Faculdade de Engenharia da Universidade do Porto

Autoteste e Correcção de Não-linearidades de Circuitos RF

Joana Azevedo Braga

Dissertação
Mestrado Integrado em Engenharia Electrotécnica e de Computadores
Major Telecomunicações

Orientador: Prof. José Machado da Silva
Co-Orientador: Eng.º Pedro Mota

29 de Julho de 2010

Resumo

Este documento descreve o trabalho de estudo e desenvolvimento de uma metodologia de estimação de parâmetros de caracterização de não linearidades e defeitos de funcionamento, assim como sua correcção, de circuitos de radiofrequência (RF). Em trabalhos anteriores tem-se considerado o caso de amplificadores de potência RF.

Começou-se por estudar e apresentar os parâmetros de caracterização das não linearidades – ponto de compressão de 1dB e ponto de intersecção de 3ª ordem - e suas consequências no desempenho dos circuitos, tais como, defeitos de funcionamento vários, espalhamento espectral e interferências.

De seguida, fez-se uma revisão do estado da arte de medições em amplificadores de potência de RF. Apresentam-se as consequências da distorção não-linear em modulações MC-CMDA e em particular, nos seus circuitos de transmissão que incluem amplificadores de potência de RF. Para se poder analisar o comportamento de um dado amplificador temos de conseguir obter a sua curva característica. Descreve-se um método já testado e validado para o fazer que é baseado na correlação cruzada entre a tensão e a corrente. Após serem adquiridas as amostras de tensão ou potência necessárias para estimar a curva característica é necessário usar um método de interpolação para estimar todos os pontos da curva que são diferentes da amostra. Por isso, decidiu-se estudar os diferentes métodos de aproximação polinomial existentes. O método de interpolação por *spline* cúbica revela ser o mais preciso para o cálculo da dos coeficientes do polinómio da característica de transferência do circuito em análise. Resultados obtidos por simulação do cálculo do ponto de compressão de 1dB mostram que erros pequenos podem ser obtidos em comparação com aqueles obtidos usando o método dos mínimos quadrados.

Uma vez que a complexidade do algoritmo matricial de cálculo de *splines* cúbicas não permite a sua implementação em FPGA com um número reduzido de componentes físicos, foi

investigada a B-spline e a sua implementação com recurso a filtros digitais como forma de contornar esta dificuldade.

Tirando partido do cálculo da aproximação polinomial em circuito foi ainda desenvolvido um algoritmo de cálculo dos pontos de compressão 1dB e de intersecção de 3ª ordem que permite ajustar o comportamento do amplificador a partir da avaliação dos coeficientes do polinómio da sua curva característica.

Descreve-se uma possível implementação em Hardware usando uma descrição em System Verilog.

Finalmente, apresentam-se as conclusões deste trabalho.

Abstract

This document describes the study and development work for a non-linearity characterization parameters estimation methodology. This methodology also detects functionality defects and allows their correction. It applies to radiofrequency (RF) circuits. In previous works the RF power amplifier has been considered.

First, a study of the non linearties characterization parameters – 1dB compression point and third order interception point – and their consequences in circuit performance, such as, several functional defects, spread spectrum and interferences.

Next, a revision from the RF power amplifiers measurement state of the art was made. The consequences from the non-linear distortion in MC-CDMA modulation and, in particular, in their transmission circuits that include RF power amplifiers are presented. In order to analyze the behavior from a particular amplifier we must be able to get its characteristic curve. A method to obtain the amplifier characteristic curve that has been tested and validated is presented. This method is based on the cross-correlation between voltage and current. After the power and voltage samples needed to estimate the characteristic curve had been acquired it is necessary to use one interpolation method to obtain all the points in the curve that are different from the samples. For that reason, it was decided to study the different interpolation methods available. The cubic spline interpolation method, disclose to be the most precise in the circuit transfer characteristic polynomial coefficients calculus. 1dB compression point calculus simulation results show that very small errors can be obtained using the cubic spline method when compared to the ones that are found using the minimum squares method.

Since the cubic spline matrix form calculus complexity does not allow its implementation in FPGA using a small amount of physical components, the B-spline and its implementation using digital filters, as a way around this difficulty, was investigated.

Taking advantage from the in-circuit polynomial approximation calculus it was developed a 1dB compression point and third order interception point calculus algorithm that allows adjusting the amplifier behavior taking into account the polynomial coefficients from its characteristic curve.

A System Verilog possible implementation is described.

Finally the work conclusions are presented.

Índice

Resumo iii
Abstract v
Índice vii
Lista de Figuras x
Lista de tabelas xi
Abreviaturas e Símbolos xii

Capítulo 1 15

Introdução 15
Distorcer - alterar a forma ou o padrão característico de algo. (dicionário Houaiss da Língua portuguesa) 15
1.1 Distorções não lineares 16
1.2 O requisito de linearidade 16
1.3 Modulação linear 17
1.4 Adaptação de polinómios 17
1.5 Motivação para a realização do Trabalho 18
1.6 Estrutura do documento. 18

Capítulo 2 21

Estado da Arte 21
2.1.1 Distorção de Amplitude 21
2.1.1.1 Característica "Square-Law" 21
2.1.1.2 Característica de terceira ordem 23
2.1.2 Ponto de compressão 1dB 23
2.1.3 Ponto de intercepção de terceira ordem 25

2.2 Medições em amplificadores de potência 27
2.2.1 Teste "Two – Tone" 27
2.2.4 Técnicas de Linearização de Amplificadores de Potência 28
2.2.5 Método de medida de não-linearidade em amplificador de potência integrado em tecnologia CMOS 0,35um 30
2.2.6 Estimar não linearidades de amplificadores de potência RF após correlação cruzada entre a tensão e a corrente de saída 32

2.4 Métodos de aproximação polinomial 34
2.4.1 Introdução 34
2.4.2 Método de Lagrange 35
2.4.3 Comparação de dois métodos de Interpolação polinomial 35

2.5 Sumário 40

Capítulo 3 41

Aproximação Polinomial por Meio de *Splines* 41

3.1 Interpolação de Spline – Polinómios de Spline 42

3.2 Spline Cúbico 42
3.2.1 Cálculo matricial de Spline Cúbico 42
3.2.2 Implementação do Algoritmo matricial de cálculo do Spline Cúbico 43
3.2.3 Interpolação de Spline – Polinómios de Spline – B-spline 45

3.2.4 Interpolação de B-Spline via filtro digital 46
3.2.5 Interpolação rápida por spline cúbica 48
3.3 Algoritmo para cálculo do ponto de compressão 1dB 49
3.3.1 Resultados 50
3.4 Implementação em MATLAB de Spline cúbica com recurso a filtros digitais 56
3.4.1 Regra de Horner 56
3.4.2 Aplicação da regra de Horner na resolução da B - spline cúbica. 61
3.6 Algoritmo para cálculo de P1dB e IP3 a partir de uma característica de transferência conhecida 63

Capítulo 4 65

4.1 Introdução 65
4.2 Módulos Verilog 66
4.2.1 Módulo Verilog "conversão_escalas.v" 67
4.2.2 Módulo de Verilog "polinómio2.v" 67
4.2.3 Módulo de System Verilog "filtros_para_coeficientes1.sv" 67
4.2.4 Módulo "cubic_spline36.v" 67
4.3 Resultados da Implementação em Verilog 68

