$J(w) = \|y - Xw\|_2^2 + \lambda R(w) \tag{4.2}$$

In general, the main idea of the regularization techniques is to add a regularization term $R(w)$ to the cost function. Thus a cost function for Ridge and LASSO is presented as a restricted version of the ordinary least squares (OLS). In the case of Ridge, the constraint will be based on the ℓ_2-norm of the vector of coefficients, while in the case of LASSO, it will be based on the ℓ_1-norm of the vector of coefficients.

4.2.1.1. Ridge regression or Tikhonov ℓ_2 regularization

In ℓ_2 regularization, the goal is to minimize the residual sum of squares subject to a constraint on the sum of squares of the coefficients.

$$\begin{aligned} &\min_{w} \|y - X \cdot w\|^2 \\ &\text{subject to } \sum_{i=1}^{M} |w_i|^2 \leq r_2. \end{aligned} \tag{4.4}$$

This restriction forces the coefficients to be in a sphere of radius r_2. The solution of the regression of Ridge is those coefficients that satisfied the restriction set, as can be seen in figure 4.1 . Another way to represent the cost function is the following:

$$\min_{w} \|y - X \cdot w\|^2 + \lambda \|w\|_2^2$$

Once written in a compact form, the coefficients can be extracted using the following equation:

$$w_{Ridge} = (X^H X + \lambda I)^{-1} X^H y \tag{4.5}$$

where I is the identity matrix. w_{ridge} depends on the shrinkage parameter λ, as it grows, there will be fewer coefficients that meet the condition, whileλ tends to zero, an expression similar to OLS is obtained. Based on this concept, the w_{ridge} will be calculated for different λ, thus obtaining an NMSE vs number of coefficients curve.

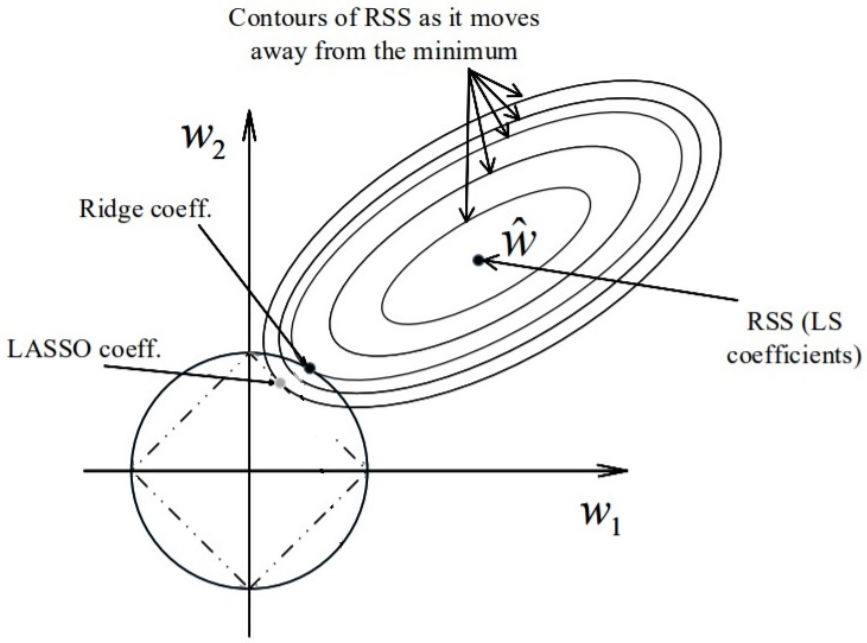

Figure 4.1: Ridge and LASSO 2D graphical representation [3]

Algoritmo 1: Ridge Algorithm for Dimensionality Reduction

INPUTS: X, y, K
OUTPUT: W
initialization;
for $i \leftarrow 1$ **to** $length(K)$ **do**
    Generate matrix identity I
    Calculate w_{ridge} *using (4.5)*
    Select coefficients that have an absolute value greater than a $w_{threshold}$
    Store the select coefficients in W_i
end

The following algorithm has as inputs: the normalized regressor matrix X, the y vector of PA output data and a vector K that contains all λ values. The algorithm provides as output a matrix W where each column contains the index of the selected coefficients taking into account the λ vector. Note that the length of K will be the same than the columns of W. First, the algorithm finds a value from λ that reduces the coefficients obtained to below $w_{threshold}$ in modulus. With this, we assure that the matrix is well regularized. Then, indexes of coefficients that meet the condition for that particular λ is selected and stored in the output matrix W as a column. The above steps are repeated for all input lambdas until final K value is reached.

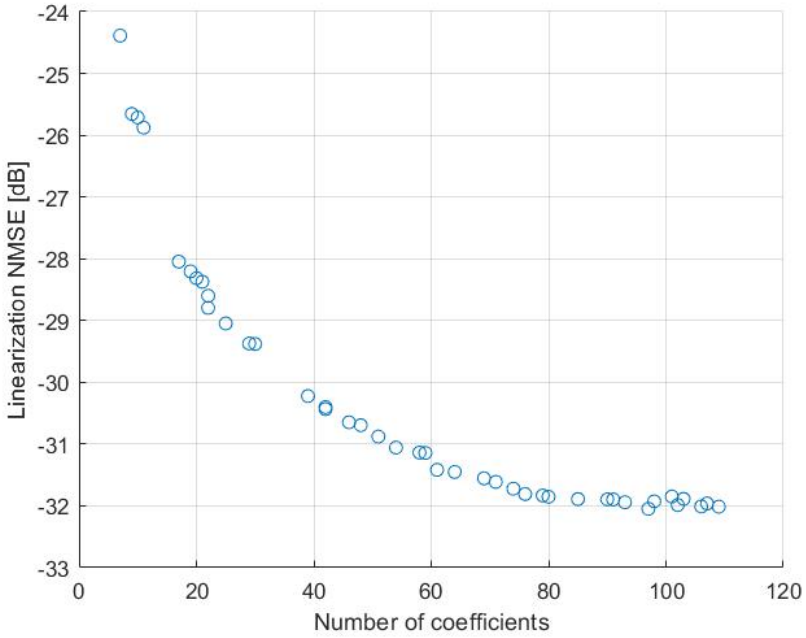

Figure 4.2: NMSE vs number of coefficients for Ridge

Once completed, each column of the output matrix W will represent a configuration for which its NMSE can be calculated. Figure 4.2 show the NMSE for each of the K values.

In figure 4.3 it can be seen that the lambda value affects the NMSE and the number of coefficients. Note that the horizontal axis corresponds to the $log_{10}(\lambda)$. For very small values of λ, the model is not well regularized, since the restriction set is insufficient; therefore, the NMSE is similar to that of the original model, and the number of coefficients also. As λ increases, there are fewer regressors that comply with the set restriction, and therefore the number of coefficients decreases while NMSE increases.

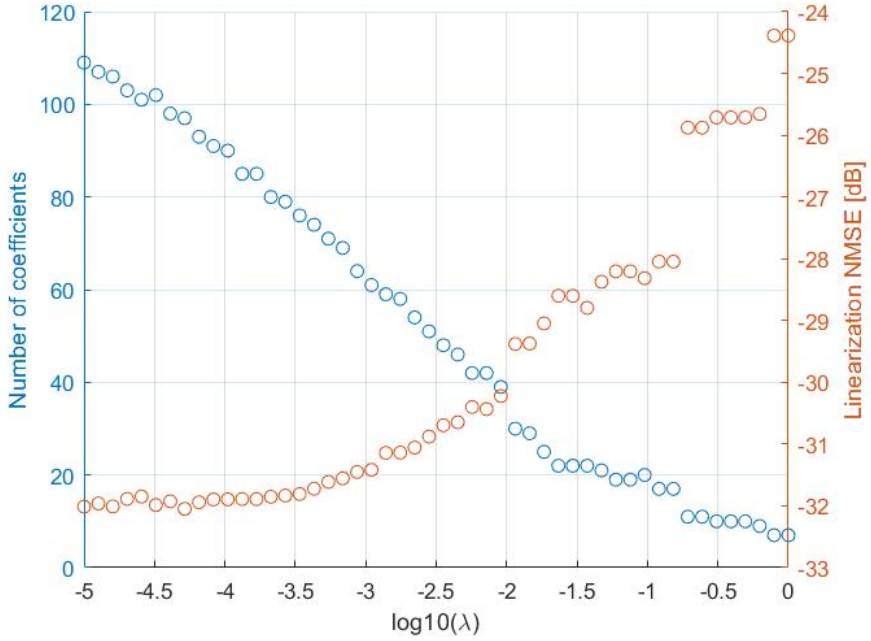

Figure 4.3: Variation of NMSE and the number of coefficients with respect to λ

4.2.1.2. LASSO or ℓ_1 regularization

Similarly to Ridge regression, LASSO can be used for both regularization and to generate a sparse model. Whereas the constraint of the Ridge regression is the sum of square of the coefficients, the LASSO constraint consists of the sum of the absolute value of the coefficients. Thus, the solution of LASSO regression satisfies to the following ℓ_1 optimization problem,

$$\begin{aligned} &\min_{w} \|y - X \cdot w\|^2 \\ &\text{subject to } \sum_{i=1}^{M} |w_i| \leq r_1. \end{aligned} \tag{4.6}$$

The constrained cost function can also be written as following:

$$\min_{w} \|y - X \cdot w\|^2 + \lambda \|w\|_1$$

As depicted in Figure 4.1, this constraint forces the coefficients to stay within the diamond shape.

Since the two algorithms works very similar, the same technique that has been applied in

the ridge will be used.

Algoritmo 2: LASSO Algorithm for Dimensionality Reduction

INPUTS: X, y, K
OUTPUT: W
Generate X_{LASSO} and y_{LASSO} with (4.7)
Calculate W_{LASSO} with all K values
for $i \leftarrow 1$ **to** $length(K)$ **do**
    Set i column of W_{LASSO} as w_i
    Store nonzero coefficients of w_i in $_{nonzero}$
    Store in W the $w_{nonzero}$ vector
end

LASSO has as input data: the matrix of regressors X, the output signal y and the vector K and as output data, a matrix W where each column indicates the coefficients that comply with the ℓ_1-norm condition. Note that the output matrix W will have the same number of columns than the K vector length.

When computing LS, the fact that the samples are complex does not add an extra to the problem, the solution is calculated the same as for the real numbers. The same goes for Ridge as its solution does not differ much from the solution of the LS in terms of calculation. However, for LASSO the thing changes, it cannot be applied directly to the complex sample vectors. The input matrix will be adapted to solve this problem, as can be seen in the equation (4.7).

$$X_{LASSO} = \begin{bmatrix} Re[X] & -Img[X] \\ Img[X] & Re[X] \end{bmatrix}$$
$$y_{LASSO} = \begin{bmatrix} Re[y] \\ Img[y] \end{bmatrix} \tag{4.7}$$

Once the problem has been solved, the algorithm can be applied. The objective is to calculate the vector of coefficients for each of the K values. Once the calculations to obtain w_{LASSO} have been carried out, it has to be readapted to obtain the same original length with real part and imaginary part

$$w = w_{LASSO}(1:L) + i \cdot w_{LASSO}(L:end), \tag{4.8}$$

where L is the number of columns in the X matrix.

Figure 4.5 shows the variation of the NMSE and the number of coefficients as a function of $log_{10}(\lambda)$. For very small values of λ, the model is barely regularized, since the restriction set is insufficient; therefore, the NMSE is similar to that of the original model, and the number of coefficients also. As λ increases, there are fewer regressors that comply with the set restriction, and therefore the NMSE gets worse, and the number of coefficients decreases similar to Ridge.

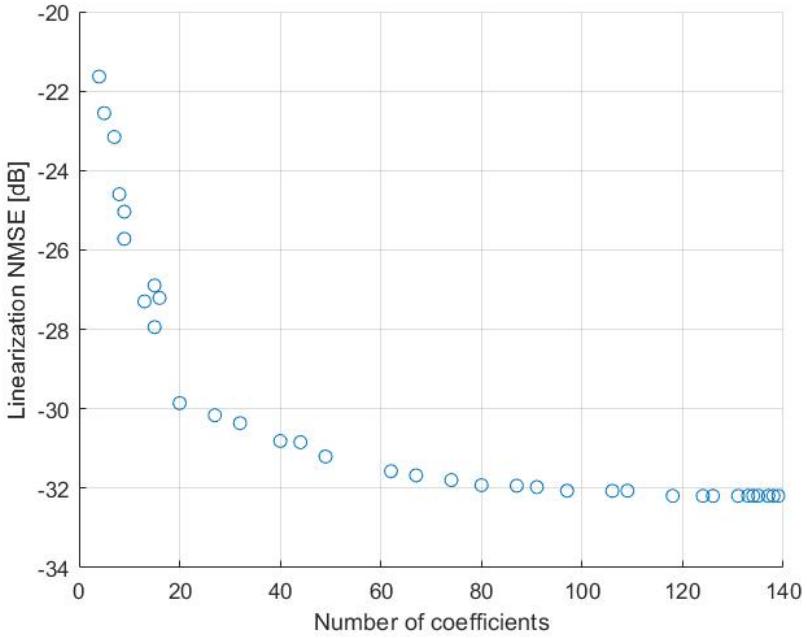

Figure 4.4: NMSE vs number of coefficients for LASSO

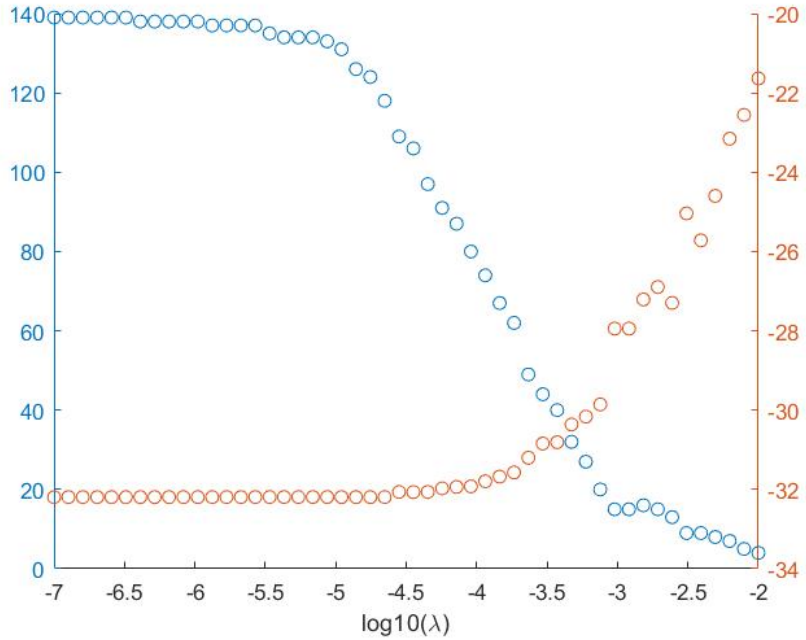

Figure 4.5: Variation of NMSE and the number of coefficients with respect to λ

4.2.2. Orthogonal Matching Pursuit

Orthogonal Matching Pursuit (OMP) is a greedy and iterative algorithm used in [14] for PA behavioral modeling purposes. This algorithm find the solution of a problem by adding element by element in an iterative step-by-step manner optimally at each stage. Once the

PA behaviour is modelled, not all the kernels (i.e., regressors, basis functions) obtained contribute to the improvement of the modelling, indeed, the suppression of those elements will allow us to enhance the robustness and efficiency of the modelling. Given a matrix of regressors, OMP creates a new support set adding step by step regressor that has more resemblance with the residual, measured by their cross-correlation.

Algoritmo 3: OMP algorithm [14]

INPUTS: X, y
OUTPUT: S, $\hat{w}$
$Initialization: r^{(0)} \leftarrow y$, $S^{(0)} \leftarrow \varnothing$
for $t \leftarrow 1$ **to** m_{max} **do**
  $g_{\{i\}}^{(t)} \leftarrow \frac{X_{\{i\}}^H}{\|X_{\{i\}}\|_2} r^{(t-1)}$
  $s^{(t)} \leftarrow supp(H_1|g^{(t)})$
  $S^{(t)} \leftarrow S^{(t-1)} \cup s^{(t)}$
  $\hat{w}_{S^{(t)}} \leftarrow X_{S^{(t)}}^{\dagger} y$
  $\hat{y}^{(t)} \leftarrow X_{S^{(t)}} \hat{w}$
  $r^{(t)} \leftarrow y - \hat{y}^{(t)}$
end

The input variables for this algorithm are the matrix X of regressors and the output data y already defined in (3.14). The output variables are the vector S, which will contain all the regressors that the OMP algorithm will select in order from major to minor importance and the last output variable is the coefficient vector $\hat{w}$ also ordered from major to minor importance. Note that the indexes stored in S will coincide with the coefficient vector w.

Initializing the residue r with a value equal to the output y. The loop starts with the variable g where the correlation between the matrix of regressors and the residue will be calculated. This variable is a vector of length M which contains in each position a value that indicates how correlated this regressor is with the residue.

$$g_{\{i\}}^{(t)} \leftarrow \frac{X_{\{i\}}^H}{\|X_{\{i\}}\|_2} r^{(t-1)} \tag{4.9}$$

The operation $supp(H_1|g^{(t)})$ consists on obtaining the index of the maximum value of g, thus obtaining the index of the most correlated element. OMP algorithm add the index $s^{(t)}$ to our support set $S^{(t)}$. Then, an estimation of the coefficients vector with just the columns that belong to the support set is performed with the following equation:

$$\hat{w}_{S^{(t)}} \leftarrow X_{S^{(t)}}^{\dagger} y, \tag{4.10}$$

The Moore–Penrose pseudoinverse $X^{\dagger}$ computes the LS solution and can be written as the following equation:

$$X^{\dagger} = (X^H X)^{-1} X^H, \tag{4.11}$$

where H is the Hermitian transpose operator. The output $\hat{y}$ is estimated by means of the coefficients that have already been added to the set and finally the residue is updated by

subtracting between the estimated output and the real output.

$$r^{(t)} = y - \hat{y}^{(t)} \tag{4.12}$$

This last step is very important because it eliminates the possibility of a regressor being selected again. This loop is repeated until the residue obtained is low enough to consider it zero or until all available regressors are selected.