Capítulo 5 71

Conclusão 71
5.1 Trabalhos Futuros 73

ANEXO A – Listagens de MATLAB 75

A.1 P1dB_Spline 76
A.2 zona_linear 78
A.3 calculo_P1dB_IP3_ganho 78
A.4 horner 80
A.5 spline_cubico 80
A.6 calculo_P1dB_IP3_ganho para FPGA 82
A.7 P1dB_Spline 87
A.8 P1dB_Spline 88
A.9 P1dB_Spline5 89
A.10 Spline Matricial – listagem de "csfit" retirada do livro [16] 91
A.11 Cálculo de P1dB com recurso a spline matricial 91

Anexo B – Listagens de Verilog 95

B.1 Conversão de escalas 96
B.2 Filtro Digital baseado na regra de Horner 97
B.3 Polinómio de Spline cúbica – polinomio2.v 100
B.4 Módulo de teste do "polinómio2.v" 100
B.5 Módulo de Filtros para coeficientes em System Verilog 101
B.6 Módulo que implementa a spline cúbica para 36 amostras 118
B.7 Módulo de teste da Spline cúbica 119
B.8 Módulo "Factores de escala.v" 119

B.9 Ficheiro de dados de entrada “alfa.dat” 121
B.10 Ficheiro de dados de entrada “beta.dat” 121
B.11 Ficheiro de dados X.dat 122
B.12 Ficheiro de dados Y.dat 123
B.13 Ficheiro de dados C0_menos.dat 124
B.14 Ficheiro de dados P3.dat 124
B.15 Script dos resultados do teste de spline cúbico 125
Referências 127

Lista de Figuras

FIGURA 2-1 PONTO DE COMPRESSÃO 1DB DE UM AMPLIFICADOR COM CARACTERÍSTICA DADA POR VOUT 23
FIGURA 2-2 CARACTERÍSTICA DE TRANSFERÊNCIA (A) 25
FIGURA 2-4 CURVA SINUSOIDAL DE 1V E 1HZ NA ENTRADA (B) 25
FIGURA 2-3 CURVA OBTIDA NA SAÍDA DO AMPLIFICADOR 25
FIGURA 2-5 FIGURA DO PONTO DE INTERSECÇÃO DE TERCEIRA ORDEM, IP3, DE UM AMPLIFICADOR NÃO LINEAR 26
FIGURA 2-6 ESPECTRO DA RESPOSTA NO DOMÍNIO DAS FREQUÊNCIAS DE UM AMPLIFICADOR NÃO LINEAR AO QUAL SE APLICOU O TESTE DE DOIS SINAIS DE DUAS FREQUÊNCIAS DIFERENTES NA ENTRADA 27
FIGURA 2-7 REALIMENTAÇÃO NEGATIVA NUM AMPLIFICADOR DE RF 29
FIGURA 2-8 APROXIMAÇÃO POLINOMIAL DE LAGRANGE 36
FIGURA 2-9 POLINÓMIO DE LAGRANGE DE 3ª ORDEM VERSUS POLINÓMIO DE 3ª ORDEM ORIGINAL 37
FIGURA 2-10 SPLINE CÚBICA 38
FIGURA 3-1 PONTOS DE CALCULO DE SPLINE CÚBICO V CURVA OBTIDA POR MÉTODO DOS MÍNIMOS QUADRADOS 44
FIGURA 3-2 - CURVA CARACTERÍSTICA VI/VO DESENHADA COM BASE EM 7 AMOSTRAS E COM O USO DO MÉTODO DE INTERPOLAÇÃO DE SPLINE CÚBICO MATRICIAL 45
FIGURA 4-1 SISTEMA IMPLEMENTADO 66

Lista de tabelas

TABELA 0-1 VALORES DE P1DB ESTIMADOS COM INTERPOLAÇÃO DE SPLINE CÚBICO MATRICIAL E 36 AMOSTRAS 54

TABELA 0-2 VALORES DE P1DB ESTIMADOS COM INTERPOLAÇÃO DE SPLINE CÚBICO MATRICIAL E 6 AMOSTRAS..... 55

TABELA 0-3 RESULTADOS DO CÁLCULO DO PONTO DE COMPRESSÃO 1DB COM RECURSO A INTERPOLAÇÃO POR SPLINE CÚBICA BASEADA EM FILTROS DIGITAIS. 61

Abreviaturas e Símbolos

Lista de abreviaturas (ordenadas por ordem alfabética)

ACPR	*Adjacent channel power ratio*
ACSSB	*Amplitude companed single-sideband*
ADC	*Analog to Digital Converter*
AM	*Amplitude Modulation*
ASIC	*Application-Specific Integrated Circuit*
CDMA	*Code Division Multiple Access*
CW	*Carrier Wave*
CSA	Coeficiente de Sobre-Amostragem
DC	Corrente contínua
DFT	*Discrete Fourier Transform*
DS-CDMA	*Direct-Sequence Code-Division-Multiple-Access*
FM	*Frequency Modulation*
FPGA	*Field Programmable Gate Array*
HDL	*Hardware Description Languages*
IMD	*Intermodulation Distortion*
MC-CDMA	*Multi-Carrier Code-Division-Multiple-Access*
M-IMR	*Multitone Intermodulation Ratio*
NPR	*Noise power ratio*
PA	*Power Amplifier*
PCB	*Printed circuit board*
PEP	*Peak envelope power*
QAM	*Quadrature amplitude modulation*
QPSK	*Quadrature shift keying*
OFDM	*Orthogonal-Frequency-Division-Multiplexing*
RF	*Radio Frequency*
RTL	*Register Tranfer Logic*
SSB	*Single Side Band*
TTIB	*Transparent tone in band*

Lista de símbolos

ω	Frequência angular
α	Ângulo

Capítulo 1

Introdução

Distorcer *- alterar a forma ou o padrão característico de algo. (dicionário Houaiss da Língua portuguesa)*

Em condições ideais o sinal recebido à saída de uma ligação ponto a ponto é uma cópia exacta do sinal enviado. Nos canais reais, devido, por exemplo, à limitada largura de banda do canal, aos atrasos na transmissão ou ao aparecimento de sinais parasitas (ruído), o sinal recebido é na verdade uma aproximação do sinal enviado.

As distorções consistem em alterações do sinal devido a imperfeições do sistema. Podem ser reduzidas através de um projecto adequado ou de componentes adicionais de compensação.

Um canal de transmissão pode ser visto como um filtro. Considera-se que não introduz distorção quando, aplicando um sinal x(t) qualquer à sua entrada, a saída y(t) pode ser definida da seguinte forma: $y(t) = K\,x(t - t_d)$. Isto é, admitimos que a amplitude de x(t) vem afectada por uma constante K e que x(t) sofre um atraso. Esta situação corresponde a considerar que o canal é um sistema linear e invariante no tempo com a seguinte resposta em frequência: $H(f) = Ke^{-j2\pi ftd}$. Um canal linear não introduz distorção se: $|H(f)| = Ke^{argH(f)=-2\pi td\pm m\pi}$.

Considera-se que há distorção de amplitude quando a amplitude de $H(f)$ não é constante – as amplitudes das componentes do sinal às diversas frequências são multiplicadas por constantes diferentes. Por outro lado, há distorção de fase quando a fase de $H(f)$ não é linear ou não passa pela origem. Neste caso, as componentes do sinal às diversas frequências são afectadas por atrasos diferentes: $t_d(f) = -\frac{argH(f)}{2\pi f}$ [7].

1.1 Distorções não lineares

Os canais (ou filtros) não lineares não podem ser representados por uma resposta em frequência ou por uma resposta impulsional. Neste caso podemos relacionar a entrada com a saída através de funções do tipo $y(t) = F[x(t)]$. Utilizando uma aproximação polinomial para a função F, temos $y(t) = a_1x(t) + a_2x(t)^2 + a_3x(t)^3 + \ldots$ ou no domínio das frequências $Y(f) = a_1X(f) + a_2X(f) * X(f) + a_3X(f) * X(f) * X(f) + \ldots$

A compensação das distorções não lineares é em muitos casos, impossível. No caso de canais (ou filtros) não lineares, o sinal de saída pode ser mais rico harmonicamente do que o sinal de entrada [7].

Todos os amplificadores possuem esta característica de distorcer os sinais que amplificam. A ocorrência de distorção e em consequência não linearidades em amplificadores áudio é muito desagradável para o ouvido. Amplificadores de alta-fidelidade foram sendo projectados e refinados ao longo dos anos de modo a reduzir este fenómeno para níveis considerados inaudíveis pelo ouvido humano. Com a descoberta da correcção por realimentação H.S.Black permitiu que se alcançasse este objectivo com relativa facilidade [1].

Quando consideramos amplificadores de radiofrequência, a eficiência espectral, as interferências e a necessidade de respeitar os outros utilizadores do espectro são factores que se tornam essenciais, incluindo considerações de erro do próprio sinal [1].

1.2 O requisito de linearidade

Todos os sistemas rádio têm necessidade de reduzir a um mínimo as interferências que provocam nos outros utilizadores do espectro. Têm por isso de manter a sua transmissão dentro da largura de banda que lhes foi atribuída pela autoridade nacional para as comunicações e não podem irradiar energia significativa fora da sua banda. As não linearidades intrínsecas dos equipamentos rádio provocam distorção do sinal transmitido e resultam na geração de sinais fora do canal de frequência ou banda desejado. Estes produtos indesejados da distorção são potencialmente fontes de interferência para os outros utilizadores do espectro e têm de ser reduzidos a um nível que permita que todos possam funcionar de um modo satisfatório.

No caso de um transmissor de alta potência este requisito torna-se mais significativo; pois os produtos da distorção embora, muitas vezes, menores que o sinal principal de saída, podem ser bastantes grandes em termos absolutos e daí causarem interferência.

Felizmente, os sistemas de rádio podem aplicar filtragem com sucesso de modo a reduzir as componentes harmónicas da distorção para um nível aceitável.

Assim, transmissores irradiando um único sinal de portadora com modulação de um sinal constante em amplitude ao longo do tempo (como por exemplo uma estação emissora de FM)

podem ser limitados a irradiar apenas dentro da sua banda por meio exclusivo de filtragem. Transmissores de sinais que variam a sua amplitude ao longo do tempo (incluindo AM, SSB e outros) podem ainda usar filtros para eliminar harmónicos, mas irão também produzir produtos de distorção de intermodulação (IMD – *intermodulation distortion*) próximos do canal desejado e estes não podem ser eliminados facilmente por filtragem [1].

1.3 Modulação linear

As modulações lineares podem ser definidas como sendo aquelas nas quais a informação é transmitida quer na amplitude quer na fase do sinal RF (radiofrequência). A amplitude do sinal RF no tempo varia e, por isso, tem que ser preservada de modo a guardar toda a informação contida no sinal original da mensagem. Um interesse recente nas modulações lineares tem sido alimentado pela necessidade de uma utilização mais eficiente do espectro, tanto para transmissão de voz analógica (e. g., em conversações telefónicas) como para transmissão de dados.

A modulação "*Single-sideband*" é uma escolha popular para transmissão analógica de voz e um número de avançadas derivações técnicas tem sido usado, tais como:
ACSSB – "amplitude companded single sideband", e TTIB – "*transparent tone in band*". Muitas modulações para transmissão de dados requerem que ambos, a fase e a amplitude do sinal RF sejam preservados. Um interesse recente tem estado centrado à volta das modulações "*quadrature amplitude modulation*" (QAM) and "*quadrature shift keying*" (QPSK) com um interesse específico em 16-QAM e $\pi/4$ – shift QPSK para sistemas de rádio móvel.

A maior desvantagem que evitou a adopção generalizada de muitas das técnicas de modulação linear (i.e. SSB) foi o requisito de um amplificador linear no transmissor. A forma tradicional de amplificação linear em RF tem sido o projecto de circuitos de classe A ou classe AB e estes são ambos pouco eficientes e não particularmente lineares. São, assim, apenas adequados para certas formas de comunicação ponto a ponto e estão longe do ideal para serem usados num ambiente de rádio móvel. Considerações sobre a duração de baterias restringem o uso de tais amplificadores de um ponto de vista de eficiência, e os efeitos de próximo-afastado (*near-far effects*) podem resultar em níveis inaceitáveis de interferência para os outros utilizadores, devido aos seus níveis inerentes de distorção.

O uso de uma modulação verdadeiramente linear num sistema de rádio móvel requer um amplificador de grande linearidade e, desse modo, o uso de alguma técnica de linearização [1].

1.4 Adaptação de polinómios

Novos desenvolvimentos na adaptação de polinómios como meio de estimar as não linearidades dos circuitos de rádio frequência foram apresentados com o objectivo de

aumentar a precisão da estimativa em particular na presença de fortes não linearidades. Resultados de simulação obtidos com amplificador classe E e F ilustram tais conclusões. A aproximação polinomial tem sido também explorada na implementação de algoritmos de correcção e estimação de não linearidades. A proposta apresentada em [4] fornece três modos distintos de operação: teste, ajuste da potência constante na saída e correcção de não linearidades. Um protótipo em hardware foi construído e validado [6].

A medição em-circuito da potência de sinais RF é habitualmente feita recorrendo a detectores de pico (ou RMS) de tensão sendo a potência inferida assumindo que a carga é conhecida, o que de facto pode não acontecer. Como forma de se obter uma medição mais exacta da potência é proposto em [8] o recurso à correlação cruzada entre a tensão e a corrente na carga, operação que permite obter uma medição directa da potência activa na carga.

1.5 Motivação para a realização do Trabalho

O objectivo deste trabalho é transformar uma medição tradicional em laboratório e com equipamento complexo, analisadores de espectro e osciloscópios, como por exemplo o teste de dois tons, numa medição *on-chip* e com o circuito em funcionamento.