Once the algorithm is finished, a representation between the NMSE and the number of coefficients can be seen graphically in figure 4.6. The overffiting effect can be clearly seen, after 80 coefficients the NMSE no longer improves. One way to solve this problem is applying the Bayesian information criterion (BIC) presented in [21] to prune some coefficients and determine the optimum number of coefficients m_{opt}. In statistics, the BIC is a criterion for model selection among a finite set of models; the model with the lowest BIC is preferred. The BIC is defined as the sum of a term that depends on the error and a penalty that is related to the number of model coefficients m

$$BIC = 2N \ln \hat{\sigma}_e^2 + 2m \ln 2N \tag{4.13}$$

where $\hat{\sigma}_e^2$ is the error variance and N is the number of samples used for the model identification. The BIC in (4.6) acts as a trade-off between the error and the number of components. In an iterative sorting algorithm, the modeling error is commonly known as the residual error r; therefore:

$$\hat{\sigma}_e^2 = Var[e] = Var[y - \hat{y}], \tag{4.14}$$

where $Var[\cdot]$ represents the variance of its argument.

Once the BIC of all the components of the model has been obtained, the minimum BIC obtained corresponds to m_{opt}

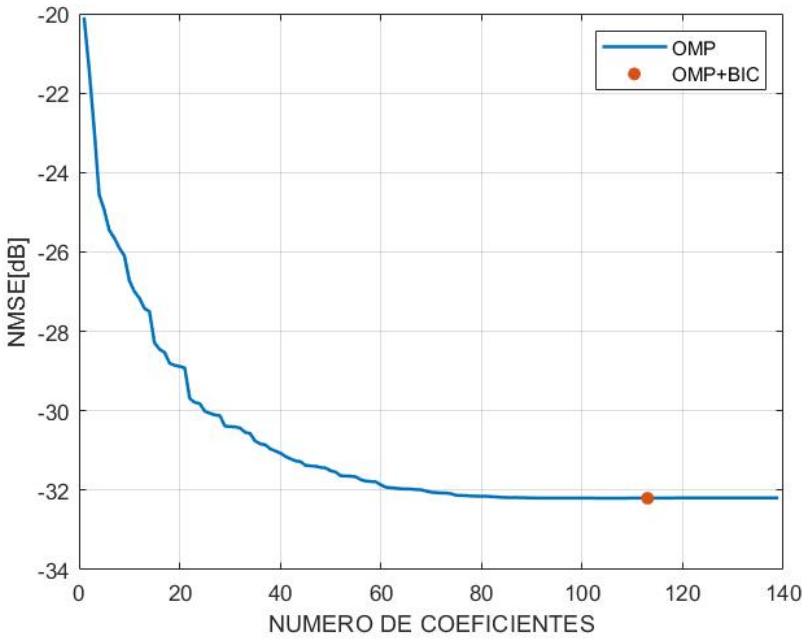

Figure 4.6: Linearization NMSE for OMP with BIC

4.2.3. Doubly Orthogonal Matching Pursuit

OMP is a greedy algorithm that makes a hard decision based on a local optimal criterion whereby the estimated output of the model is always orthogonal to the residual, orthogonalization from which OMP takes its name. Since OMP selects one component of the model at each iteration, a pseudoinverse of the data matrix with a number of columns that is equal to the iteration value has to be performed. The estimation of Volterra coefficients is intricate since the basis functions of the Volterra series are highly correlated. This correlation leads to a large condition number in the model matrices, implying that the equations system is ill-conditioned, affecting the least squares (LS) solution. In this section, a variation of OMP known as Doubly orthogonal matching pursuit (DOMP) presented in [20] to enhance the selection of coefficients in a sparse parameter identification of the model, is described. The main difference between them is the addition of the Gram–Schmidt process at one step of the OMP algorithm, decorrelating the selected regressors and those still to be selected.

The goal of this process is given a subspace W, we would like to find an orthogonal or orthonormal basis for W. The Gram-Schmidt process allows to start with any basis for W and construct a new basis that is orthogonal. Starting from a basis $\{x_1, x_2, ..., x_p\}$ for W and the idea is to construct new vectors $v_1, v_2, ..., v_p$ as follows:

$$\begin{aligned} v_1 &= x_1 \\ v_2 &= x_2 - \tfrac{x_2 \cdot v_1}{v_1 \cdot v_1} v_1 \\ v_3 &= x_3 - \tfrac{x_3 \cdot v_1}{v_1 \cdot v_1} v_1 - \tfrac{x_3 \cdot v_2}{v_2 \cdot v_2} v_2 \\ &\vdots \\ v_p &= x_p - \tfrac{x_p \cdot v_1}{v_1 \cdot v_1} v_1 - \tfrac{x_p \cdot v_2}{v_2 \cdot v_2} v_2 - \cdots - \underbrace{\frac{x_p \cdot v_{p-1}}{v_{p-1} \cdot v_{p-1}} v_{p-1}}_{projection\ of\ x_p\ onto\ v_{p-1}} \end{aligned} \tag{4.15}$$

At each iteration, the projection of x_i onto v_{i-1} is subtracted to obtain , that is going to guarantee two things: that v_i is orthogonal to v_{i-1} and also it is going to guarantee that the span of v_i and v_{i-1} is the same. So the process is taking away the portions of x_i that point on the direction of v_{i-1} therefore the remaining component of x_i is orthogonal to v_{i-1}, v_{i-2} and the ones before that in case they are calculated. When the process has been completed, the v vectors will all be orthogonal to each other while maintaining the span. If an orthonormal basis is desired, we first use Gram-Schmidt to find an orthogonal basis then we normalize each vector to obtain an orthonormal basis. Adding this process to the OMP, the following algorithm is obtained where the pseudocode of the DOMP presented by [20] can be seen.

Algoritmo 4: DOMP algorithm [20]

INPUTS: X, y

OUTPUT: S, $\hat{w}$

$Initialization: Z^{(0)} \leftarrow X \; r^{(0)} \leftarrow y \; S^{(0)} \leftarrow \varnothing$

for $t \leftarrow 1$ **to** m_{max} **do**

$g_{\{i\}}^{(t)} \leftarrow \frac{Z_{\{i\}}^{H}}{\|Z_{\{i\}}\|_2} r^{(t-1)}$

$s^{(t)} \leftarrow supp(H_1|g^{(t)})|$

$S^{(t)} \leftarrow S^{(t-1)} \cup s^{(t)}$

$p^{(t)} \leftarrow Z_{\{i^{(t)}\}}^{(t-1)H} Z^{(t-1)}$

$Z^{(t)} \leftarrow Z^{(t-1)} - p^{(t)} \otimes Z_{\{i^{(t)}\}}^{(t-1)}$

$\hat{w}_{S^{(t)}} \leftarrow X_{S^{(t)}}^{\dagger} y$

$\hat{y}^{(t)} \leftarrow X_{S^{(t)}} \hat{w}$

$r^{(t)} \leftarrow y - \hat{y}^{(t)}$

end

The algorithm returns a support set, S, whose elements are sorted in decreasing impact over the output and the vector of coefficients w. The initial state of the support set is empty, $S^{(0)} = \varnothing$, since no components have been added to it yet. Prior to the algorithm iterations, the matrix $Z^{(0)} = X$ is defined. This matrix will be used to keep the information of the orthogonalized regressors, and after the algorithm execution, it will be equal to the result of applying the Gram–Schmidt procedure to the regressors matrix X in the order of the final support set $S^{(end)}$. The residual is set to $r^{(0)} = y$, since it keeps track of the remainder left to be captured by the selected regressors of the model, and initially, there are no components in the support set.

The loop start with the normalization of basis components, dividing each regressor of Z by its ℓ_2-norm followed by the selection of the component with the highest normalized scalar resolution in the direction of the residual.

$$g_{\{i\}}^{(t)} = \frac{Z_{\{i\}}^{H}}{\|Z_{\{i\}}\|_2} r^{(t-1)} \tag{4.16}$$

The algorithm performs the Gram-Schmidt orthogonalization by first obtaining the vector projections p of the selected regressors onto each one of the components of the basis. As explained above, projection is subtracted from each regressor;

$$\begin{aligned} p^{(t)} &= Z_{\{i^{(t)}\}}^{(t-1)H} Z^{(t-1)} \\ Z^{(t)} &= Z^{(t-1)} - p^{(t)} \otimes Z_{\{i^{(t)}\}}^{(t-1)} \end{aligned} \tag{4.17}$$

hence, the selected component is orthogonal to the remaining of the basis set. Therefore, a double orthogonalization will be obtained as the name indicates. Finally kernel vector is computed through LS and the residual updated. Figure 4.7 shows NMSE vs number of coefficients and the optimum number of coefficients with BIC.

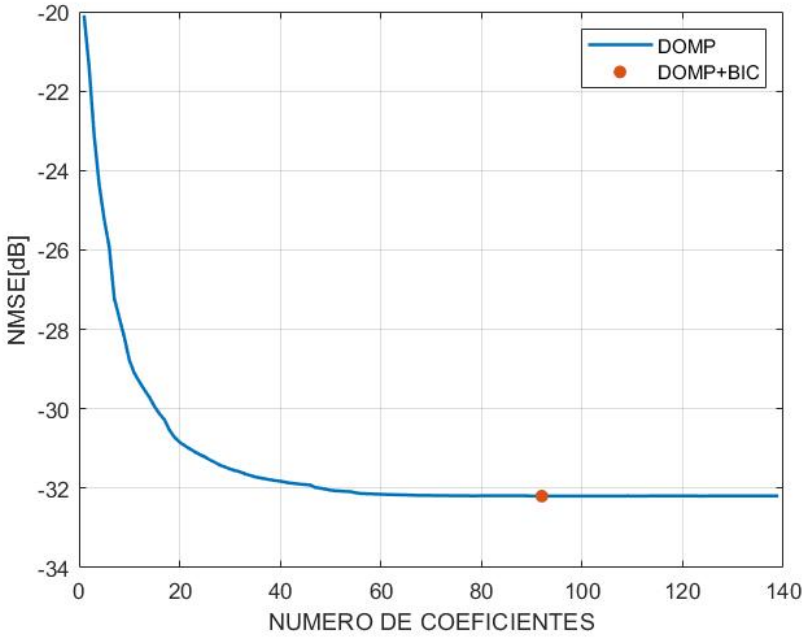

Figure 4.7: Linearization NMSE for DOMP with BIC

4.2.4. Subspace Pursuit

DOMP and OMP are greedy pursuit algorithms that iteratively add new components to the support set S, which is defined as the set of components in the model that are not zero, providing element selection, i.e., a sorted list of regressors for the model. Although these algorithms are easy-to-implement and fast algorithms they do not have recovery guarantees. Subspace Pursuit is a thresholding technique which provide element selection and pruning at once presented in [14], and theoretical performance guarantees that belong to the family of thresholding algorithms.

Algoritmo 5: SP algorithm [14]

INPUTS: X, y, k
OUTPUT: S, $\hat{w}$
$Initialization: r^{(0)} \leftarrow y$, $S^{(0)} \leftarrow \varnothing$
for $t \leftarrow 1$ **to** $stopping\ criterion$ **do**
$g^{(t)}_{\{i\}} \leftarrow \frac{X^H_{\{i\}}}{\|X_{\{i\}}\|_2} r^{(t-1)}$
$s^{(t)} \leftarrow supp(H_k|g^{(t)})$
$S^{(t-0.5)} \leftarrow S^{(t-1)} \cup s^{(t)}$
$\hat{w}_{S^{(t-0.5)}} \leftarrow X^{\dagger}_{S^{(t-0.5)}} y,\ \hat{w}_{\overline{S^{(t-0.5)}}} = 0$
$S^{(t)} \leftarrow supp(\hat{w}_{S^{(t-0.5)}})$
$\hat{y}^{(t)} \leftarrow X_{S^{(t)}} \hat{w}$
$r^{(t)} \leftarrow y - \hat{y}^{(t)}$
end

The algorithm start by setting the residue and the support set:

$$r^{(0)} \leftarrow y,\ S^{(0)} \leftarrow \varnothing \tag{4.18}$$

Unlike the OMP and DOMP algorithms, the SP algorithm does not select a single element but the k elements most correlated to the residue to be added to the intermediate support set.

$$\begin{aligned} g_{\{i\}}^{(t)} &= \frac{X_{\{i\}}^{H}}{\|X_{\{i\}}\|_2} r^{(t-1)} \\ s^{(t)} &= supp(H_k|g^{(t)}) \end{aligned} \tag{4.19}$$

The estimation of the kernel intermediate kernel vector is then performed and those coefficients belonging to the complement of the support set are set to zero.

$$\hat{w}_{S^{(t-0.5)}} = X^{\dagger}_{S^{(t-0.5)}} y, \tag{4.20}$$

The k highest values of $\hat{w}_{S^{(t-0.5)}}$ are added to the support set $S^(t)$ Finally a second pseudoinverse is performed to update $\hat{w}$.

While both OMP and DOMP add one component to the support set per iteration, SP run iteratively for a desired sparsity level, providing a k-sparse solution until reaching the stopping criterion $r^{(t-1)} > r^{(t)}$. OMP is a particular case of SP in which the value of $k = 1$. Note that if k is equal to 1, only one component will be added to the support set in each iteration. Figure 4.8 shows how the NMSE vs number of coefficients curves varies as we change the k.

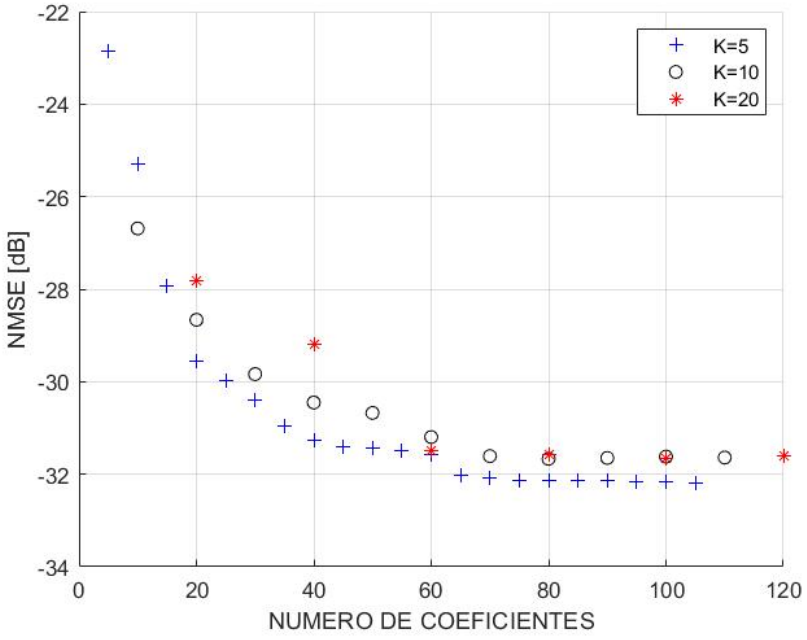

Figure 4.8: Linearization NMSE with different values of k

4.2.5. Random Forest

Random forests is an ensemble learning method for classification or regression that operates by constructing a multitude of decision trees. A decision tree is a type of supervised learning algorithm that is mostly used in classification problems. A tree has many analogies in life and turns out it is influenced in a wide area of machine learning covering both classification and regression trees. A decision tree is a flowchart like structure where each internal node denotes a test on an attribute, each branch represents an outcome of a test, and each leaf or terminal node holds a class label. The topmost node in a tree is the root node.

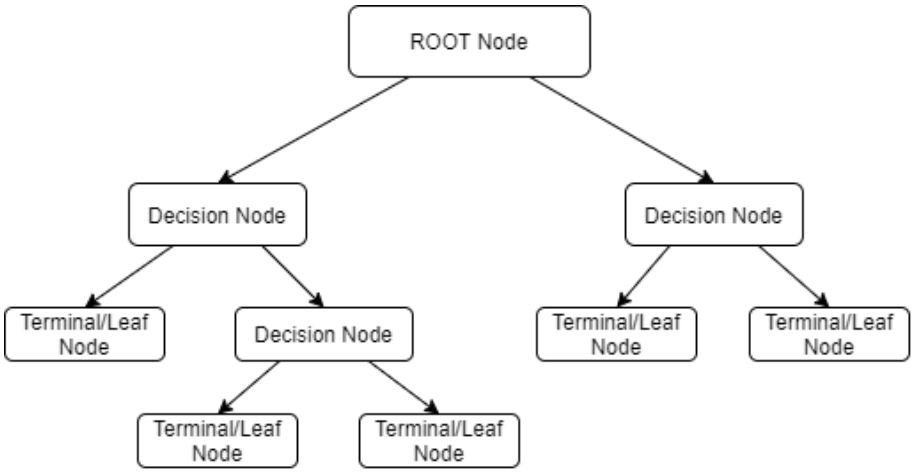

Figure 4.9: Decision tree attributes

In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree like a model of decisions. It can handle both numerical as well as categorical data and nonlinear relationships between parameters do not affect the performance of the algorithm. The main disadvantages are that decision tree learners can create over complex trees that do not generalize the data well, also known as overfitting. Decision trees can become unstable because small variations in the data might result in a completely different tree generated. This is called variance, which is needed to be lowered by methods of bagging and boosting. This can be mitigated by training multiple trees, where features and samples are randomly sampled with replacement.

The random forest algorithm is capable of performing both regression and classification tasks. As the name suggests, this algorithm creates the forest with a number of decision trees. In general, the more trees in the forest, the more robust the prediction and thus higher accuracy. It is going to use the same method of constructing the decision with the information gain to model multiple decision trees to create the forest. In a random forest, we grow multiple trees as opposed to a single tree in the court model. To classify a new object based on attributes, each tree gives a classification and the tree vote for that class. The forest chooses the classification having the most votes over all the other trees in the forests. In [37] an application of the random forest technique to behavioral modeling component selection is presented.

Table 4.1: Input data to the Random Forest algorithm [37]

t	**R_1**	**R_2**	$\cdots$	**R_m**	**NMSE**
1	1	0	$\cdots$	0	$29.81dB$
2	1	0	$\cdots$	1	$22.17dB$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
N_T	0	0	$\cdots$	1	$26.17dB$

For ease of understanding, we will start considering binary decision trees, where variables can only value 0 or 1. In order to obtain homogeneous subsets, a simple technique is to split on a variable and measure the variance of the output when the value of this variable is fixed. The variable with the lowest output variance will be considered as the next one to use as split. Hence, variance will be computed as:

$$var(j) = c_o\ var(s_o) + c_i\ var(s_i), \tag{4.21}$$

where c_i is the amount of samples that belong to the s_i subset of the output, which accomplish that the $j-th$ variable is equal to i. When working with non-binary decision trees, the only difference of this method is that a previous step needs to be made. At binary trees, there is no need to worry about where to split the values of each variable, since there are only two different values. However, when variables are not binary, the first step requires finding the best split point at each variable. This is usually made by going over the whole range of values of the variable, dividing the output dataset and comparing the variance computed as described in (4.21) until the lowest is found. This process continues until the maximum number of splits (known as depth of the tree) is reached.