Existem outras medições de circuitos RF como, por exemplo, cálculo da distorção de intermodulação, cálculo da potência adjacente ao canal, cálculo da potência de ruído e teste de vários tons (este inclui o teste de dois tons). Podemos encontrar na obra *"High-Linearity RF Amplifier Design."* descrições detalhadas de todas estas medições *[1]*.

Já foram realizados diversos trabalhos nesta área como, por exemplo, um método de medida de não linearidade em amplificador de potência integrado em tecnologia CMOS 0.35u [4], [6]. Deseja-se com este novo trabalho obter um método mais simples e expedito para realizar os cálculos necessários para estimar as não linearidades de um amplificador de potência ou de um outro qualquer circuito de RF. Este novo desenvolvimento poderá depois ser usado para actuar no dispositivo RF. Esta actuação poderá ser através da implementação de técnicas de linearização, em particular, no caso de um amplificador de potência a técnica de pré-distorção. Uma descrição pormenorizada das técnicas de linearização é feita no livro *"High-Linearity RF Amplifier Design." [1]*.

O interesse em ter um amplificador linear é notório em algumas modulações como, por exemplo, em MC-CDMA e DS-CDMA em que o desempenho de ambos os sistemas pode ser semelhante com o uso de técnicas de pré-distorção nos amplificadores de potência [18].

1.6 Estrutura do documento.

No primeiro capítulo é feita uma introdução ao problema da distorção e da distorção não-linear, fala-se do requisito de linearidade, da modulação linear e da adaptação de polinómios

como meio de estimar não-linearidades. A motivação para a realização deste trabalho é apresentada assim como a estrutura deste documento.

No segundo capítulo é apresentado o estado da arte na caracterização da distorção não-linear. São apresentados os pontos de compressão 1dB e de intersecção de 3ª ordem. O método de medição de duas frequências ou teste de "two-tone" é descrito, fala-se ainda de técnicas de linearização. Na secção 2.2.5 descreve-se um método de medida de não linearidade *on-chip*. De seguida, apresenta-se o método de correlação cruzada entre a corrente e a tensão como um método para medir a potência de uma forma directa. Termina o capítulo, com uma descrição de diferentes métodos de interpolação polinomial.

No capítulo 3 descreve-se a interpolação polinomial de Spline cúbico na sua forma matricial e na sua forma discreta B-Spline. São apresentados algoritmos para cálculo do ponto de compressão de 1dB e do ponto de intersecção de terceira ordem. Faz-se ainda uma comparação de resultados obtidos no cálculo do ponto de compressão 1dB através do uso de um algoritmo que o calcula baseado em interpolação polinomial, mas fazendo variar o método de interpolação que é usado. Finaliza o capítulo uma descrição de um algoritmo adaptado para FPGA que implementa o cálculo dos pontos P1dB e IP3 baseado na interpolação polinomial de B-Spline e na regra de Horner.

No capítulo 4 apresenta-se uma possível implementação em Hardware, descrita em liguagem Verilog, para o algoritmo adaptado para FPGA descrito no capítulo 3.

No Capítulo 5 apresentam-se as conclusões deste trabalho.

São ainda incluídas em anexo as listagens dos programas MATLAB usados.

Por fim, as referências usadas.

Capítulo 2

Estado da Arte

2.1 Introdução

Para se poder discutir a linearidade de amplificadores e métodos para assegurar que a linearidade é mantida, é necessário examinar a natureza da distorção nos amplificadores em todas as suas formas e estabelecer técnicas para determinar os seus níveis de forma rápida e precisa.

A distorção em amplificadores de áudio tem sido uma preocupação desde há muitos anos e um esforço considerável de projecto resultou praticamente na sua eliminação nos modernos amplificadores de alta-fidelidade. As técnicas de realimentação usadas convencionalmente às frequências de áudio são, no entanto, inaplicáveis na sua generalidade a amplificadores de radiofrequência devido a problemas de estabilidade com grandes larguras de banda e ao custo de ganhos elevados em andares de radiofrequência. Resultando daí que muitos projectos de amplificadores de RF têm de analisar o compromisso de linearidade versus eficiência. Apresentam-se de seguida diferentes formas de distorção em amplificadores RF em conjunto com alguns métodos de medida e caracterização.

2.1.1 Distorção de Amplitude

2.1.1.1 Característica “Square-Law”

A forma mais simples de não-linearidade de amplitude pode ser ilustrada pela soma de um segundo termo `a característica de transferência do amplificador: um termo proporcional ao quadrado da tensão de entrada.

$$V_{out}(t) = K_1 V_{in}(t) + K_2 V_{in}(t)^2 \quad (1.1)$$

Esta forma de característica de transferência é referida como sendo de segunda ordem devido à potência de dois que foi introduzida.

Quanto maior for o coeficiente do termo de segunda ordem K_2, mais pronunciada é a curva da característica de transferência e daí maior é a distorção da forma de onda de entrada. Note-se que no domínio das frequências uma segunda componente do sinal surgiu ao dobro da frequência original ($2.f_1$) e isto dá origem ao termo de "distorção do segundo harmónico" ("*second harmonic distortion*") usado para descrever a forma de distorção não-linear introduzida pelo segundo termo. Note-se ainda que um termo DC também resulta do termo de segunda ordem na característica de transferência.

Um exame à amplitude da componente harmónica de segunda ordem indica que esta aumenta em proporção com o quadrado do sinal de entrada (e também em proporção com a constante, K_2). A amplitude da componente de frequência fundamental, no entanto, apenas aumenta em proporção com o ganho fundamental K_1. Como resultado, é evidente que a amplitude do segundo harmónico aumenta a uma taxa superior à da amplitude da componente fundamental. Podemos, pois, prever um ponto no qual as componentes fundamental e de segunda ordem têm o mesmo nível: o nível de sinal para o qual, isto ocorre é denominado "ponto de intersecção de segunda ordem" ("*second-order intercept point*"), usualmente expresso como uma potência em dBm. Este pode ser apresentado quer como um ponto de intersecção na entrada quer como um ponto de intersecção na saída; é mais usual o primeiro modo em especificações de "*front-end*" e o último é a forma mais usual para amplificadores de média e grande potência.

A vantagem de se usar um ponto de intersecção para indicar o desempenho linear de um amplificador é que é uma quantidade fixa a partir da qual o nível de distorção de um ponto particular de operação pode ser previsto. A percentagem de distorção harmónica que é geralmente especificada num amplificador áudio tem de ser referida a um nível particular de potência na saída (normalmente o máximo valor especificado para o amplificador) e não dá nenhuma informação sobre o desempenho do amplificador abaixo desse nível. Um compromisso frequentemente adoptado nos amplificadores RF é o de os usar abaixo do seu valor máximo de potência, de modo a assegurar uma baixa distorção. Seria impossível prever qual o valor de potência a usar abaixo do valor máximo para um dado valor de percentagem de distorção medida se não estivesse tabelado ou apresentado graficamente.

Finalmente, note-se que uma característica de segunda ordem produz distorção harmónica, como já foi referido, mas não produz distorção de intermodulação ("*in-band intermodulation distortion*"). Isto é uma distinção importante, em geral, entre não linearidades de ordem par e de ordem ímpar: não linearidades de ordem par não geram distorção de intermodulação.

2.1.1.2 Característica de terceira ordem

Um conjunto de problemas muito diferentes ocorre num amplificador que tem um termo de terceira ordem na sua característica de transferência.

$$V_{out}(t)v = K_1 V_{in}(t) + K_3 V_{in}(t)^3 \quad (1.2)$$

Esta característica é mostrada para K_1 = 10 e K_3 = -3 na Figura número 2-1. Note-se que a forma de onda da saída não é simétrica acima e abaixo do eixo horizontal e que um termo a três vezes a frequência original do sinal de entrada apareceu no espectro. Este sinal é o terceiro harmónico e dá origem à sua descrição como "distorção de terceiro harmónico" ("*third-order harmonic distortion*"). Note-se ainda que não existe componente contínua DC para a distorção de terceira ordem ao contrário do que acontece com a distorção de segunda ordem.

2.1.2 Ponto de compressão 1dB

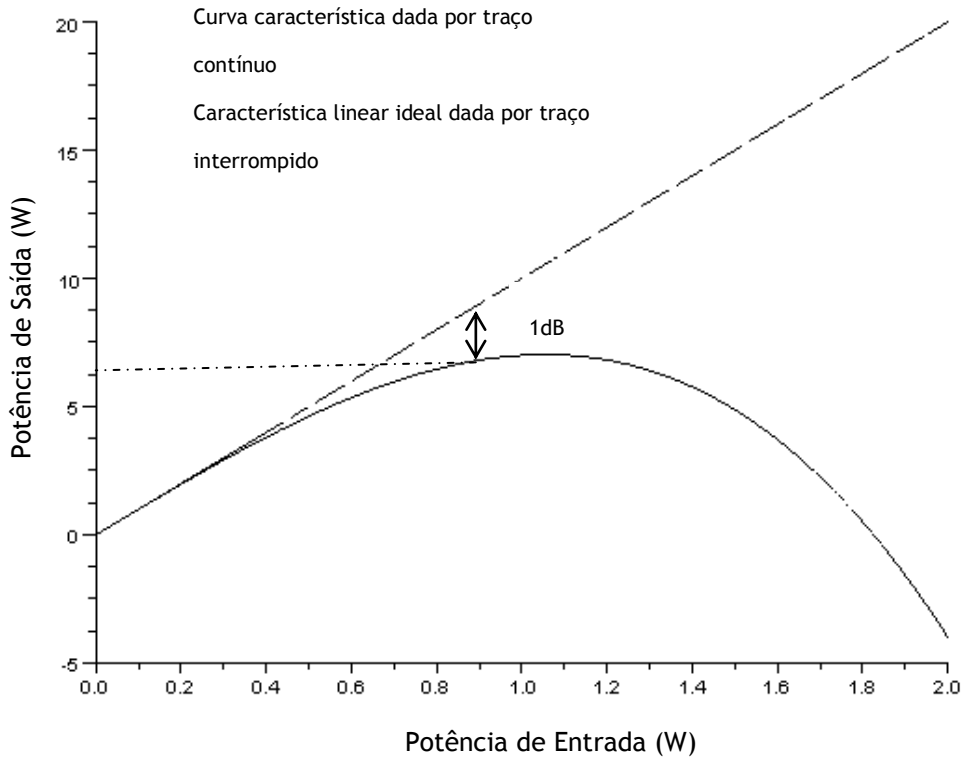


Figura 2-1 Ponto de compressão 1dB de um amplificador com característica dada por Vout

O ponto de compressão 1 dB de um amplificador refere-se ao nível de potência no qual a curva de transferência característica de um amplificador se desvia da característica linear de um amplificador ideal no total de 1dB.

Um outro problema surge quando se consideram duas portadoras moduladas em amplitude como sinais de entrada, em vez do caso simples de dois sinais não modulados como os casos

analisados anteriormente. A quantidade de compressão de um dado sinal depende do valor instantâneo do outro sinal que está a ser amplificado. É assim, possível, que a modulação de amplitude de uma portadora se transfira para a outra portadora e vice-versa. Este problema é conhecido como "*cross-modulation*" e poderá representar um grande problema nos receptores AM quando confrontados com sinais muito potentes, assim como em transmissores que operam próximo da saturação.

Embora o ganho de compressão seja um problema para amplificadores com não-linearidades de terceira ordem, de maior preocupação são os produtos de intermodulação que surgem a frequências $2f_2 - f_1$ e $2f_1 - f_2$. Estes produtos da distorção surgem na banda do sinal de interesse e distorcem a forma de onda, desejada, do sinal de entrada. Mais, uma vez que estes produtos aparecem dentro da banda de interesse, é usualmente, impossível filtrar, ao contrário dos produtos harmónicos a $3f_1$ e $3f_2$. Por este motivo técnicas avançadas de linearização de amplificadores são necessárias de forma a assegurar a sua eliminação.

2.1.3 Ponto de intercepção de terceira ordem

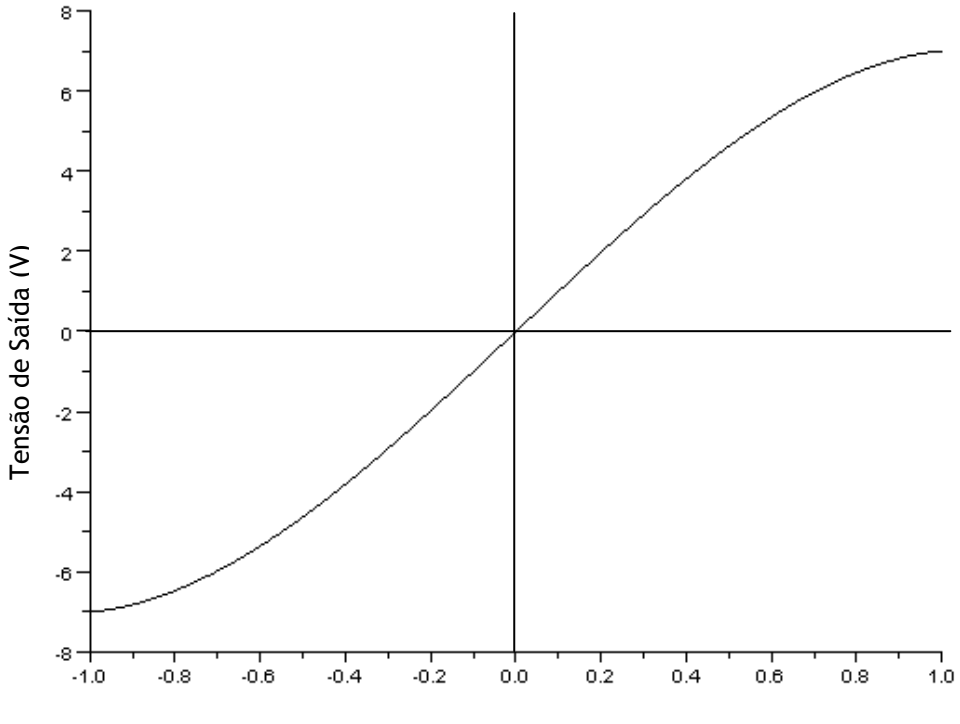


Figura 2-2 Característica de Transferência (a)