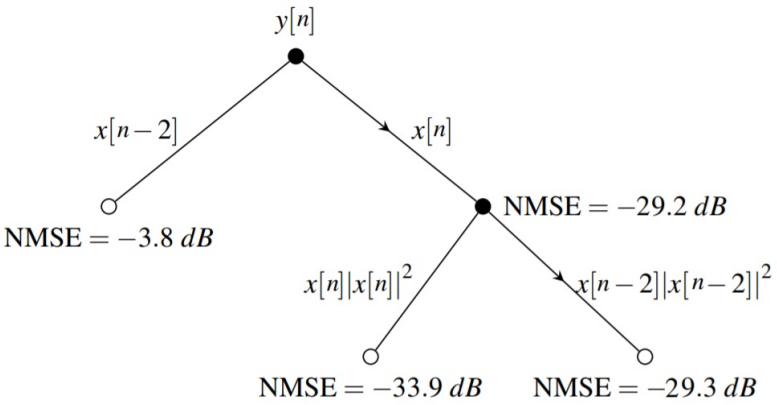

Figure 4.10: Decision tree representation of a GMP model[37]

The input data are the X matrix of regressors containing M columns and N rows, the output signal y and N_T which will be the number of trees. The algorithm returns an output vector with a length equal to M, where each value indicates the importance of a single regressor. In the first iteration, the table 4.1 is generated, and the NMSE of each row is calculated. In the second iteration, the different decision trees are generated, where each decision tree contains its determined splits taking into account the variance. The out-of-bag (OOB) is calculated as the average NMSE of all the decision trees previously calculated. In order to

compute the output vector Imp, the value of each column will be modified while leaving the rest of the columns intact, and the new NMSE obtained for each case will be calculated. The importance will be the subtraction between the NMSE for that case and the OOB.

Algoritmo 6: Random forest algorithm [37]

INPUTS: X, y, N_T = number of trees
OUTPUT: Imp
Initialization:
for $i \leftarrow 1$ **to** N_T **do**
    Create table with M/2 random regressors
    Calculate NMSE of each model
end
for $tree \leftarrow 1$ **to** N_T **do**
    Use 2/3 of the length of the input signal:
    for $split \leftarrow 1$ **to** M **do**
        Randomly select $M/3$ regressors of the model
        Split in the regressor with the lowest variance
    end
    Calculate NMSE of the tree
end
Compute the OOB as $\sum_{t=1}^{N_T} \frac{NMSE_t}{N_T}$
for $f \leftarrow 1$ **to** M **do**
    Shuffle values of column f and leave the rest of the table unchanged to obtain X'_f
    Calculate $NMSE_f$ of regressor matrix X'_f
    Compute importance regressor as: $Imp(f) = NMSE_f - OOB$
end

The indexes of regressors can be ordered from highest to lowest once the importance of each regressor is calculated. Figures 4.11 shows the variation of NMSE as a function of the number of coefficients. In order to see how the N_T variable affects the NMSE, three plots have been made, where each one of them represents the NMSE vs number of coefficients curve for a given number of trees.

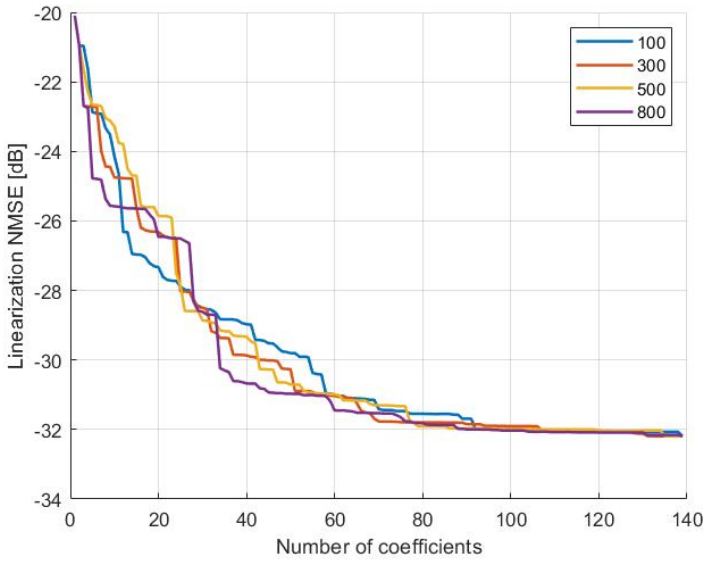

Figure 4.11: Random Forest algorithm with different number of trees

4.3. Heuristic search algorithms

Before calculating the coefficients of a DPD and applying it to a PA, it is necessary to determine the best structure for the DPD. The objective is to find the structure that allows for the best trade-off between good linearization performance and low complexity. For example, the structure of an MP model is fixed by 2 parameters, the non-linearity order K and the memory length L. It is easy to test all possible values in a given bounded range of values, e.g. $K < 11, L < 11$ requires $11^2 = 121$ tests. Compared to the MP model, the GMP model improves the modeling accuracy significantly and achieves a better trade-off for accuracy versus complexity. It is a good choice for DPD and PA modeling. But its structure is determined by 8 integer parameters. So the determination of the structure by an exhaustive search implies a very high computational cost. For example, if all the parameters are bounded to a maximal value of 7 there are 7^8 tests to do.

Heuristic algorithms are designed to solve a problem in a faster and more efficient way than traditional methods by sacrificing precision for speed. Employing a cost or merit function, we give importance to each of the variables to be tuned. In this project, we try to find a configuration that offers the best performance with the minimum possible computational cost. Therefore it is translated in a trade-off between NMSE and number of coefficients. These algorithms start from a set of several models to find the model that best fits the defined cost or merit function. Since we try to get a graph where we can see the NMSE vs the number of coefficients curve, we will make an adaptation where we will save all the calculated models until we get the optimal configuration. With all these combinations in the same graph, a Pareto front will be made to obtain the best result for each number of coefficients. The cost function to be applied will have the following structure:

$$J_i = Nc_i + \lambda(NMSE_i - NMSE_{threshold}), \tag{4.22}$$

where J_i indicate the value of the cost function of model i, Nc_i represent the number of coefficients, $\text{NMSE}_{threshold}$ and λ are constants values that will be used to adjust the NMSE

importance. To approximate λ value the chosen subset is the set of all MP model structures with memory order p_a bounded by $p_{a,max}$ and memory length m_a bounded by $m_{a,max}$. This subset contains $p_{a,max}m_{a,max}$ MP structures (with is much smaller than the number of GMP structures). We calculate the NMSE for all the element in this subset, the minimum NMSE value obtained for these models is labelled as NMSE_{min} and $\text{NMSE}_{threshold}$ =NMSE_{min}· 0.8. Once $\text{NMSE}_{threshold}$ has been calculated, the system of equations can be solved and λ calculated as follows:

$$\lambda \geq \frac{Nc_i - Nc_j}{NMSE_j - NMSE_i} \tag{4.23}$$

To obtain the values ofλ and NMSE, a MP subset has been used where $k_{a,max}$ is bounded by 2 and $l_{a,max}$ is bounded by 12. This gives a total combination of 24 models to be calculated. Once the calculations of (4.23) have been made, the results obtained are: NMSE_{min} =-25.76 dB and $\lambda \geq 8.9$.

Despite this equation has been obtained in an approximate way, the λ has been tuned to obtain better results. After an exhaustive test of values, $\lambda = 15.2$ is the one that has given the best results. Therefore, the cost function to be used for the analysis of the heuristic algorithms is:

$$J_i = Nc_i + 15.2(NMSE_i - (-25.76)) \tag{4.24}$$

In order to evaluate the performance of some selected heuristic algorithms, we restrict the configuration parameters of the GMP to the following intervals:

- $p_a = [6\ 7\ 8\ 9];\ p_b = [2\ 3\ 4\ 5];\ p_c = [2\ 3\ 4\ 5]$
- $m_a = [0:1:10];\ m_b = [-4:1:5];\ m_c = [-4:1:5]$
- $l_b = [1:1:4];\ l_c = [1:1:4]$

4.3.1. Hill Climbing

Hill climbing (HC) algorithm is a local search algorithm which continuously moves in the direction of increasing elevation to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak values where no neighbor has a higher value. The solution obtained may not be the global optimal maximum or the best solution for the problem but it is basically the best possible solution in a very reasonable period of time. This implies that Hill climbing solves the problem where we need to maximize or minimize a given function by choosing values from the given inputs. A very good example of this is the travelling salesman problem where you need to minimize the distance travel by the salesman. Figure 4.12 shows the flowchart of Hill climbing. An algorithm based on HC search for the GMP model structure which provides the best trade-off between modeling accuracy and its complexity is presented in [31]. Applying the previous idea to our problem, in a discrete set, each node x_i is assigned to a unique GMP model structure. The coordinate of x_i consists of 8 dimensions: p_a^i, p_b^i, p_c^i, m_a^i, m_b^i, m_c^i, l_b^i, l_c^i. The value of a cost function J_i is associated to each node x_i.A neighbor of node x_i is defined in [31] as a node of which parameters are $p_a^i + \delta_1$, $p_b^i + \delta_2$, $p_c^i + \delta_3$, $m_a^i + \delta_4$,$m_b^i + \delta_5$, $m_c^i + \delta_6$, $l_b^i + \delta_7$, $l_c^i + \delta_8$ where $\delta \in [0, \pm 1]$ and only one of the parameters can be different to zero at a same time.

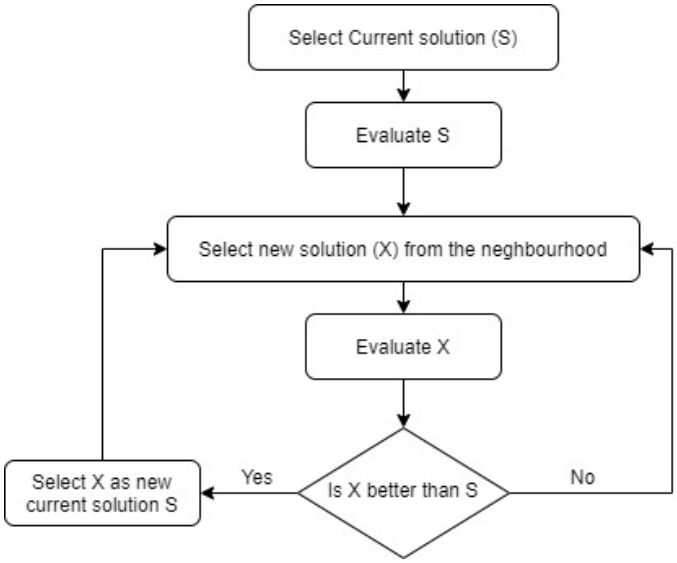

Figure 4.12: Hill climbing flowchart [31]

The search procedure is described in algorithm 7. As input data, the variable R is a matrix that in each row indicates the values of the parameters and the variable x_k is the point from which the algorithm will start. As output data, the algorithm returns the variable x_b that indicates the best node that the algorithm has been able to find taking into account the given cost function. The first step is to evaluate the cost function for the x_k node and set this node as the current node. Once the node on which the algorithm will focus is selected, all the possible combinations of neighbours for that node will be generated. The cost of each of the neighbouring nodes will be calculated and the one with the lowest cost will be compared with the current node. If the selected neighbor node has a lower cost than the current node, it means that better results have been found and therefore the neighbor node is updated as the current one. In case the current node is the best among all its neighbors, the loop is considered completed. Finally the x_b node will be the last best node found during the loop.

Algoritmo 7: Hill climbing algorithm [31]

INPUTS: R, x_k
OUTPUT: x_b
Initialization: Select a initial node x_k
Calculate the J_o with the cost function
Set initial node as the current node
while *parameters inside the restricted range* R **do**
    Generate a set of neighbors according to the definition of neighbor
    Evaluate the cost function of current node neighbors
    Select the neighbor with the lowest cost function
    if *Neighbor's cost is lower than current node cost* **then**
        Set neighbour as current node
    else
        end loop
    end
end
Take current node as the solution x_b

Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation, this implies that the results obtained are not the best and that they are very sensitive to the initial conditions. In this algorithm the initial condition is the point x_k. Therefore we will try several x_k values to see how the results varies (see Figure 4.13). Table 4.2 shows the results obtained starting from three different x_k values until reaching the neighbor x_b with the best cost function.

The first start point $x_{k,1}$ has the following parameters

- $p_a = [6];\ p_b = [2];\ p_c = [2]$
- $m_a = [0:1:3];\ m_b = [-4:1:0];\ m_c = [-4:1:0]$
- $l_b = [1];\ l_c = [1:1:3]$

The second start point $x_{k,2}$ has the following parameters

- $p_a = [7];\ p_b = [3];\ p_c = [5]$
- $m_a = [0:1:6];\ m_b = [-4:1:2];\ m_c = [-4:1:0]$
- $l_b = [1];\ l_c = [1:1:4]$

The third start point $x_{k,3}$ has the following parameters

- $p_a = [8];\ p_b = [3];\ p_c = [3]$
- $m_a = [0:1:8];\ m_b = [-4:1:3];\ m_c = [-4:1:3]$
- $l_b = [12];\ l_c = [12]$

Table 4.2: Hill Climbing results

Initial point x_k	**Best point** x_b	**cost**	**Nº coefficients**	**NMSE**
$p_a^k = 6,\ p_b^k 2,\ p_c^k = 2;$ $m_a^k = [0:1:4],\ m_b^k = [-4:1:0], m_c^k = [-4:1:0];$ $l_b^k = 1, l_c^k = [1\ 2\ 3];$	$p_a^b = 6,\ p_b^b = 2,\ p_c^b = 2;$ $m_a^b = [0\ 1\ 2], m_b^b = [-4:1:0], m_c^b = [-4:1:0];$ $l_b^b = 1,$ $l_c^b = [1\ 2\ 3];$	-106.23	38	-30.11 dB

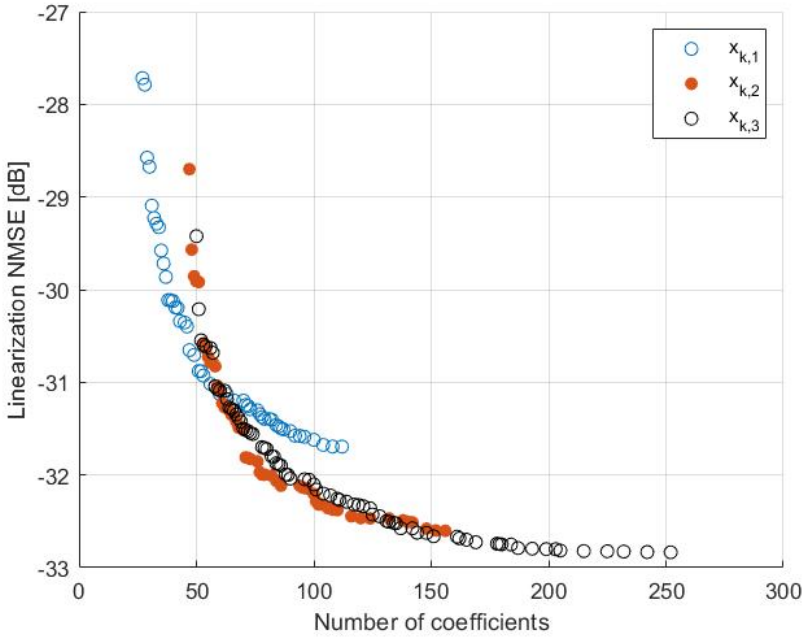

Figure 4.13: Hill climbing with different x_k

4.3.2. Genetic Algorithm

Genetic Algorithms (GA) are adaptive methods which may be used to solve search and optimization problems. They are based on the genetic processes of biological organisms. Over many generations, natural populations evolve according to the principles of natural selection and survival of the fittest. In nature, individuals in a population compete with each other for resources such as food, water and shelter. Those individuals which are most successful in surviving will have relatively larger numbers of offspring. Poorly performing individuals will produce few of even no offspring at all. This means that the genes from the highly adapted, or fit individuals will spread to an increasing number of individuals in each successive generation. GAs work with a population of individuals, each representing a possible solution to a given problem. Each individual is assigned a fitness score according to how good a solution to the problem it is. The fittest individuals are given the opportunities to reproduce with other individuals in the population. This produces new individuals as offspring, which share some features taken from each parent. The least fit members of the population are less likely to get selected for reproduction, and so they are eliminated from the population. A whole new population of possible solutions is then produced by selecting the best individuals from the current generation, and crossing them to produce a new set of individuals. This new generation contains a higher proportion of the characteristics possessed by the good members of the previous generation. In this way, over many generations, good characteristics are spread throughout the population, being mixed and exchanged with other good characteristics and the most promising areas of the search space are explored. If the GA has been designed well, the population will converge to an optimal solution to the problem.

Before a GA can be run, a suitable coding for the problem must be done. We also require a fitness function, which in our case we take the cost function defined in (4.24), which

assigns a cost to each of the coded solution. During the run, parents must be selected for reproduction, and recombined to generate offspring.