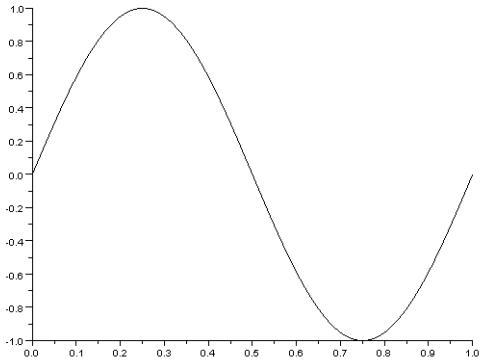


Figura 2-4 curva sinusoidal de 1v e 1Hz na entrada (b)

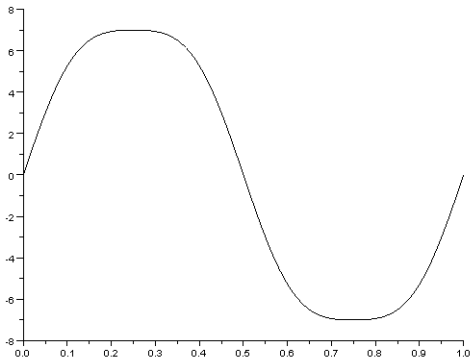
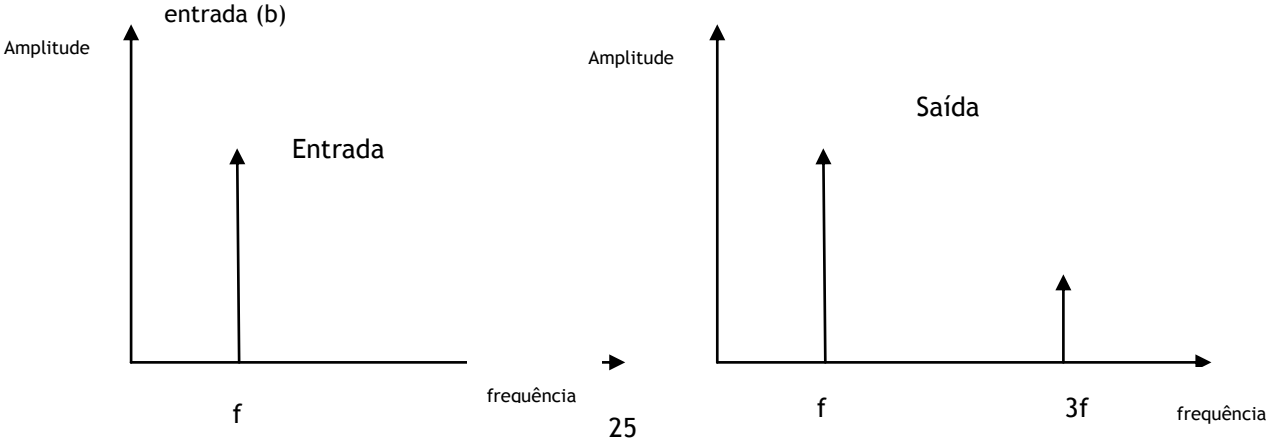


Figura 2-3 Curva obtida na saída do amplificador



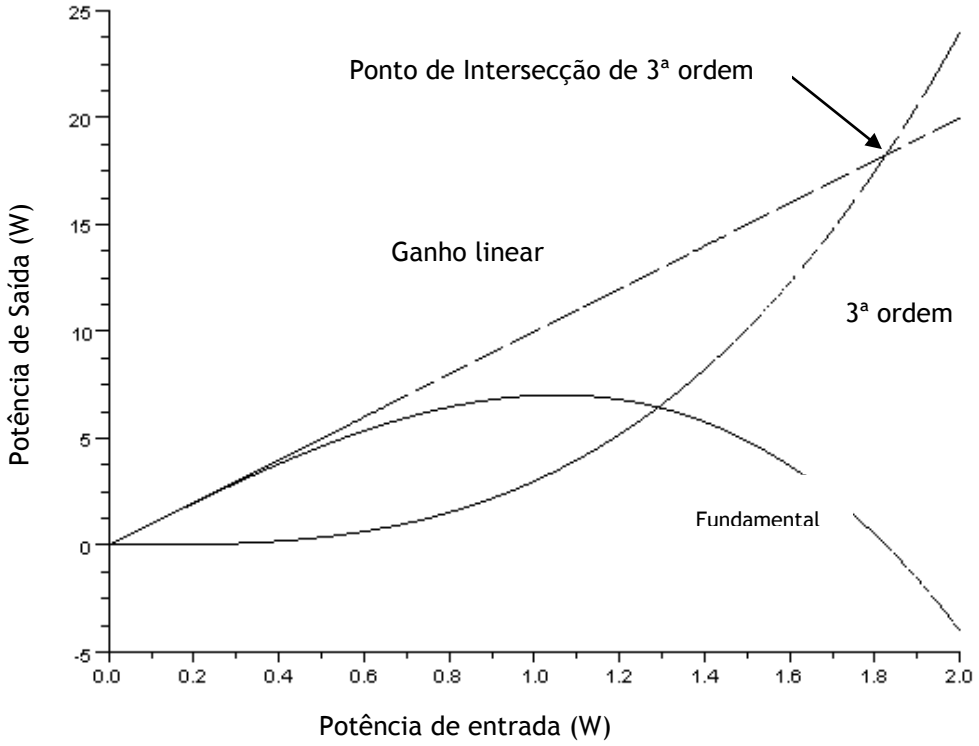


Figura 2-5 Figura do ponto de intersecção de terceira ordem, IP3, de um amplificador não linear

A forma de onda da entrada é sinusoidal e toma o valor de $V_{in}(t) = V_p \sin(\omega t)$. Este sinal é a entrada de um amplificador com a seguinte característica de transferência: $V_{out}(t) = 10V_{in}(t) - 3V_{in}(t)^3$.

O sinal resultante na saída é pois: $V_{out}(t) = 10V_p \sin(\omega t) - 3\left[V_p \sin(\omega t)\right]^3$ que pode ser simplificado para

$$V_{out}(t) = 10V_p \sin(\omega t) - \frac{9{V_p}^3}{4}\sin(\omega t) + \frac{3{V_p}^3}{4}\sin(3\omega t) \qquad (2.1).$$

O primeiro termo representa a amplificação linear do sinal fundamental (sinal de entrada) e o termo final representa a distorção de terceira ordem. O termo do meio dá origem à forma pouco habitual da característica fundamental da figura 2-5 já que provoca o cancelamento parcial da fundamental pois surge à mesma frequência. O nível deste sinal de cancelamento é proporcional ao cubo da amplitude do sinal de entrada e daí poder rapidamente ter um efeito significativo no nível da componente fundamental no sinal de saída.

Assim, num amplificador em que a distorção de terceira ordem predomina, o ganho linear da característica fundamental tem de ser extrapolado de modo a obter-se o ponto de intercepção de terceira ordem. Como é indicado pelo ponto em que a linha contínua que corresponde às componentes fundamentais do sinal intercepta a linha a traço interrompido que corresponde às componentes de 3ª ordem do sinal de entrada na figura 2-5.

2.2 Medições em amplificadores de potência

2.2.1 Teste "Two – Tone"

O teste "*Two-Tone*" (duas frequências) é um método quase universalmente aceite de determinar a linearidade de um amplificador e pode ilustrar distorções quer de fase quer de amplitude presentes num amplificador. O efeito do teste "t*wo-tone*" é variar a envolvente do sinal de entrada por toda a gama possível de valores que pode tomar de modo a testar o amplificador ao longo de toda a sua característica de transferência.

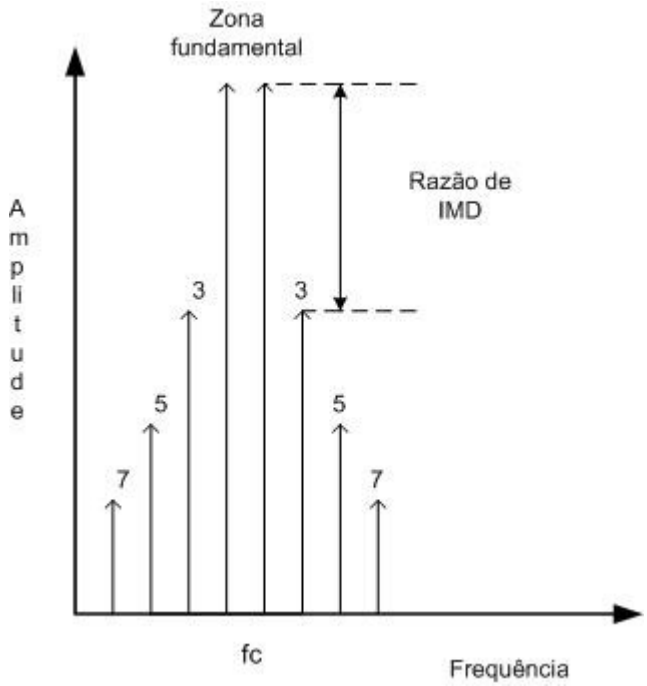


Figura 2-6 Espectro da resposta no domínio das frequências de um amplificador não linear ao qual se aplicou o teste de dois sinais de duas frequências diferentes na entrada

Quando visto no domínio da frequência, o espectro de um teste de dois tons é mostrado na figura 2-6. Os sinais individuais são frequências portadoras não moduladas e se o teste for construído cuidadosamente, não deverão aparecer outros produtos na banda de interesse (existem inevitavelmente alguns sinais presentes às frequências de harmónicos do gerador, mas deverão ser pequenos).

Se este teste for visto no domínio dos tempos, a variação da envolvente pode ser vista claramente. Os níveis do sinal deverão ser compostos de maneira a que o pico de potência da envolvente (*peak envelope power* PEP) para o sinal de duas frequências é igual à máxima potência nominal à qual o amplificador vai ser usado. Para dois sinais sem modulação, sinusoidais, com o mesmo nível, a potência de pico da envolvente do sinal resultante de duas frequências é 6 dB maior que a potência da portadora (CW) em qualquer uma das frequências

(sendo a potência média 3 dB maior do que a potência da portadora (CW) em qualquer uma das frequências.)

Da aplicação do teste de dois tons a um amplificador com não linearidade de segunda ordem resulta que cada um dos dois tons (frequências da portadora) terá um harmónico de segunda ordem e as frequências soma e diferença adicionais irão surgir.

No caso geral de uma distorção causada por uma não-linearidade de qualquer ordem, quando aplicamos o sinal de teste "*two tone*", novas frequências serão geradas, e terão a seguinte forma: $f_{imn} = mf_1 \pm nf_2$ sendo o 'm' e 'n' números inteiros positivos (incluindo o zero) e 'm+n' é igual à ordem da distorção. Assim, para uma não-linearidade de terceira ordem, as frequências adicionais serão:

$$f_{im1} = 3f_1;\ f_{im2} = 3f_2;\ f_{im3} = 2f_1 + f_2;\ f_{im4} = f_1 + 2f_2;\ f_{im5} = 2f_1 - f_2;\ f_{im6} = f_1 - 2f_2; \quad (2.2)$$

2.2.4 Técnicas de Linearização de Amplificadores de Potência

Existem diferentes técnicas de linearização de amplificadores de potência que podem ser divididas em quatro grandes categorias: a) técnicas de linearização com realimentação negativa (*feedback*); b) técnicas de linearização com realimentação positiva (*feedfoward*); c) técnicas de pré-distorção e d) técnicas de transmissores lineares usando processamento de sinal [1].

Os amplificadores de potência podem ser linearizados com realimentação negativa. Algum do sinal de saída é devolvido à entrada reduzindo da igual forma o ganho e a distorção.

Quando a realimentação negativa de radiofrequência não é suficiente, a realimentação modulada "envelope feedback" pode aumentar em alguns dB a redução da distorção.

A pré-distorção pode ser usada para linearizar um amplificador cujas características são estáveis ao longo do tempo. Uma não linearidade é adicionada à forma de onda de entrada de maneira a tornar a saída linear.

A linearização com realimentação positiva "feed-foward" pode ser autocorrectiva e bastante fiável ao longo do tempo. A saída do amplificador é amostrada e subtraída à entrada e o sinal de erro resultante é amplificado e subtraído à saída, cancelando a distorção [17].

2.2.4.1 Realimentação negativa de RF ("*RF feedback*")

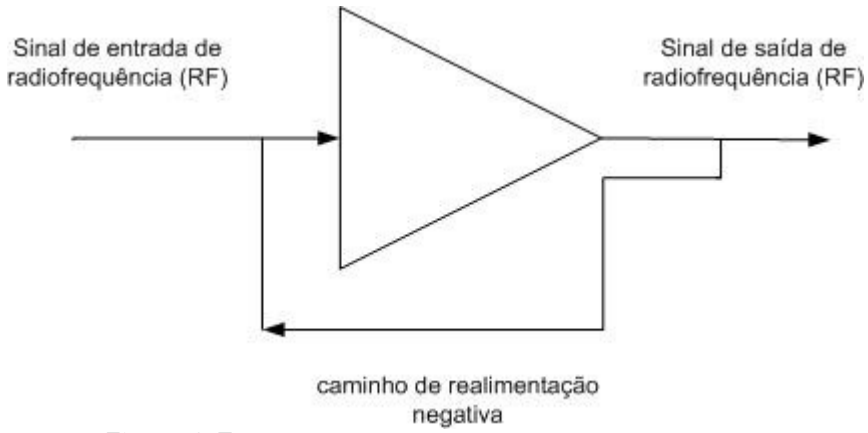


Figura 2-7 Realimentação negativa num amplificador de RF

A adição de realimentação negativa reduz o ganho do amplificador e torna-o mais linear. Todos os produtos de distorção são reduzidos pela redução de ganho em dB.

Se a redução do ganho é muito elevada os resultados são excelentes.

No entanto, o amplificador com malha de realimentação negativa pode oscilar se o atraso através do amplificador e da malha de realimentação for muito grande [17].