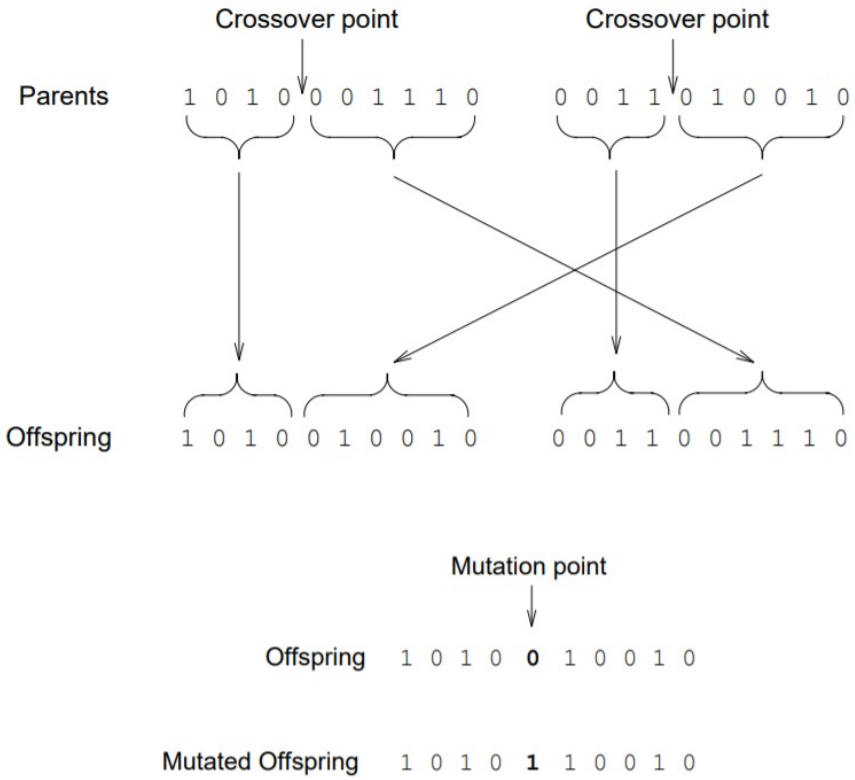

Figure 4.14: Graphical representation of crossover and mutation [26]

During the reproductive phase of the GA, individuals are selected from the population and recombined, producing offspring which will comprise the next generation. Parents are selected randomly from the population using a scheme which favours the more fit individuals. Good individuals will probably be selected several times in a generation, poor ones may not be at all. Having selected two parents, their chromosomes are recombined, typically using the mechanisms of crossover and mutation.

The most basic forms of these operators are as follows (see figure 4.14): Crossover takes two individuals, and cuts their chromosome strings at some randomly chosen position, to produce two head segments, and two tail segments. The tail segments are then swapped over to produce two new full length chromosomes . The two offspring each inherit some genes from each parent. Mutation is applied to each child individually after crossover. It randomly alters each gen with a small probability, that add robustness to the algorithm.

In the article [25] an integer genetic algorithm for the determination of orders of polynomial series for predistortion or power amplifier modeling.Due to the type of problem we are trying to solve, to perform the integer optimization, standard GA with functions for generating integer population and integer mutations will be used as explained in [25]. Basically we are trying to obtain a vector of integer numbers representing the different GMP models based

on fitness function already defined.

Algoritmo 8: Genetic algorithm [25]

INPUTS: R
OUTPUT: x_b
Initialization: Generate initial random integer populations P of individuals from R
while $generation \leq max\ generations$ **do**
  Evaluate the cost function of all individuals in the current generation
  Find the best individuals
  for *i=1 with step 2 to population size* **do**
    Selection
    Crossover
    Mutation
  end
  Create new population from offspring
  $generation = generation + 1$
  if $Stall$ **then**
    Break;
  end
end

GA algorithm deals with individuals, in this case an individual is a particular GMP model and the population is a subset of models chosen randomly from the models set of possible values R previously defined. Applying the cost function in 4.24, the algorithm will apply the process of selection, crossover and mutation as the generations pass until reaching the maximum number of generations or until a stopping criterion is reached. The best individual according to the cost function will be set as x_b. Table 4.3 shows the results obtained starting from a random model and finding the best individuals x_b parameters and its cost. This mean that the individuals with the lowest cost is equal to -112.7 achieving -30.14 dB with 32 coefficients. The GA Matlab function was used to obtain the results and shown in figure 4.15. Where the crossover fraction is set to 0.8 that means that the fraction of the population at the next generation not including elite children that the crossover function creates is 80%. The elite count is a positive integer specifying how many individuals in the current generation are guaranteed to survive to the next generation and set to 0.05·$PopulationSize$. The maximum number of generations is set to $100 \cdot NVar(8)$ and population size is set to 200.

Table 4.3: Genetic algorithm results

Initial point	**Best point** x_b	**cost**	**Nº coefficients**	**NMSE**
Random	$p_a^b = 6,\ p_b^b = 2,\ p_c^b = 3;$ $m_a^b = [0\ 1\ 2],\ m_b^b = [-4:1:0],\ m_c^b = [-4:1:0];\ l_b^b = 1,$ $l_c^b = [1\ 2\ 3\ 4];$	-112.7	32	-30.14 dB

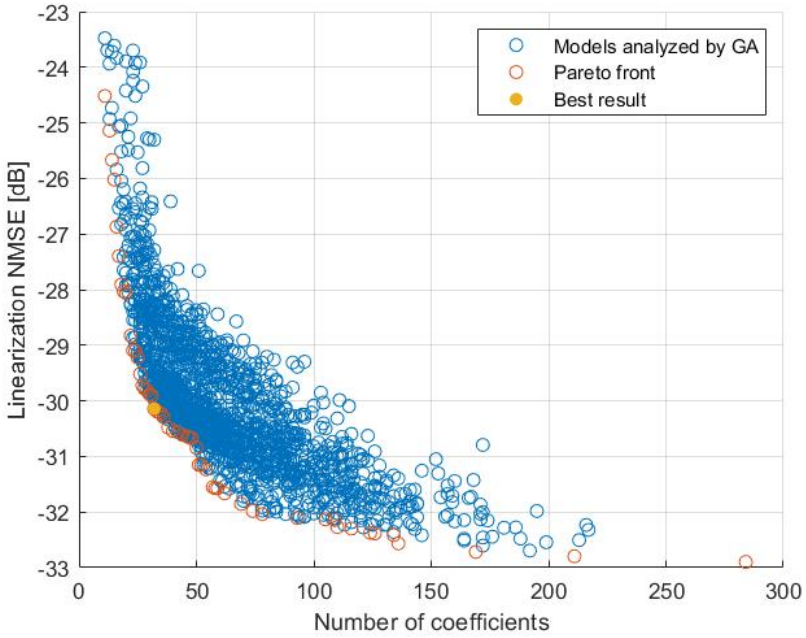

Figure 4.15: Pareto front and best individual for genetic algorithm

4.3.3. Simulated Annealing

The simulated annealing algorithm presented in [33] and applied for the extraction of power amplifier (PA) behavioral model parameters is inspired from the process of annealing in metal work and is used to obtain low energy states over over solid metal. The way annealing works in metal work is as follows: the temperature of a solid metal is increased until the metal melts and becomes liquid after that,in the liquid state it is cooled down until the particles rearranged in the ground state of the solid. This is a simulation of the process of annealing and can be used to generate a solution to combinatorial optimization problems where the cost of a solution is related with the energy of a state. Simulated Annealing is a modified version of hill climbing. Starting from a random point in the search space, a random move is made. If this move takes us to a higher point, it is accepted. If it takes us to a lower point, it is accepted only with probability $p(t)$, where t is time. The function $p(t)$ begins close to 1, but gradually reduces towards zero similar to the process of cooling of a metal as already explained.

$$p(t) = e^{-\frac{\delta(E-E')}{T}}, \tag{4.25}$$

where $\delta(E'-E)$ is the energy difference between current and next model to move and T is the temperature. If $\delta(E'-E)$ is negative, i.e. the transition decreases the energy, the movement is accepted with probability $p = 1$. It is important to note that the condition that the system always switches to a lower energy system when one is found is not at all necessary for the success of the method. When $\delta(E'-E)$ is positive the transition probability $p(t)$ is always different from zero, i.e., the system can move to a higher energy state (worse solution) than the current state. This property prevents the system from being trapped in a local optimum. As the temperature tends to the minimum, the probability of

transition to a higher energy state tends to zero asymptotically. When T reaches zero, the algorithm will only accept changes to lower energy states. Due to this property, the temperature plays a very important role in the control of the evolution of the system. At high temperatures, the system will tend to large energy jumps between the states, while at lower temperatures, the changes in energy will be smaller.

Algoritmo 9: Simulated Annealing algorithm [17]

INPUTS: R
OUTPUT: x_b
Select a random model in R as current model x_c
Evaluate the cost function of x_c
Set initial temperature T
while *Stopping criterion is not met* **do**
  Generate a new neighbor
  Evaluate neighbor cost function J_{i+1}
  if $J_i \leq J_c$ **then**
    Update neighbor model x_i as current model x_c
  else
    update neighbor model x_i as current model x_c with probability p
  end
  Decrease temperature T periodically
end
Set current model x_c as best model x_b

The algorithm initiates by choosing a random model and setting current model. The initial value of temperature T is an important parameter for successful implementation of SA. If the value is too high, then it takes more reduction to converge. If too small, the search process may less than perfect so that the points could potentially global optimum be exceeded. Then a evaluation of a new solution is done. If $J_i \leq J_c$, then new neighbor is accepted and it replaces x_c, update the existing optimal solution. On the other hand, if $J_i \geq J_c$, new neighbor x_i can also be accepted with a probability p.

Table 4.4 shows the results obtained starting from a random model and finding the best model x_b parameters and its cost. This mean that the individuals with the lowest cost is equal to -111.35 achieving -30.45 dB with 38 coefficients. The Simulated annealing Matlab function was used to obtain the results in figure 4.16 with an adaptation for integer variables presented in [17].

Table 4.4: Simulated annealing results

Initial point	**Best point** x_b	**cost**	**Nº coefficients**	**NMSE**
Random	$p_a^b = 6,\ p_b^b = 3,\ p_c^b = 3;$ $m_a^b = [0\ 1],\ m_b^b = [-4:1:4],$ $m_c^b = [-4:1:0];\ l_b^b = 1,\ l_c^b = [1\ 2\ 3\ 4];$	-111.35	38	-30.45 dB

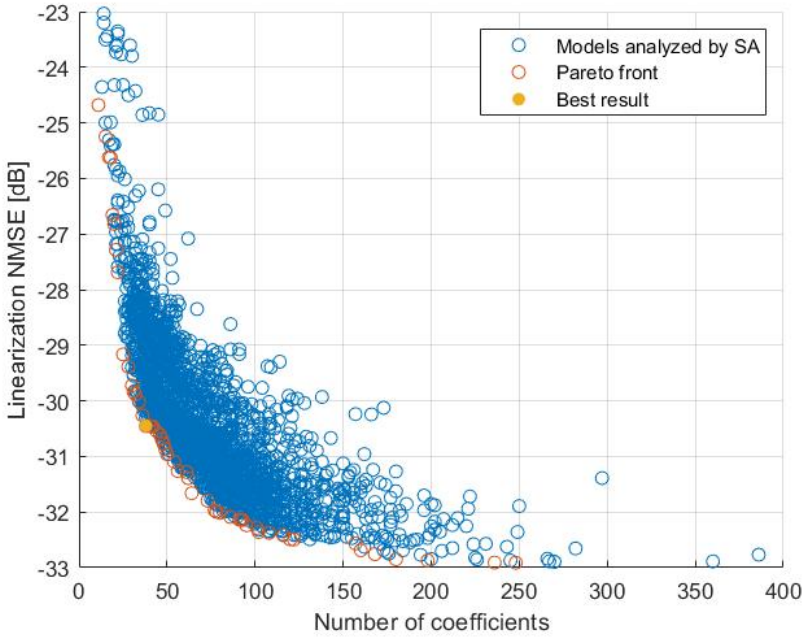

Figure 4.16: Pareto front and best result for Simulated Annealing algorithm

4.3.4. Adalipo

The Adalipo algorithm is an extension of LIPO proposed in [18] which involves an estimate of the Lipschitz constant and takes as input a parameter $p \in (0, 1)$ and a nondecreasing sequence of Lipschitz constant $k_{i\in}$. The algorithm is initialized with a Lipschitz constant $\hat{k}_1$ set to 0 and alternates randomly between two distinct phases: exploration and exploitation. Indeed, at step $t < n$, a Bernoulli random variable B_{t+1} of parameter p which drives this trade-off is sampled. If $B_{t+1} = 1$, then the algorithm explores the space by evaluating the function over a point uniformly sampled over X. Otherwise, if $B_{t+1} = 0$, the algorithm exploits the previous evaluations by making an iteration of the LIPO algorithm with the smallest Lipschitz constant of the sequence $\hat{k}_t$ which is associated with a subset of Lipschitz functions that probably contains f. Once an evaluation has been made, the Lipschitz

constant estimate $\hat{k}_t$ is updated.

Algoritmo 10: Adalipo algorithm [18]

INPUTS: $n,\ k_{i\in},\ \mathcal{X} \subset^{d},\ f \in U_{k\geq 0} Lip(k)$
OUTPUT: $Return\ X_{\hat{i}_n}\ where\ \hat{i}_n\ \in\ argmax_{i=1\cdots n} f(X_i)$
Initialization: Let $X_1 \sim \mathcal{U}(\mathcal{X})$
Evaluate $f(X_1),\ t \leftarrow 1, \hat{k}_1 \leftarrow 0$
while $t < n$ **do**
  Let $B_{t+1} \sim \mathcal{B}(p)$
  if $B_{t+1} = 1$ **then**
    Exploration: Let $X_{t+1} \sim \mathcal{U}(\mathcal{X})$
  else
    Exploitation: Let $X_{t+1} \sim \mathcal{U}(\mathcal{X}_{\hat{k}_t,t}$ where $\mathcal{X}_{\hat{k}_t,t}$ denotes the set of potential maximizers
  end
  Evaluate $f(X_{t+1}),\ t \leftarrow t+1$
  Let $\hat{k}_t = inf\left\{k_{i\in} : \underset{i\neq j}{max} \frac{|f(X_i)-f(X_j)|}{\|X_i - X_j\|_2} \leq k_i\right\}$
end

The algorithm described previously was adapted to solve the problem of finding the best model that fit our cost function. The results obtained can be seen in the figure 4.17 where each one of the blue points represent a model that Adalipo has evaluated and the green points represents the Pareto front. The values of n is set to 15000 so that the number of iterations can't pass this number.

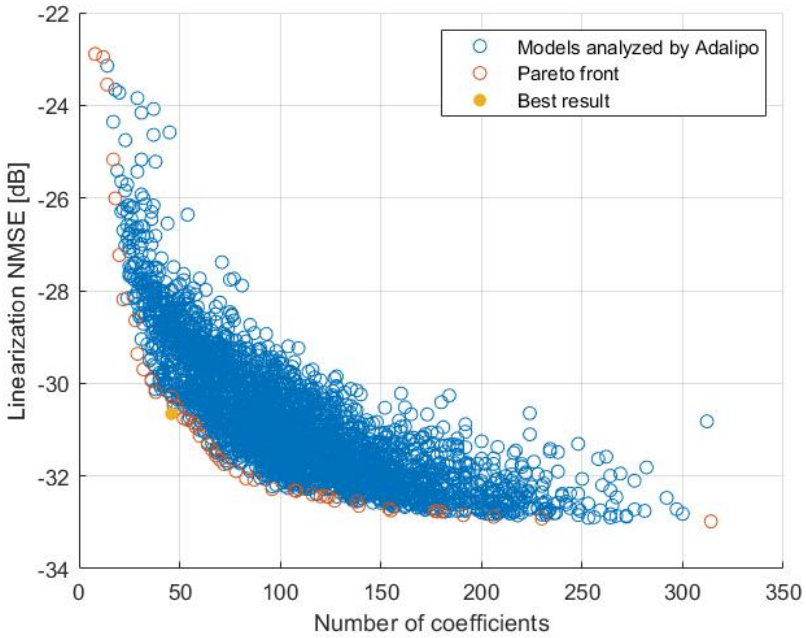

Figure 4.17: Pareto front and best result for Adalipo

4.3.5. Dynamic Model Sizing

Dynamic Model Sizing (DMS) is a model structure adaptation algorithm proposed in [22], which can well address the challenges of adaptive model pruning problems while achieving good pruning performance. The algorithm starts from a given model structure and iteratively searches for a new model suitable for the current PA condition. To achieve the desired objective, the algorithm explores new basis functions that are potentially beneficial for DPD modeling and removes old ones that have a negligible impact on linearization performance. Therefore, the update of the structure mainly consists of two algorithmic steps, namely model pruning and model growing. As we can observe, this algorithm is a combination of greedy pursuit idea of pruning the less important regressors combined with the search for other basis functions outside the first model selected, typical of heuristic algorithms.

The goal of the model pruning is to remove the unimportant terms in the model to make the model more efficient without degrading the performance. To do so, an effective and robust metric to measure the importance of model basis functions must be developed first. The significance of model coefficients can be evaluated using the z-test to evaluate its statistical significance:

$$z_i = \frac{|\, w_i \,|}{\sqrt{\nu_i}}, \tag{4.26}$$

where z_i is the quantification of the importance of regressor $i-th$ regressor,ν_i is the diagonal elements of matrix $(\mathbf{X^H X})^{-1}$ and w_i is the LS estimation of coefficient i. Therefore, based on the vector z, the importance of all coefficients in the current model can be evaluated, and the N_{prune} terms with the smallest z should be removed from the current model. Besides the pruning strategy, it is important to find potentially important terms that are not included in the current model. A straightforward approach is to consider all possible model terms in every iteration, but this will require high computational complexity and can reduce the robustness of model pruning due to a large number of coefficients involved.

It is thus desirable to consider only a subset of the full model terms that are considered useful in increasing model accuracy. The idea is to select only those neighbours of the elements that have been left after making the prune. The concept is very similar to Hill Climbing but with a slight difference. The idea of a neighbor is not a new model but a basis function that has the non-linear terms very close to the remaining basis function after the prune process. Taking MP model as an example, the nonlinear term $|x(n-m)|^{(p-1)}x(n-m)$ has polynomial order p and memory depth m, so it corresponds to the point (p,m) in the feature space. Nonlinear terms that lie in its neighborhood in the feature space, such as the terms corresponding to $(p\pm 1,m)$ and $(k,m\pm 1)$, are likely to provide similar modeling capability of these characteristics. Thus, new terms close to the important basis functions are added to the model during the model growing phase. For initialization, to avoid starting from scratch, the model can start from a predetermined model structure, such as a GMP model. N_ν is the number new terms that will be added after the process of pruning and growing at each iteration. Note that $N_\nu = N_{grow} - N_{prune}$, therefore if the model structure grow by 10 if we want to have 4 new basis functions added to the original set N_{prune} is going to be 6. The stopping criterion set is the number coefficients, so the

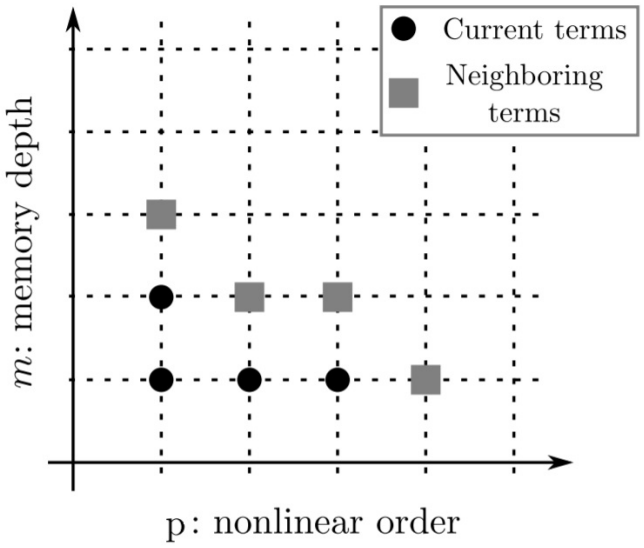

Figure 4.18: Illustration of the model terms being added during model growing [22]

model keep growing until it reach a defined number of coefficients.

Algoritmo 11: Dynamic Model Sizing algorithm [22]

INPUTS: Initial model structure X_o, N_{grow},N_{ν}, y
OUTPUT: Updated model structure X_f
while *stopping criterion is not met* **do**
    Update $N_{prune} = N_{grow} - N_{\nu}$
    Add N_{grow} coefficients to the model from candidate set X_{set}
    for $t = 1$ **to** N_{prune} **do**
        Calculate z
        Select the index with lowest z and remove it from X_o
        Update X_f with the new X_o and calculate $\hat{w}$
        if *Performance degrades dramatically* **then**
            Restore the removed index
            Break
        end
    end
    Generate Neighbors of remaining elements from X_o
    Build candidate set X_{set}
    Update N_{grow}
end

Since DMS starts with an initial GMP model, the same one defined in greedy algorithm has been used. Once algorithm 11 is executed, figure 4.19 can be plotted where x-axis represent the number of coefficients and y-axis the linearization NMSE. In this figure, it can be seen how the NMSE improves when N_{ν} values decreases.

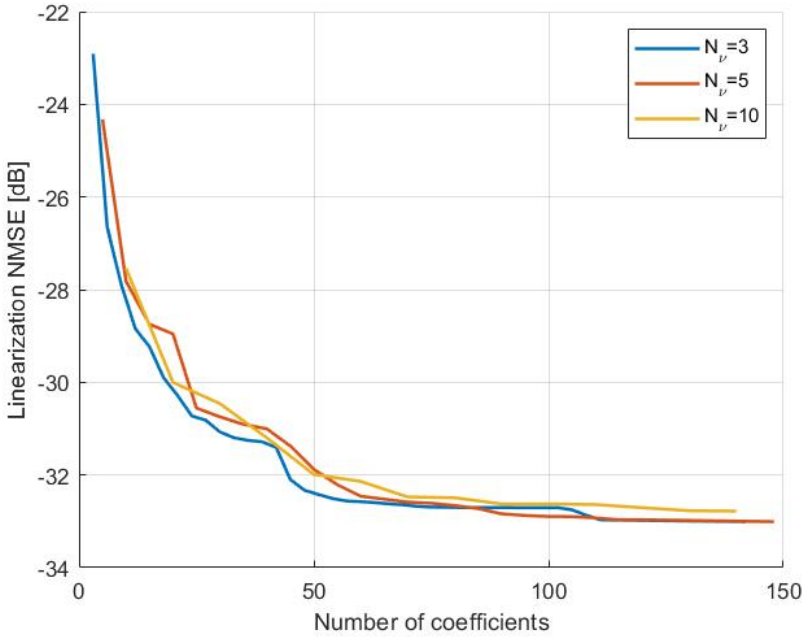

Figure 4.19: NMSE vs Number of coefficients with different N_{ν} values

CHAPTER 5. COMPARISON OF MODEL ORDER REDUCTION TECHNIQUES

In this chapter, we provide a comparison of the results obtained with the different model order reduction algorithms described in Chapter 4. First, we will begin by explaining the type of data used and the test-bench used for obtaining experimental input-output data observations. Then, the metrics for evaluating the modeling performance will be described, so that we can make a comparison on equal terms with the methods analysed in this work. Finally, a comparison among the different reduction methods previously analysed will be made. Since greedy algorithms and heuristic algorithms have a different way of posing the same problem, we will first compare the first group among them and then the heuristic algorithms among them. Finally, a comparison taking into account the algorithms from both approaches will be made discussing the most relevant issues of each of the dimensionality reduction algorithms under study.

5.1. Experimental setup and data used

The experimental data was obtained from a instrumentation-based testbench depicted in figure 5.1. The device under test was a load modulated balanced amplifier (LMBA), designed to operate with high bandwidth signals while keeping high power efficiency values. The signal generation and acquisition equipment consists of an arbitrary waveform generator (AWG) M8190A from Keysight, with a clock rate of 7.9872 GHz and 14 bits, and a digital storage oscilloscope (DSO) 90404A from Keysight operating at 20 GSa/s with 8-bit resolution to capture the RF signal directly. The digital processing and control of the equipment is carried out in a PC running MATLAB.

The RF signal used to extract the PA behavioral models consists in 4-LTE channels of 20 MHz distributed over 200 MHz, as depicted in figure 5.2. A total of 307200 data samples of the digital base-band input-output data were obtained with a sampling frequency of 614.4 MHz. From the total amount of samples, half were used for the estimation of the PA behavioral model coefficients and the other half were used for the validation purposes.

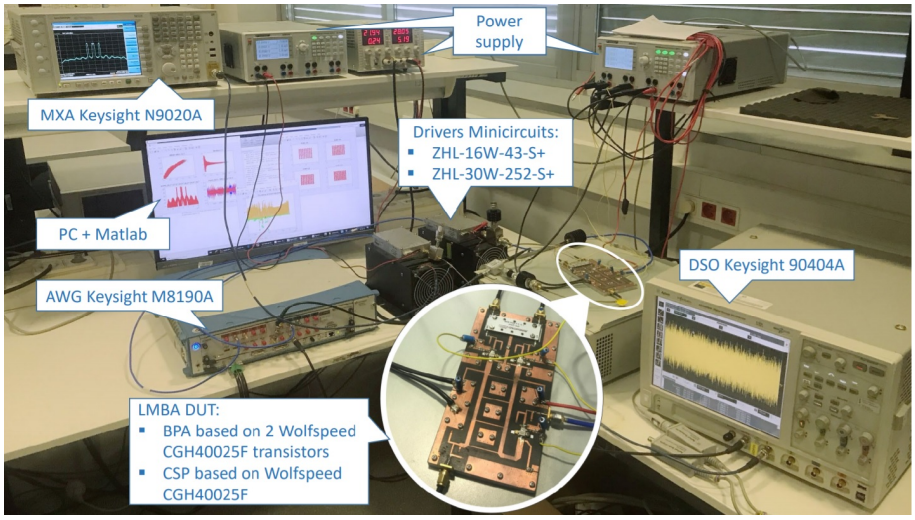

Figure 5.1: Picture of the test setup employed for LMBA experimental validation

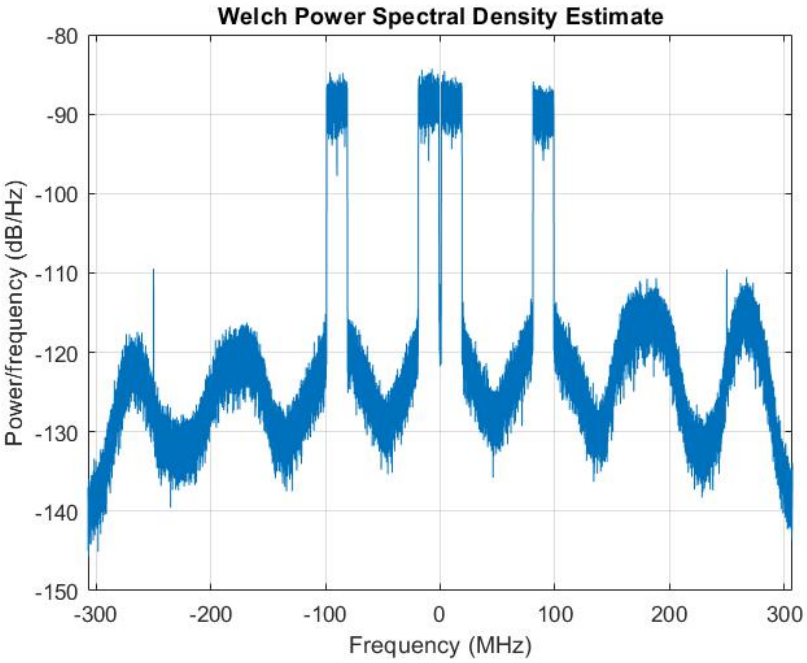

Figure 5.2: RF test signal spectrum

The algorithms used in this project are computationally intensive and the number of samples used for the model coefficients estimation is of importance in order to be able to relax the computational load. Thus, using the whole data set implies a very high running time, making it difficult to perform many tests. In order to solve this problem, two methods for the reduction of the samples used for the estimation of the coefficients will be presented below, namely, brute force and mesh selecting. The results shown throughout this work have been obtained using all samples of the signal, however in the first tests carried out to check the operation and decide some tuning parameters of the algorithms, these two sample reduction methods were used to reduce the required computational time.

As the name suggests, the brute force (BF) method consists of reducing the data that

are used abruptly to make a first study of the signal. As mentioned above, the signal contains 307200 samples of which half of them (153600) are used to extract the model coefficients. This number of samples is quite high and this causes an increase in the run time. The method consists of reducing the identification data by a factor α, $L' = \frac{L}{\alpha}$, where the number of samples used to identify the coefficients are the first L' samples.

On the other hand, with the mesh selecting method proposed in [41], a multi-dimensional mesh is created in order to capture the input data statistics. Later, a proportional pruning of the samples of each of the mesh bins is carried out. As a results, a selection of the most representative input-output data is obtained.

In order to see the difference between these two methods, we have considered a Volterra behavioral model with a parameter configuration that gives a total of 1595 coefficients. Figure 5.3 shows the results obtained when applying OMP for each type of reduced data. The legend indicates the number of samples used to obtain the coefficients.

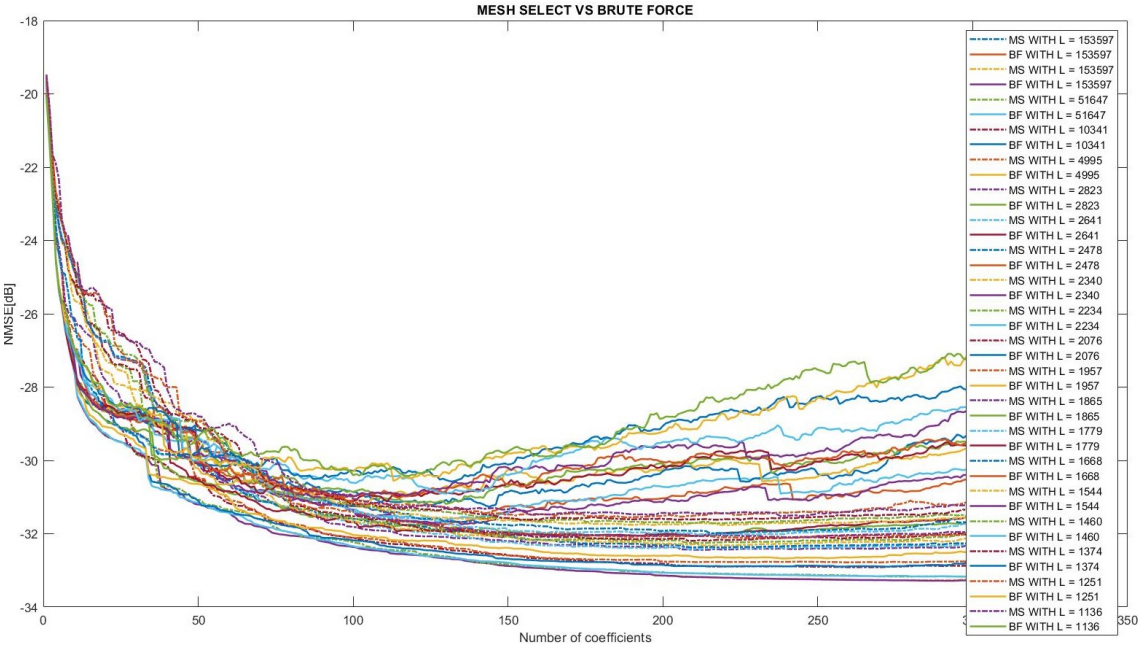

Figure 5.3: Comparison between brute force and mesh selecting

We can see how the mesh selecting method is more robust than brute force method. In brute force, as we reduce the samples, the error grows. It can be seen that for a number of coefficients equal to 200, the BF can vary up to 4 dB depending on the factor we use, being the maximum α_{max} =3000. However, the mesh select under the same conditions only worsens by a total of 2 dB. This indicates that the BF is not very reliable for a first study of the data since it can alter the results. On the other hand, mesh selecting is a method that can help us to make a first study by reducing the execution time in a remarkable way but at the price of a certain degradation of the NMSE.

5.2. Order reduction performance indicators

The algorithms described in Chapter 4 allow us to reduce the number of coefficients of a given behavioral model at the price of some degradation of the modeling performance. In this project, the GMP behavioral model has been considered in order to evaluate the performance of the dimensionality reduction algorithms under study. Once the behavioral model has been chosen, it is crucial to have accurate metrics to evaluate its performance. Several metrics have been proposed for this. This section introduces some of the most commonly used performance indicators of PA behavioral models.

5.2.1. Normalized mean square error

The normalized mean square error (NMSE) indicates how well the model is approximating the reality, i.e., the difference between the estimated and the real (measured) output squared, normalized by the measured output squared. The NMSE is normally expressed in dB:

$$NMSE = 10\log_{10}\left[\frac{\sum_{n=1}^{N} \mid y_{real}(n) - y_{mod}(n) \mid^2}{\sum_{n=1}^{N} \mid y_{real}(n) \mid^2}\right] [dB], \tag{5.1}$$

where $y_{real}(n)$ denotes the measured signal at the PA output, $y_{mod}(n)$ denotes the modeled output and N the number of samples. Since the NMSE is dominated by the in-band error, it is used to evaluate the in-band performance of the model.

5.2.2. Adjacent channel error power ratio

In order to highlight the out-of-band modeling capabilities, the adjacent channel error power ratio (ACEPR) metric is proposed. This metric calculates power of the error (between the modeled and the real signals) in the adjacent channels normalized by the in-band channel power. The ACEPR is normally expressed in dB

$$ACEPR = 10\log_{10}\left[\frac{\int_{(adj)} \mid Y_{real}(f) - Y_{mod}(f) \mid^2}{\int_{ch} \mid Y_{real}(f) \mid^2}\right] [dB], \tag{5.2}$$

where $Y_{real}(f)$ is the Fourier transform of the real output signal, and $Y_{mod}(f)$ is the Fourier transform of the modeled output signal. In the numerator, the operation is done taking into account the adjacent channels, while in the denominator we take into account the transmission channel.

5.2.3. Running time

The Landau notation could be used to measure the complexity of each of the algorithms. The main problem is that, with heuristic algorithms, it is very difficult to make a complexity analysis using Landau notation. A simple approach is to record the running time of the different algorithms. This depends a lot on the hardware from where the code is being

executed. The running time will be obtained using the same hardware to make a fair comparison.

5.3. Greedy algorithms comparison

In this project, we have considered that greedy algorithms are those algorithms that, starting from a given configuration of the behavioral model (considering a lot of coefficients), will find the minimum number of coefficients required for meeting a certain modeling performance. Therefore, it can be said that DOMP, OMP, SP, RF, LASSO and Ridge have the same purpose, but they solve the problem in different ways.

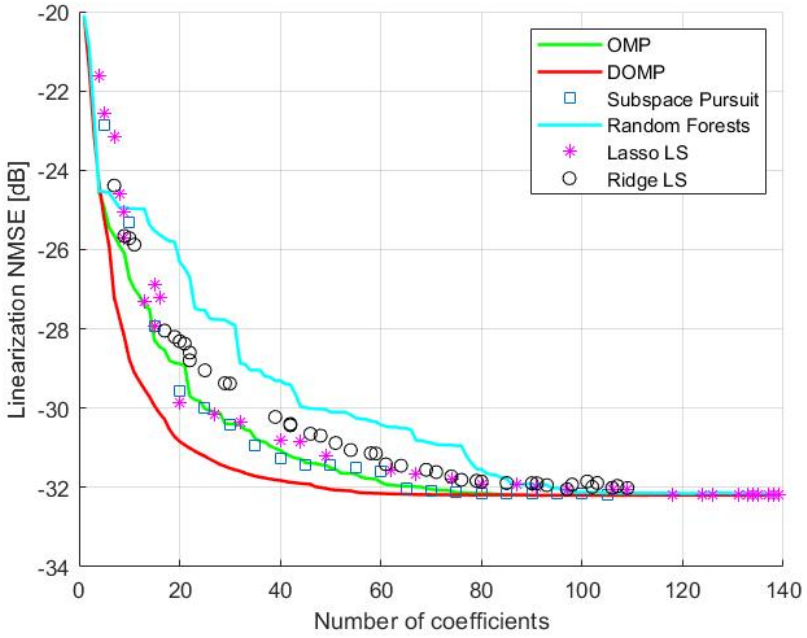

Figure 5.4: NMSE vs Number of Coefficients

As it can be observed in figure 5.4 and figure 5.5 and more concretely in table 5.1, DOMP, OMP and SP are the ones that offer the best performance, reducing the number of coefficients of the model from 139 coefficients to 70 coefficients without almost no loss of performance in terms of NMSE and ACEPR, respectively. That means a reduction of the coefficients of 50% without changing the NMSE more than 0.1 dB. On the other hand, by using the regularization techniques of LASSO and RIDGE, in order to offer the same performance in terms of NMSE, we cannot reduce the coefficients by more than 30%. Finally, Random Forest's algorithm is the one that offers the worst performance. In order to obtain the same NMSE performance, it can reduce the coefficients up to 20%.

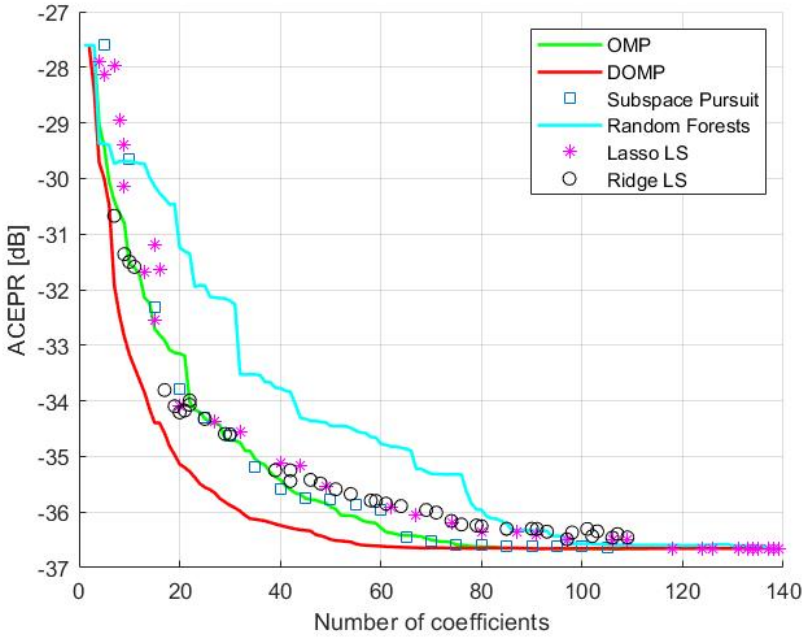

Figure 5.5: ACEPR vs Number of Coefficients

Table 5.1: Minimum number of coefficients required to achieve an NMSE≤ -32 dB and ACEPR ≤ -36.5 dB

CASE	Number of Coefficients	% of Coefficients reduction
SP	70	50%
OMP	70	50%
DOMP	50	64%
LASSO	97	30%
RIDGE	97	30%
Random Forest	97	30%

We can conclude then that the best results have been obtained with the first group of algorithms: DOMP, OMP and SP, since they classify the regressors by correlating them with the residue of the identification error. That allows the importance of each regressor to be quantified and to sort them from the most to least important. SP and OMP work in a very similar way and therefore, their NMSE/ACEPR vs. coefficients curves are very similar. With SP, for each iteration, a subset of the k elements most closely related to the residue is selected. As you can see, for a $k = 1$ we have exactly the OMP, since in each iteration we are adding only one regressor. The higher the k, the fewer iterations the code has to run and consequently slightly worse results are obtained but in a much shorter time. However, with DOMP, a double orthogonalization is guaranteed by including the the Gram-Schmidt process to the original OMP. The selected basis are always orthogonal to the residue, and at the same time, they are orthogonal to the regressors that do not belong to the set of unselected regressors. Including the Gram-Schmidt orthogonalization introduces more computational complexity, which is translated into longer running time, as can be seen

in figure 5.6. The LASSO and RIDGE regularization algorithms are good at solving the problem of bad conditioning and reducing the coefficients, but when trying to obtain the best NMSE for each number of coefficients the best results are not obtained. Equations (4.3) and (4.6) represent RIDGE and LASSO restrictions respectively, where λ is set to obtain the coefficients that meet the conditions. As we increase the λ value, we reduce the number of coefficients that meet the condition, consequently we are worsening the NMSE obtained. Therefore, when we use high values of λ, to be able to obtain the NMSE vs number of coefficients curve, the restriction is so big that we are eliminating coefficients that have high importance. Their running time is generally lower than the DOMP and OMP,

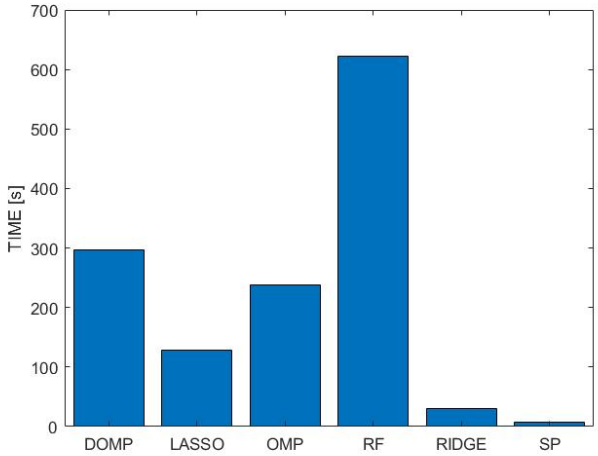

Figure 5.6: Running time greedy algorithm

but they give us a poorer results. The Random Forest (RF) algorithm offers the worst results in addition to having a much longer execution time. Being an algorithm that uses randomness and statistics to obtain results, without looking at the relationship between the regressors and the residual modeling error, it needs many data to obtain satisfactory conclusions. Therefore its execution time is much higher in addition to having the lowest performance.

As mentioned before, we start from a GMP behavioral model configuration with 139 coefficients in total. Before applying the reduction algorithms, the original model offers an NMSE of -32.19 dB, the maximum NMSE that is obtained when considering all the 139 coefficients. SP, LASSO and RIDGE work independently for each number of coefficients; therefore, they do not guarantee models with increasing capacities of error mitigation, consequently, for some coefficients values in figures 5.4 and 5.5, the error does not improve when considering more coefficients.

Table 5.2: Benchmark of greedy algorithms with 20 coefficients

CASE	**NMSE**	**ACEPR**	**NMSE-NMSE$_{opt}$**
SP	-29.56 dB	-33.79 dB	2.49 dB
OMP	-28.88 dB	-33.15 dB	3.17 dB
DOMP	-30.84 dB	-35.14 dB	1.21 dB
LASSO	-29-86 dB	-34.09 dB	2.19 dB
RIDGE	-28.31 dB	-34.29 dB	3.74 dB
Random Forest	-26.31 dB	-30.77 dB	5.74 dB

Table 5.3: Benchmark of greedy algorithms with 50 coefficients

CASE	**NMSE**	**ACEPR**	**NMSE-NMSE$_{opt}$**
SP	-31.45 dB	-35.78 dB	0.6 dB
OMP6	-31.45 dB	-35.91 dB	0.6 dB
DOMP	-32.05 dB	-36.49 dB	0 dB
LASSO	-31.3 dB	-35.54 dB	0.75 dB
RIDGE	-30.8 dB	-35.54 dB	1.25 dB
Random Forest	-30.1 dB	-35.22 dB	1.95 dB

In tables 5.2 and 5.3 , the NMSE$_{opt}$ is set to a value -32.05 dB, slightly higher than the minimum NMSE to avoid bad conditioning. With 20 coefficients, DOMP achieves an NMSE of -30.84 dB, thus obtaining the best possible result with respect to the other algorithms studied, being 1.21 dB worse than the NMSE$_{opt}$. SP, LASSO, RIDGE and OMP get quite similar results, being between 2 dB and 3 dB from the optimal result and finally Random Forest offering much worse results of almost 6 dB to the optimal NMSE. If we increase the number of coefficients needed to 50, we can see that DOMP reaches the optimal NMSE and OMP, SP and LASSO are also very close to the optimal since they have already added the most important regressors. Random Forest and RIDGE are between 2 and 1 dB of the optimum result respectively, so despite offering worse performance than the previous algorithms, they have managed to reduce most of the coefficients with a decent NMSE.

5.4. Heuristic algorithms comparison

As mentioned before, heuristic search algorithms find the best model from a large set of possible configurations. Therefore, the comparison of heuristic algorithms in figure 5.7 and figure 5.8 show the best GMP behavioral model configuration (i.e., the Pareto front) taking into account a certain number of coefficients.

Unlike the rest, hill climbing (HC) is a deterministic algorithm looking at each iteration to the neighbour with the best merit function. Consequently, it can be sensitive to local extrema. On the other hand, the GA, Adalipo and SA algorithm may be seen as a probabilistic

searches. Genetic algorithm (GA) starts from random initial population randomly spread throughout the whole solution space. Thus, it could be quite sensitive to the size of the population which in turn has an impact on the complexity. Simulated annealing (SA) uses the metallurgical term, this technique converges to a solution in the same way metals are brought to minimum energy configurations by increasing grain size. In each iteration, it probabilistically decides whether to stay in the current state or move to another state, and finally make the system enter the lowest energy state. As observed in Figure 5.7 and 5.8, HC is not showing promising results, that is because of its facility to get stuck in a local minimum.

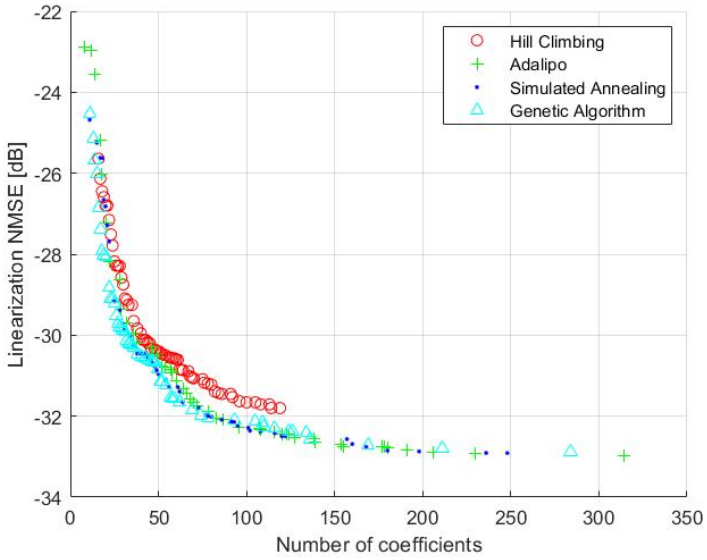

Figure 5.7: NMSE vs Number of Coefficients

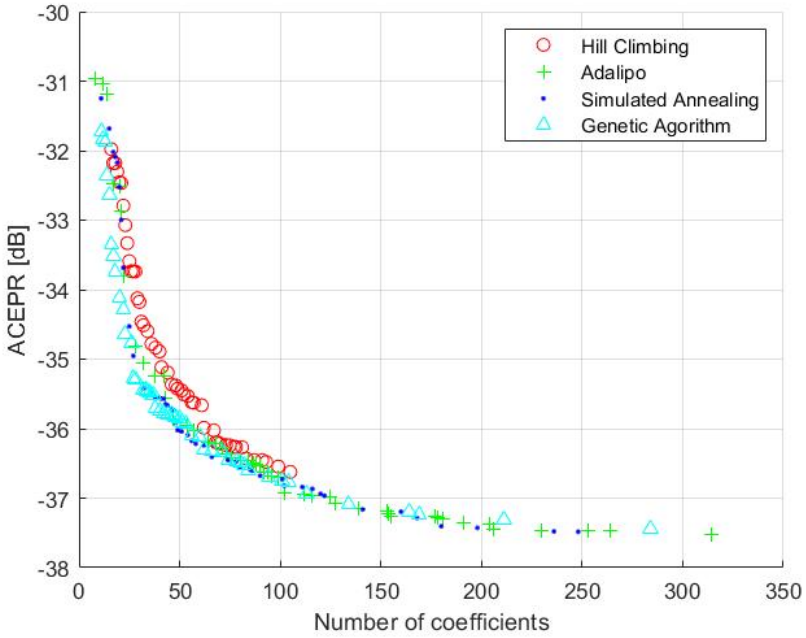

Figure 5.8: ACEPR vs Number of Coefficients

The main problem with HC is that in order to find good results, the initial conditions has to be modified by varying the initial point x_k as explained in chapter 4. However, GA, SA and Adalipo can find better results due to the randomness they offer. Since they do not focus on a single point, it can be seen that the three aforementioned algorithms can reach models with much higher number of coefficients, since they consider that they fit the cost function used. The running time of the heuristic algorithms is directly linked to the number of models that they analyze during that time, but it is also related to the number of coefficients that each model has. Studying a model with more coefficients means that it takes longer to find the optimal solution. With an exhaustive search, the number of models to be analysed would be greater than 10^6.

Table 5.4 shows that HC analyzes less than 4000 models to get the result, and focuses on models that contain less than 130 coefficients in figure 5.9. Unlike HC, the running time of the other methods can be twice or three times HC's, because they analyze models that can reach up to 300 coefficients while increasing number of models under study. Consequently, their running time increases. The use of heuristic search methods has made possible to reduce the number of models under study from 10^6 to 15000 in the most extreme case, offering very good NMSE and ACEPR performance. In the case of making the first study to evaluate the characteristics of predistortion, HC can be a good option because it offers indicative results in a shorter time. Nevertheless, if NMSE or ACEPR is critical in our project, Adalipo, GA, or SA would be the most indicated to use, sacrificing more time.

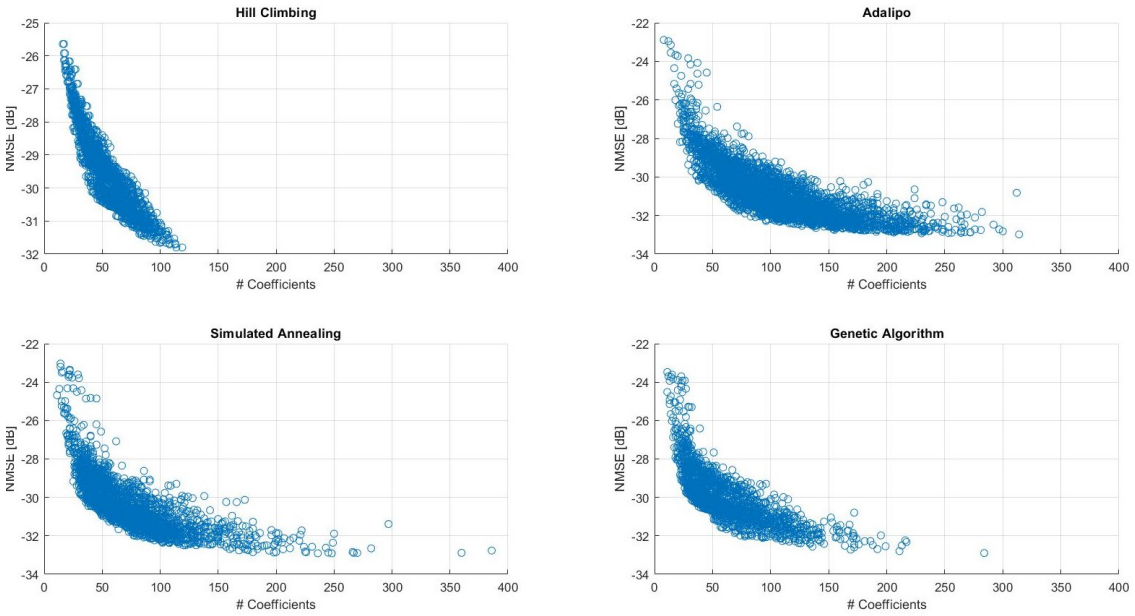

Figure 5.9: NMSE vs Number of coefficients for each of the calculates models

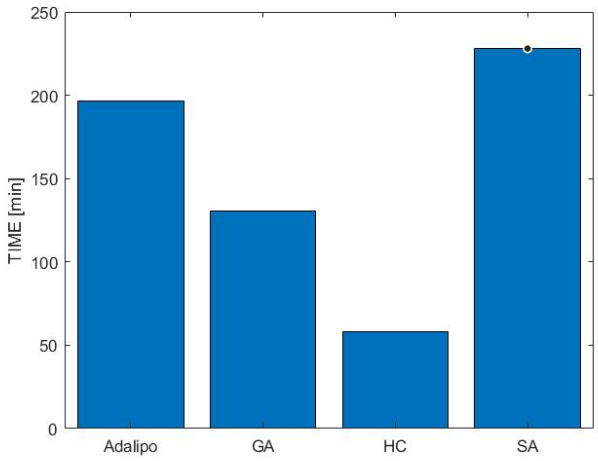

Figure 5.10: Running time heuristic algorithm

5.5. Greedy versus Heuristic algorithms

After seeing in detail how these two families of dimensionality reduction algorithms function, we will discuss some of the most critical points to take into account while providing a comparison of their modeling performance. The main difference between greedy and heuristic search algorithms is that they address two different problems. Greedy algorithms start from an initial configuration of the model and select the most relevant basis in order to have the best NMSE and ACEPR with the minimum necessary coefficients. On the other hand, the heuristic search algorithms try to find the best model configuration among a large set of possible configurations that best fits the defined cost function. For this reason,

the greedy algorithms will have a much lower running time, as it can be observed in table 5.4.

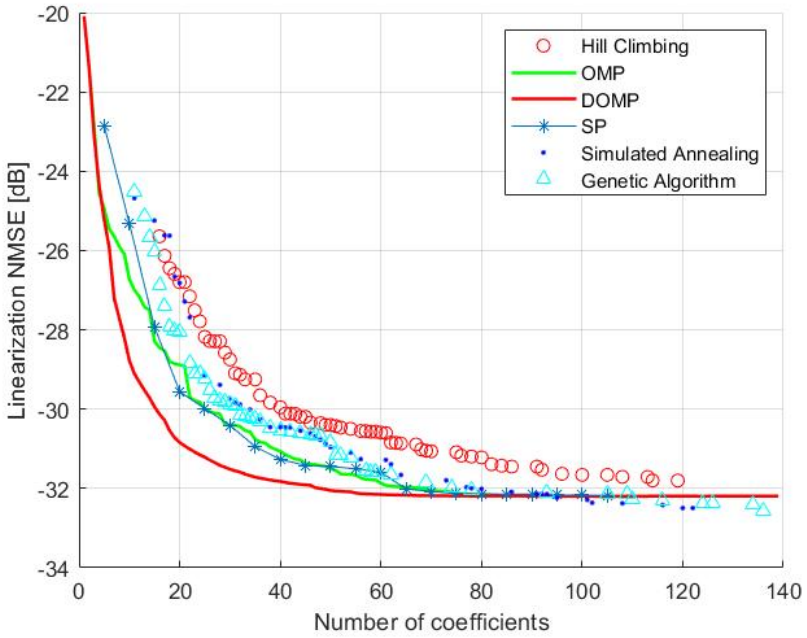

Figure 5.11: NMSE greedy vs heuristic algorithm

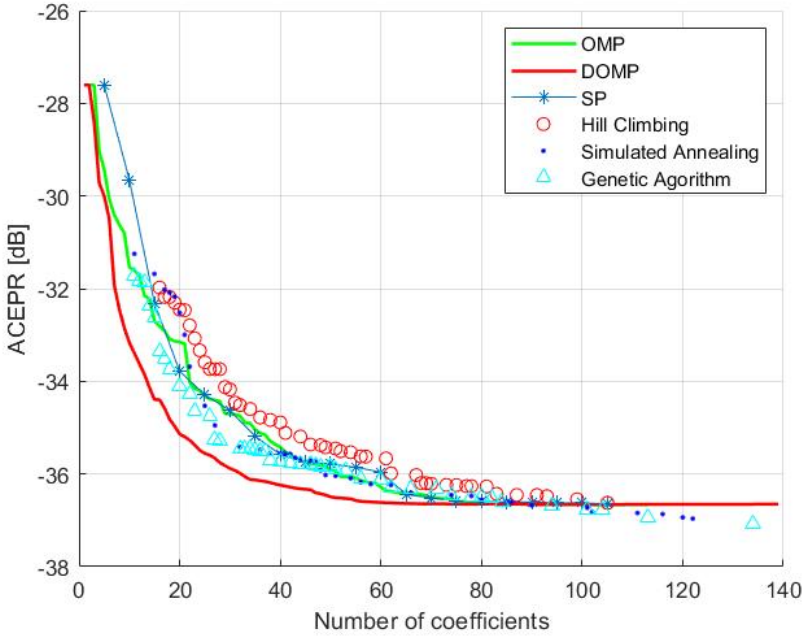

Figure 5.12: ACEPR greedy vs heuristic algorithm

Table 5.4: Benchmark of greedy and heuristic algorithm

CASE	Best NMSE	Models Analyzed	Running Time
SP	-32.19 dB	1	6.8 s
OMP	-32.19dB	1	238.1 s
DOMP	-32.19 dB	1	296.6 s
HC	-31.8 dB	3888	2361 s
SA	-32.91 dB	8238	13683 s
GA	-32.9 dB	5521	7833 s

Figure 5.11 and 5.12 shows a comparison between the three best greedy algorithms vs the three best heuristic search algorithms. For numbers of coefficients less than 80, it can be observed that the greedy algorithms tend to offer better performance. Since the greedy algorithms start from a initial configuration of the model, the NMSE that can achieve cannot be better than the one already offered by the model with all the coefficients. For that reason, as we pass the 80 coefficients, the heuristic algorithms start to be slightly better. In order to make a fairer comparison between the two families, we can set a target number of coefficients and determine the algorithm that provides the best modeling performance (in terms of NMSE and ACEPR) with the configuration found. For this purpose, the DOMP,OMP and GA algorithms will be used with a modified cost function,

$$J_i = \mid Nc_i - Nc_i \mid 50 \tag{5.3}$$

where Nc_k is the number of coefficients the model will focus on, and Nc_i is the number of coefficients of the model i. The equation has its minimum point in $Nc_i = Nc_k$, and any other value different from J_k implies a very high cost. Therefore the model will be able to analyze numerous models with a number of coefficients equal or close to Nc_i.

Table 5.5: DOMP, OMP and GA performance with 10, 20, 40 and 100 coefficients

CASE	Number of Coefficients	NMSE	Models Analyzed	Running Time
DOMP	10	-28.74 dB	1	2.26 s
	20	-30.84 dB	1	11.23
	40	-31.82 dB	1	27.68 s
	100	-32.19 dB	1	113 s
OMP	10	-26.73 dB	1	1.785 s
	20	28.9 dB	1	4.63 s
	40	-31.06 dB	1	17.31 s
	100	-32.19 dB	1	130 s
GA	10	-26.5 dB	281	312 s
	20	-29.9 dB	390	475 s
	40	-30.6 dB	506	721 s
	100	-32.3 dB	731	1071 s

After adjusting the cost function, table 5.5 shows how DOMP and OMP algorithms still have a much lower computational cost and offer a better NMSE and ACEPR in a shorter time;

however, they do not explore new basis functions of other models. On the other hand GA, despite exploring less models than the first cost function used, still have a higher running time than OMP and DOMP. Greedy and heuristic algorithms have their advantages and disadvantages; besides, they solve two different problems.

Finally, we include in the comparison the dynamic model sizing (DMS) algorithm previously explained in chapter 4. The DMS combines the best of both families of dimensionality reduction algorithms. Similarly to OMP or DOMP it provides a sorted list of the most relevant regressors taking into account the z-test to prune the less relevant coefficients; and similar to HC it grows including neighbors of the remaining elements' neighbors. The performance of DMS is shown in figures 5.13 and 5.14. With a running time of 2796 seconds, DMS shows the best results, applying the idea of looking for the importance of the regressor, typical of greedy algorithms, and the concept of analyzing models different than the initial one, typical of heuristic algorithms, obtaining a NMSE_{opt} of -33 dB for 140 coefficients.

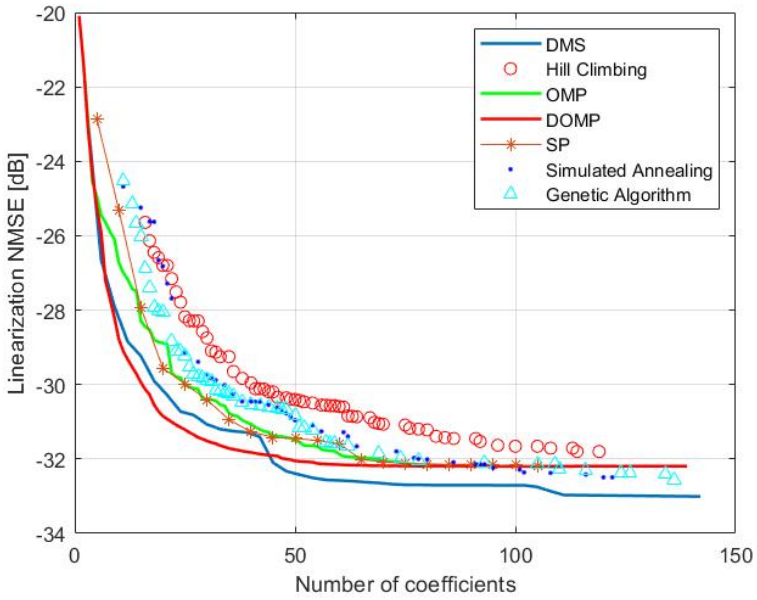

Figure 5.13: NMSE DMS vs greedy and heuristic algorithms

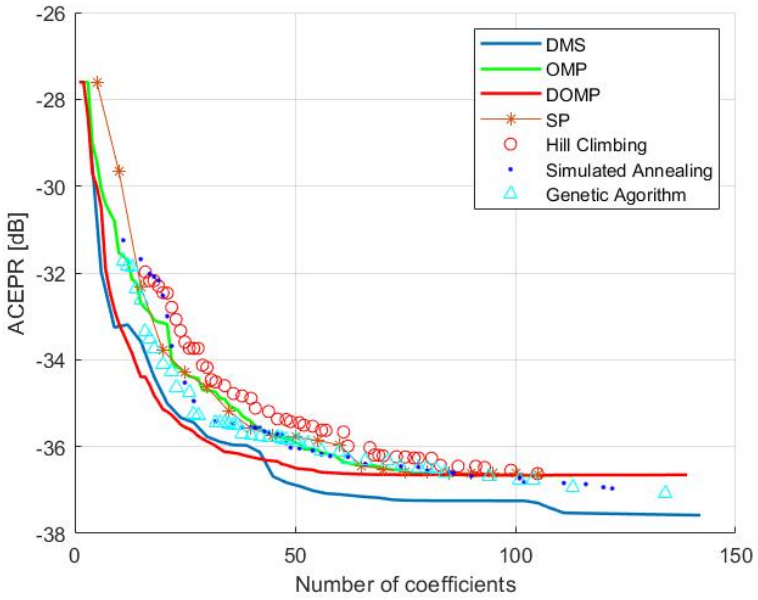

Figure 5.14: ACEPR DMS vs greedy and heuristic algorithms

CHAPTER 6. CONCLUSION

When it comes to improving the transmission rate in an RF system, energy consumption and system efficiency become key factors. We know that PA is the most energy-consuming element in the transmission chain and that by improving its performance, the overall system improves. Using DPD techniques, the amplifier's efficiency is greatly improved as it can work in close to compression. A proper selection of the PA behavioral modeling is crucial for DPD linearization. The use of behavioral models with many coefficients can cause overfitting or poor conditioning which ultimately would degrade the linearization performance. In addition, having a very high number of coefficients will result in high computational complexity and unnecessary power consumption from the digital signal processor.

There are several dimensionality reduction methods to address this problem. In this work, we have focused on greedy algorithms and heuristic search algorithms. These two families of algorithms work in different ways, and each one has its advantages or disadvantages depending on the problem to be treated. A comparison in terms of run time is not completely fair since heuristics search algorithms analyze several models over a much more extended time, but this allows us to compare several results.

On the other hand, the greedy algorithms can reduce the number of coefficients of a model up to 65% without degrading the NMSE or ACEPR. This can leads us to conclude that combining some of the strengths of both families of algorithms can allow us to obtain even better results. The DMS algorithm is one example of this concept and manages to obtain, together with DOMP for low number of coefficients, the best possible performance in terms of NMSE and ACEPR.

As future work to be done, we will try to find alternatives to the DMS algorithm, combining the DOMP approach in which the most relevant coefficients are selected from an original configuration but allowing the model to grow and explore alternative basis functions not present in the original configuration set.

BIBLIOGRAPHY

[1] H. Watanabe et al., "High-efficiency X band GaN power amplifier for small satellite downlink system," 2013 IEEE MTT-S International Microwave Symposium Digest (MTT), Seattle, WA, 2013, pp. 1-4, doi: 10.1109/MWSYM.2013.6697795. 3

[2] Di Mauro, Giuseppe Lawn, M. Bevilacqua, Riccardo. (2017). Survey on Guidance Navigation and Control Requirements for Spacecraft Formation-Flying Missions. Journal of Guidance, Control, and Dynamics. 41. 1-22. 10.2514/1.G002868. ix, 4

[3] Gilabert, Pere L., et al. "Machine Learning for Digital Front-End: a Comprehensive Overview." Machine Learning for Future Wireless Communications (2020): 327-381.

[4] R. Neil Braithwaite. A comparison of indirect learning and closed loop estimators used in digital predistortion of power amplifiers. In 2015 IEEE MTT-S International Microwave Symposium. IEEE, may 2015. doi: 10.1109/mwsym.2015.7166826 ix, 18, 21, 23

[5] Gilabert, Pere L., " Multi Look-Up Table Digital Predistortion for RF Power Amplifier Linearization" (2008).

[6] D. R. Morgan, Z. Ma, J. Kim, M. G. Zierdt and J. Pastalan, "A Generalized Memory Polynomial Model for Digital Predistortion of RF Power Amplifiers," in IEEE Transactions on Signal Processing, vol. 54, no. 10, pp. 3852-3860, Oct. 2006, doi: 10.1109/TSP.2006.879264.

[7] P. L. Gilabert et al., "Order reduction of wideband digital predistorters using principal component analysis," 2013 IEEE MTT-S International Microwave Symposium Digest (MTT), Seattle, WA, 2013, pp. 1-7, doi: 10.1109/MWSYM.2013.6697687. 19

[8] Zhu, Anding Brazil, T.J.. (2007). An Overview of Volterra Series Based Behavioral Modeling of RF/Microwave Power Amplifiers. 1 - 5. 10.1109/WAMICON.2006.351917. ix, 8, 10, 11, 16

[9] M. A. Hussein, O. Venard, B. Feuvrie and Y. Wang, "Digital predistortion for RF power amplifiers: State of the art and advanced approaches," 2013 IEEE 11th International New Circuits and Systems Conference (NEWCAS), Paris, 2013, pp. 1-4, doi: 10.1109/NEWCAS.2013.6573671. 16

[10] Chani-Cahuana, Jessica. Digital Predistortion for the Linearization of Power Amplifiers. Diss. Department of Signals and Systems, Chalmers University of Technology, 2015.

[11] Sezgin, Ibrahim Can. "Different Digital Predistortion Techniques for Power Amplifier Linearization." (2016). 15

[12] J. Reina-Tosina, M. Allegue-Martínez, C. Crespo-Cadenas, C. Yu and S. Cruces, "Behavioral Modeling and Predistortion of Power Amplifiers Under Sparsity Hypothesis," in IEEE Transactions on Microwave Theory and Techniques, vol. 63, no. 2, pp. 745-753, Feb. 2015, doi: 10.1109/TMTT.2014.2387852.

[13] T. Q. A. Pham, “Contribution to dimensionality reduction of digital predistorter behavioral models for RF power amplifier linearization,” Tesi doctoral, UPC, Departament de Teoria del Senyal i Comunicacions, 2019.

[14] J. A. Becerra, M. J. Madero-Ayora and C. Crespo-Cadenas, ”Comparative Analysis of Greedy Pursuits for the Order Reduction of Wideband Digital Predistorters,” in IEEE Transactions on Microwave Theory and Techniques, vol. 67, no. 9, pp. 3575-3585, Sept. 2019, doi: 10.1109/TMTT.2019.2928290.

[15] A. S. Tehrani, H. Cao, S. Afsardoost, T. Eriksson, M. Isaksson and C. Fager, ”A Comparative Analysis of the Complexity/Accuracy Tradeoff in Power Amplifier Behavioral Models,” in IEEE Transactions on Microwave Theory and Techniques, vol. 58, no. 6, pp. 1510-1520, June 2010, doi: 10.1109/TMTT.2010.2047920. ix, 11, 13

[16] Guillena Berenguer, Estefania. “Linealización de Un Amplificador Balanceado Con Modulación de Carga Mediante Un Predistorsionador Digital Basado En Redes Neuronales Para Comunicaciones En Vehículos Aereos No Tripulados.” 6 Oct. 2020. Web. 7 Jan. 2021. 27, 28, 32

[17] Abramson, David Randall, Marcus. (1997). A Simulated Annealing code for General Integer Linear Programs. Annals of Operations Research. 86. 10.1023/A:1018915104438.

[18] Malherbe, C. and N. Vayatis. “Global optimization of Lipschitz functions.” ICML (2017).

[19] Blumensath, Thomas Davies, Mike. (2007). On the difference between orthogonal matching pursuit and orthogonal least squares. 45

[20] J. A. Becerra, M. J. Madero-Ayora, J. Reina-Tosina, C. Crespo-Cadenas, J. García-Frías and G. Arce, ”A Doubly Orthogonal Matching Pursuit Algorithm for Sparse Predistortion of Power Amplifiers,” in IEEE Microwave and Wireless Components Letters, vol. 28, no. 8, pp. 726-728, Aug. 2018, doi: 10.1109/LMWC.2018.2845947. 46, 47

[21] J. A. Becerra, M. J. Madero-Ayora, R. G. Noguer and C. Crespo-Cadenas, ”On the Optimum Number of Coefficients of Sparse Digital Predistorters: A Bayesian Approach,” in IEEE Microwave and Wireless Components Letters, vol. 30, no. 12, pp. 1117-1120, Dec. 2020, doi: 10.1109/LMWC.2020.3027878.

[22] Y. Li and A. Zhu, ”On-Demand Real-Time Optimizable Dynamic Model Sizing for Digital Predistortion of Broadband RF Power Amplifiers,” in IEEE Transactions on Microwave Theory and Techniques, vol. 68, no. 7, pp. 2891-2901, July 2020, doi: 10.1109/TMTT.2020.2982165. 30, 31

[23] Zhu, Anding Pedro, Jose Brazil, T.J.. (2007). Dynamic Deviation Reduction-Based Volterra Behavioral Modeling of RF Power Amplifiers. Microwave Theory and Techniques, IEEE Transactions on. 54. 4323 - 4332. 10.1109/TMTT.2006.883243. 29

[24] S. Wang, M. A. Hussein, G. Baudoin, O. Venard and T. Gotthans, ”Comparison of hill-climbing and genetic algorithms for digital predistortion models sizing,” 2016 IEEE International Conference on Electronics, Circuits and Systems (ICECS), Monte Carlo, 2016, pp. 289-292, doi: 10.1109/ICECS.2016.7841189. ix, 48, 49

[25] T. Gotthans, G. Baudoin and A. Mbaye, "Optimal order estimation for modeling and predistortion of power amplifiers," 2013 IEEE International Conference on Microwaves, Communications, Antennas and Electronic Systems (COMCAS 2013), Tel Aviv, 2013, pp. 1-4, doi: 10.1109/COMCAS.2013.6685279.

[26] Beasley, David, David R. Bull, and Ralph Robert Martin. "An overview of genetic algorithms: Part 1, fundamentals." University computing 15.2 (1993): 56-69.

[27] Etzion, Tuvi Ostergard, Patric. (1998). Greedy and heuristic algorithms for codes and colorings. Information Theory, IEEE Transactions on. 44. 382 - 388. 10.1109/18.651069. 42, 43

[28] Kaur, Rajbir, and M. S. Patterh. "Overview of the Linearization Techniques to mitigate the nonlinear effects of Power Amplifier." An International Journal of Engineering Sciences 17 (2016): 479-485. ix, 42

[29] Brinco De Sousa, Pedro Miguel. "Digital Predistortion of Wideband Satellite Communication Signals with Reduced Observational Bandwidth and Reduced Model Order Complexity." 17 Oct. 2014. Web. 7 Jan. 2021.

[30] D. Wisell, J. Jalden and P. Handel, "Behavioral Power Amplifier Modeling Using the LASSO," 2008 IEEE Instrumentation and Measurement Technology Conference, Victoria, BC, 2008, pp. 1864-1867, doi: 10.1109/IMTC.2008.4547349.

[31] S. Wang, M. A. Hussein, O. Venard and G. Baudoin, "Optimal sizing of generalized memory polynomial model structure based on Hill-Climbing heuristic," 2016 46th European Microwave Conference (EuMC), London, 2016, pp. 190-193, doi: 10.1109/EuMC.2016.7824310. ix, 38, 39

[32] P. L. Gilabert et al., "Order reduction of wideband digital predistorters using principal component analysis," 2013 IEEE MTT-S International Microwave Symposium Digest (MTT), Seattle, WA, 2013, pp. 1-7, doi: 10.1109/MWSYM.2013.6697687.

[33] P. L. Gilabert, D. D. Silveira, G. Montoro, M. E. Gadringer and E. Bertran, "Heuristic Algorithms for Power Amplifier Behavioral Modeling," in IEEE Microwave and Wireless Components Letters, vol. 17, no. 10, pp. 715-717, Oct. 2007, doi: 10.1109/LMWC.2007.905628. 44

[34] Rere, L.M. Fanany, Mohamad Ivan Arymurthy, Aniati. (2015). Simulated Annealing Algorithm for Deep Learning. Procedia Computer Science. 72. 10.1016/j.procs.2015.12.114.

[35] J. A. Becerra, D. Herrera, M. J. Madero-Ayora and C. Crespo-Cadenas, "Sparse Model Selection of Digital Predistorters Using Subspace Pursuit," 2018 13th European Microwave Integrated Circuits Conference (EuMIC), Madrid, 2018, pp. 190-193, doi: 10.23919/EuMIC.2018.8539910.

[36] Mozos Ruiz, José Miguel. "Model Order Reduction Techniques for Power Amplifier Digital Predistortion Linearization." 25 July 2016. Web. 7 Jan. 2021.

[37] L. Álvarez-López, J. A. Becerra, M. J. Madero-Ayora and C. Crespo-Cadenas, "Determining a Digital Predistorter Model Structure for Wideband Power Amplifiers through Random Forest," 2020 IEEE Topical Conference on RF/Microwave Power Amplifiers for Radio and Wireless Applications (PAWR), San Antonio, TX, USA, 2020, pp. 50-52, doi: 10.1109/PAWR46754.2020.9036004. ix, xi, 34, 35, 36

[38] Álvarez López, L. (2019). Mitigación de efectos no lineales en sistemas de comunicación basado en técnicas de aprendizaje profundo. (Trabajo Fin de Grado Inédito). Universidad de Sevilla, Sevilla.

[39] A N Tikhonov and V Y Arsensin. Solution of ill-posed problems. V H Winston, Washington DC, 1977.

[40] Robert Tibshirani. Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society, Series B, 58:267–288, 1994.

[41] T. Wang, P. L. Gilabert, "Mesh Selecting for Computational Efficient Power Amplifier Behavioral Modeling and Digital Predistortion Linearization" IEEE Microwave and Wireless Components Letters. Accepted for publication in Oct. 2020 (to be published in late 2020 or early 2021).

22

22

APPENDICES

APPENDIX A. MATLAB CODE

Orthogonal matching pursuit

```
function [U_red,index,n_0,index_tot,n_0_nan]=
    Orthogonal_Matching_Pursuit_and_BIC(U,y)
[L,n_columns]=size(U);
index=zeros(n_columns,1);
BIC=zeros(n_columns,1);
e=y;
ite=1;
disp('Starting OMP+BIC algorithm. Please, be patient...');
n_0_nan=0;
for i=1:n_columns
   [~, pos]=max(abs(U'*e));
   index(i)=pos;
   if(isnan(((U(:,nonzeros(index))'*U(:,nonzeros(index)))^-1)*U(:,nonzeros(
       index))'*y))
       break
   else
       n_0_nan= n_0_nan + 1;
       w=((U(:,nonzeros(index))'*U(:,nonzeros(index)))^-1)*U(:,nonzeros(
           index))'*y;
   end
   y_est=U(:,nonzeros(index))*w;
   e=y-y_est;
   Sigma_e=(L^-1)*norm(e,2);
   BIC(i)=(2*L*log(Sigma_e)+2*i*log(2*L));
   if (i==100*ite); fprintf('\n Number of processed columns= %d \n',i);
       ite=ite+1;
   end
end
   [~,n_0]=min(BIC);
 j=1;
 U_red=U(:,index(1:n_0));
 missing=zeros(n_columns-n_0,1);
 for i=1:n_columns
     if(isempty(find(index(1:n_0)==i)))
     missing(j)=i;
     j=j+1;
     end
 end
 index_tot=[index(1:n_0); missing];

end
```

Doubly Orthogonal Matching Pursuit

```
function [U_red,index,n_0,index_tot,n_0_nan]=
    Doubly_Orthogonal_Matching_Pursuit_and_BIC(U,y)
[L,n_columns]=size(U);
index=zeros(n_columns,1);
BIC=zeros(n_columns,1);
Z=U;
e=y;
ite=1;
disp('Starting OMP+BIC algorithm. Please, be patient...');
```

```
    n_0_nan=0;
    for i=1:n_columns
        [~, pos]=max(abs(Z'*e));
        index(i)=pos;
        p = Z(:,pos)'*Z;
        Z = Z - kron(p,Z(:,pos));
        if(isnan(((U(:,nonzeros(index))'*U(:,nonzeros(index)))^-1)*U(:,nonzeros(
            index))'*y))
            a=1;
        else
            n_0_nan= n_0_nan + 1;
            w=((U(:,nonzeros(index))'*U(:,nonzeros(index)))^-1)*U(:,nonzeros(
                index))'*y;
        end
        y_est=U(:,nonzeros(index))*w;
        e=y-y_est;
        Sigma_e=(L^-1)*norm(e,2);
        BIC(i)=(2*L*log(Sigma_e)+2*i*log(2*L));
        if (i==100*ite); fprintf('\n Number of processed columns= %d \n',i);
            ite=ite+1;
        end
    end
        [~,n_0]=min(BIC);
     j=1;
     U_red=U(:,index(1:n_0));
     missing=zeros(n_columns-n_0,1);
     for i=1:n_columns
         if(isempty(find(index(1:n_0)==i)))
         missing(j)=i;
         j=j+1;
         end
     end
     index_tot=[index(1:n_0); missing];

    end
```

Subspace Pursuit

```
    z = zeros(size(Phi,2),1); %size(z) = N*1
    v = u;
    t = 1;
    numericalprecision = 1e-12;
    T = [];
    while (t <= maxiterations) && (norm(v)/norm(u) > tol)
        y = abs(Phi'*v);
        [vals,z] = sort(y,'descend');
        Omega = find(y >= vals(2*K) & y > numericalprecision);
        T = union(Omega,T);
        b = pinv(Phi(:,T))*u;
        [vals,z] = sort(abs(b),'descend');
        K_indices =(abs(b) >= vals(K) & abs(b) > numericalprecision);
        T = T(K_indices);
        z = zeros(size(Phi,2),1);
        b = b(K_indices);
        z(T) = b;
        v = u - Phi(:,T)*b;
        t = t+1;

```

```
end
n_0=length(T);
index = T;
U_red = Phi(:,T);
```

LASSO

```
U_lasso=[real(X_id_cut) -imag(X_id_cut);imag(X_id_cut) real(X_id_cut)];
y_lasso=[real(y_id_cut);imag(y_id_cut)];
lambdas=logspace(1,6,50)/1e8;
tic
w_lasso=lasso(U_lasso,y_lasso,'Lambda',lambdas);
for aa=1:size(w_lasso,2)
    w_lasso2=w_lasso(1:orderi,aa)+j*w_lasso(orderi+1:end,aa);
    I_lasso=find(w_lasso2);
    w_lasso2=w_lasso2(I_lasso);
    y_est=X_val_tot(:,I_lasso)*w_lasso2;
    NMSE_id_PA(aa)=dpd_Qmeasurements(y_val,y_est,'NMSE');
    N_coef(aa)=length(w_lasso2);
    w_LS=(X_id_cut(:,I_lasso)'*X_id_cut(:,I_lasso))^-1*X_id_cut(:,I_lasso)
        '*y_id_cut;
    y_est_LS=X_val_tot(:,I_lasso)*w_LS;
    NMSE_id_PA_LS(aa)=dpd_Qmeasurements(y_val,y_est_LS,'NMSE');
    ACEPR(aa)=acepr(y_val,y_est_LS,20e6,fs);
end
RT_lasso=toc;
v_plots(n_plots)=scatter(N_coef,NMSE_id_PA_LS,'o');
```

Hill Climbing

```
close all; clear all;
Beh_Models_path
load('data\IN_OUT_Multiband_Signal.mat');
L=length(xBB);
x_id=xBB(1:L/2);  y_id=yBB(1:L/2);  x_val=xBB(L/2+1:end); y_val=yBB(L/2+1:
    end);


%Constantes para calcular la funcion de coste.
NMSE_threshold=-25.7664*0.8; lambda=1.0122*15;

%model selection
PA_model= 'GMP';
 %armamos una matriz donde cada fila significa
matriz_inicio(1,:)=[1 1 1 4 6 6 2 2];


for r=1:size(matriz_inicio,1)%en cada iteracion va a correr el HC empezando
     en diferentes puntos.
    pos_parameters_actual=matriz_inicio(r,:);
    switch PA_model
        case 'MP' %Este primer modelo nos sirve para calcular las
            constantes

        case 'GMP'
            %Caargamos los vectores desde donde se van a generar las
            %diferentes combinaciones de diferentes modelos.
            tic
```

```
Pa=[6 7 8 9];Pb=[2 3 4 5];Pc=[2 3 4 5];
taus_Ma=[0:1:10]; taus_Mb=[-4:1:5]; taus_Mc=[-4:1:5];
taus_Lb = [1:1:4];
taus_Lc = [1:1:4];
%Elegimos los parametros de primer modelo y calculamos sus
%caracteristicas.
Pa_actual=Pa(pos_parameters_actual(1));
Pb_actual=Pb(pos_parameters_actual(2));
Pc_actual=Pc(pos_parameters_actual(3));
taus_Ma_actual=taus_Ma(1:pos_parameters_actual(4));
taus_Mb_actual=taus_Mb(1:pos_parameters_actual(5));
taus_Mc_actual=taus_Mc(1:pos_parameters_actual(6));
taus_Lb_actual=taus_Lb(1:pos_parameters_actual(7));
taus_Lc_actual=taus_Lc(1:pos_parameters_actual(8));
[X_id,N_coef_actual]=Build_Data_matrix_GMP(x_id,Pa_actual,
    Pb_actual,Pc_actual,...
    taus_Ma_actual,taus_Mb_actual,taus_Mc_actual,
        taus_Lb_actual, taus_Lc_actual);
[X_val,order_val]=Build_Data_matrix_GMP(x_val,Pa_actual,
    Pb_actual,Pc_actual,...
    taus_Ma_actual,taus_Mb_actual,taus_Mc_actual,
        taus_Lb_actual, taus_Lc_actual);
w_pa=(X_id'*X_id)^-1*X_id'*y_id;
y_est=X_val*w_pa;
NMSE_actual=dpd_Qmeasurements(y_val,y_est,'NMSE');
%cost_actual=N_coef_actual+lambda*(NMSE_actual-NMSE_threshold);
cost_actual=0.78*N_coef_actual+2.39*NMSE_actual;
encontrado=0;
cont=0; %para saber cuantas veces a buscado nuevos vecinos
acumular_NMSE=[];
acumular_N_Coef=[];
while(encontrado<1)
    fprintf('\n Evaluando grupo de vecinos numero: = %d\n',cont
        +1);
    [Pal,Pbl,Pcl,Mal,Mbl,Mcl,Lbl,Lcl]=Generate_Neighbors_GMP(Pa
        , Pb, Pc, taus_Ma,...
        taus_Mb, taus_Mc, taus_Lb, taus_Lc,
            pos_parameters_actual);
    neighboors=length(Pal);
    %calcular coste de todos los vecinos
    ite=1;
    for i=1:neighboors
        PaN=Pa(Pal(i));
        PbN=Pb(Pbl(i));
        PcN=Pb(Pcl(i));
        MaN=taus_Ma(1:Mal(i));
        MbN=taus_Mb(1:Mbl(i));
        McN=taus_Mc(1:Mcl(i));
        LbN=taus_Lb(1:Lbl(i));
        LcN=taus_Lc(1:Lcl(i));
        [X_id,N_coef(i)]=Build_Data_matrix_GMP(x_id,PaN,PbN,PcN
            ,MaN,MbN,McN,LbN,LcN);
        [X_val,order_val]=Build_Data_matrix_GMP(x_val,PaN,PbN,
            PcN,MaN,MbN,McN,LbN,LcN);
        %data validation
        w_pa=(X_id'*X_id)^-1*X_id'*y_id;
        y_est=X_val*w_pa;
        NMSE_id_PA(i)=dpd_Qmeasurements(y_val,y_est,'NMSE');
```

```
                        ACEPR(i)=acepr(y_val,y_est,20e6,fs);
                        %cost(i)=N_coef(i)+lambda*(NMSE_id_PA(i)-NMSE_threshold
                            );
                        cost(i)=0.78*N_coef(i)+2.39*NMSE_id_PA(i);
                        if (i==100*ite); fprintf('\n Numero de vecinos
                            evaluados = %d de %d \n',i...
                                ,neighboors);
                            ite=ite+1;
                        end
                    end
                    %comprobar si hay alguno vecino mejor que el actual
                    [M,I]=min(cost);
                    if(M<cost_actual)
                        cost_actual=M;
                        pos_parameters_actual=[Pal(I),Pbl(I),Pcl(I),Mal(I),Mbl(
                            I),Mcl(I),Lbl(I),Lcl(I)];
                        NMSE_actual=NMSE_id_PA(I);
                        N_coef_actual= N_coef(I);
                        cont=cont+1;
                    else
                        encontrado=2;
                        fprintf('\n Modelo Encontrado\n');
                    end
                    acumular_NMSE=[acumular_NMSE NMSE_id_PA ];
                    acumular_N_coef=[acumular_N_Coef N_coef ];
                    acumular_ACEPR=[acumular_ACEPR ACEPR];
                end
                RT_HC=toc;
            case 'VOLTERRA'


        end
        COST_TOTAL(r,:)=cost;
        NMSE_TOTAL(r,:)=NMSE_id_PA;
        N_COEF_TOTAL(r,:)=N_coef;
        CONT_TOTAL(r)=cont;
        POS_PARAMETER_TOTAL(r,:)=pos_parameters_actual;
        INDEX_TOTAL(r)=I;
        N_coef=acumular_N_coef;
        NMSE_id_PA=acumular_NMSE;
        ACEPR=acumular_ACEPR;
        figure(r)
        [pareto_x,pareto_y]=pareto(N_coef,NMSE_id_PA);
        [ACEPR_x,ACEPR_y]=pareto(N_coef,ACEPR);
        figure(r+1)
        plot(pareto_x,pareto_y,'ok','LineWidth',1.5);
        xlabel('NUMERO DE COEFICIENTES');
        ylabel('NMSE [dB]');
        title('NMSE vs NUMERO DE COEFICIENTES CON HILL CLIMBING');
        fprintf('\n Configuracion numero  %d \n',r);
    end
    save('data\HC\Hill_climbing_GMP_11146622','cost','NMSE_id_PA','N_Coef','
        pareto_x','pareto_y',...
        'RT_HC','ACEPR_x','ACEPR_y');
```

Dynamic Model Sizing

```
        function [P,V,I,NMSE,N_coef,ACEPR]=DMS(x_id,x_val,X_id,X_val,y_id,y_val
```

```
    ,params,N_grow,N_grow2,N_nu)
load('data\IN_OUT_Multiband_Signal.mat');

Target_NMSE=-33;
P=X_id;
V=X_val;
NMSE_anterior=0;
params1=params;
I=(1:1:size(params,2));
%N_grow=120;
%N_nu=3;
a=0;
cont=1;
while(a==0)
    N_prune=N_grow-N_nu;;
    NMSE_anterior=0;
    for t=1:N_prune
        %w_pa=(P'*P)^-1*P'*y_id;%% al elevar a -1 da infinito
        w_pa=P\y_id;
        nu = diag((P'*P)^-1);
        z_score=abs(w_pa)./nu.^(0.5);
        [z_score,I]=sort(z_score,'descend');
        I_temp=I;
        lowest_z=I(end);
        P(:,lowest_z)=[];
        V(:,lowest_z)=[];
        params1(:,lowest_z)=[];
        P_temp=P;
        %w_pa=(P'*P)^-1*P'*y_id;
        w_pa=P\y_id;
        w_pa_temp=w_pa;
        y_est=V*w_pa;
        NMSE_actual=dpd_Qmeasurements(y_val,y_est,'NMSE');
        ACEPR_actual=acepr(y_val,y_est,20e6,fs);
        maxDegr=NMSE_actual-Target_NMSE;
        Degr=NMSE_actual-NMSE_anterior;
        NMSE_anterior=NMSE_actual;
        if(Degr>maxDegr)
            P=P_temp;
            w_pa=w_pa_temp;
            break;
        end
    end
    NMSE(cont)=NMSE_actual;
    ACEPR(cont)=ACEPR_actual;
    N_coef(cont)=length(w_pa);
    cont=cont+1;
    if(length(w_pa)>139)
        nu = diag((P'*P)^-1);
        z_score=abs(w_pa)./nu.^(0.5);
        [z_score,I]=sort(z_score,'descend');
        params1=params1(:,I);
        P=P(:,I);
        V=V(:,I);

        break;
    end
    nu = diag((P'*P)^-1);
```

```
59            z_score=abs(w_pa)./nu.^(0.5);
60            [z_score,I]=sort(z_score,'descend');
61            params1=params1(:,I);
62            P=P(:,I);
63            V=V(:,I);
64
65            all_Neigbors=[];
66            for x=1:size(params1,2)
67                flag=params1(1,x);
68                M=params1(2,x);
69                L=params1(3,x);
70                n=params1(4,x);
71                [Neigbors]=Generate_DMS_Neighbors(flag,M,L,n);
72                all_Neigbors=[all_Neigbors Neigbors];
73            end
74            all_Neigbors=(unique(all_Neigbors','rows','stable'))';
75            all_Neigbors=all_Neigbors';
76            params1=params1';
77            [Lia,eoe]=ismember(all_Neigbors,params1,'rows');
78            final_Neigboors=all_Neigbors.*~Lia;
79            newParams = final_Neigboors(any(final_Neigboors,2),:)';
80            N_grow=N_grow2;
81            for y=1:N_grow
82                newP(:,y)=Build_Data_vector_GMP(x_id, newParams(:,y));
83                newV(:,y)=Build_Data_vector_GMP(x_val, newParams(:,y));
84            end
85            P=[P newP];
86            V=[V newV];
87            params1=[params1' newParams(:,1:N_grow)];
88
89        end
90        end
```