2.2.5 Método de medida de não-linearidade em amplificador de potência integrado em tecnologia CMOS 0,35um

Para assegurar altos níveis de linearidade ao longo do ciclo de vida de um produto, são necessárias metodologias capazes de providenciar a avaliação e correcção de não linearidades. A medida de "*Adjacent channel power ratio*" (ACPR) é o parâmetro usualmente empregue para caracterizar efeitos de não linearidades na presença de sinais de banda larga em circuitos, em particular, amplificadores de potência.
No entanto, esta medição é demasiado complicada para ser implementada "*on-chip*" já que requer gerar um sinal de estímulo de banda larga e avaliar o circuito em teste no espectro do sinal de saída. Porém, uma estimativa de ACPR pode ser obtida a partir do ponto de intersecção de 3ª ordem da entrada (IIP3). Foi demonstrado que [5]:

$$ACPR = -20.75 + 1.6 \times CF + 2(P_{1dB} - IIP3_{sine}) \quad (2.3)$$

relação que é verdadeira, em particular, para potências de entrada maiores que -20 dBm. Usando esta aproximação é possível estimar ACPR depois de medir IIP3 através da aplicação de um estímulo em forma de uma onda sinusoidal.

O recurso a aproximação polinomial (*Polynomial fiting*) foi proposta como meio de estimar o ponto de compressão 1dB (P_{1dB}) e o ponto de intersecção de 3ª ordem (IP_3) usando uma única frequência no sinal de estímulo e sem a necessidade de dois esquemas de teste separados, a adaptação de polinómios (*Polynomial fiting*).

A abertura do diagrama de olho pode também ser correlacionada com os valores dos coeficientes do polinómio adaptado. Como referido em [5] as não linearidades afectam a abertura dos diagramas de olho. Ao definir um limite mínimo de abertura do diagrama de olho, podem-se encontrar gamas de valores aceitáveis para os coeficientes do polinómio. Sabendo estes valores, pode-se a partir dos coeficientes estimados saber se a linearidade do circuito em teste é aceitável ou não para garantir o diagrama de olho requerido. Isto significa também que se podem definir bandas de tolerância baseadas no conjunto aceitável de coeficientes do polinómio.

Observando os procedimentos descritos em [4] a característica de transferência de um amplificador pode ser obtida executando um varrimento do sinal na entrada e observando a resposta na saída. No entanto, não há acesso directo à saída, de modo que tem que se encontrar outra solução para obter o polinómio.

O método proposto em [8] permite a obtenção destes coeficientes pela aplicação de aproximação polinomial a um conjunto de 3 pontos tensão de saída/tensão de entrada obtidos

da resposta do amplificador. Esta abordagem permitiu a estimação de P1dB e IIP3 com erros da ordem de 3.8%.

2.2.6 Estimar não linearidades de amplificadores de potência RF após correlação cruzada entre a tensão e a corrente de saída

Uma estimativa do ponto de compressão 1dB e do ponto de intersecção de terceira ordem, IIP3, pode ser obtida após a correlação dos sinais dinâmicos da corrente e da tensão de saída de amplificadores de potência. Esta estimativa é realizada através de medições reais de potência e não por intermédio de medições indirectas de potência a partir de medições de tensões [8].

2.2.6.1 Medição indirecta da potência

Rectificando e fazendo a média da tensão de saída RF uma medida da potência entregue pelo amplificador pode ser obtida, tendo em conta que a resistência de carga é conhecida.

Foi demonstrado que é possível estimar o ponto de compressão 1dB (P_{in} 1dB) e o ponto de intersecção de terceira ordem (P_{in} IIP3) referidos à entrada, depois de amostrar as tensões de entrada e saída do amplificador de potência [9,10,11,4]. Com estas amostras pode-se obter os coeficientes do polinómio que melhor aproxima a curva característica de transferência do amplificador de potência.

No entanto, estas amostras têm de ser capturadas em regiões da curva característica que possuam comportamento linear e não-linear (como por exemplo as regiões de saturação e corte). Para além disso, é necessário ter cuidado para que as assimetrias sejam detectadas na forma de onda [4]. Quando é esse o caso, ambos os picos positivos e negativos têm de ser considerados (a média das tensões de pico positivas e negativas foi também usada) usando detectores de máximo e de mínimo de forma a obter uma caracterização completa e correcta. Por outro lado, o objectivo é estimar a potência a partir de $P = \frac{{V_{rms}}^2}{R_L}$ assumindo que a impedância de carga é conhecida.

Características da medição indirecta de potência

- A estimativa da linearidade após a medição dos picos de tensão requer a observação quer de picos positivos quer de picos negativos de tensão de modo a obter-se o pior caso da característica não-linear.
- A observação da potência de saída não permite conhecer a simetria da curva característica.
- A observação da potência de saída permite detectar o ponto de máxima transferência de potência. Este facto pode ser explorado para implementar esquemas adaptativos capazes de calibrar a rede de adaptação da saída de modo a assegurar uma operação óptima.
- As variações de impedância à volta do valor de máxima transferência de potência são mais facilmente detectadas por observação da tensão de saída do que da potência de saída, já que a tensão tem uma derivada maior neste ponto.

- À medida que a impedância aumenta a sensibilidade da tensão a estas variações diminui já que

$$\frac{dV_L}{dZ} = \frac{2\overline{a_L}\sqrt[3]{Z_o}}{(Z + Z_{o)}{}^2}$$

mas, não diminui a sensibilidade da potência de saída às variações da impedância [8]. Estes factos permitem concluir que não é indiferente usar sensores de pico ou sensores de tensão RMS (*root mean square*) que permitem obter valores de potência através da fórmula $P = {V_{rms}^2}/{R_L}$, já que o primeiro dá informação da tensão de pico, enquanto, que o segundo dá o valor da potência média.

2.2.6.2 Medição de Potência baseada na correlação cruzada

A correlação cruzada entre dois sinais, x(t) e y(t) é definida como $\Re(\tau) = \int_{-\infty}^{+\infty} x(t) * y(t+\tau)dt$, em que a variável τ é um atraso e que se transforma em

$$\Re(\tau) = \frac{1}{2}XY\cos(\omega t + \theta) \tag{2.7}$$

Onde x(t) e y(t) são sinais periódicos de período igual a T dados por x(t) = X $\sin(\omega t)$ e y(t) = Y $\sin(\omega t + \theta)$. Se x(t) e y(t) são a tensão e a corrente de um circuito, (2.8) dá a potência activa do circuito quando o atraso τ é nulo. Isto é,

$$\Re vi(0) = \frac{1}{2}VI\cos(\theta) \equiv P = \frac{1}{T}\int_0^T V(t) \times I(t)dt = \frac{1}{2}(V \times I) \times \cos(\theta) \tag{2.8}$$

O resultado da correlação dá-nos uma tensão contínua (DC) proporcional à da potência activa. No caso da tensão e da corrente apresentarem frequências diferentes a correlação também inclui a potência devida às frequências de intermodulação [8].

2.4 Métodos de aproximação polinomial

Nas secções anteriores foram apresentados os principais parâmetros de caracterização do desempenho de amplificadores RF, dos métodos communmente usados para os determinar, de descrever os principais efeitos da presença de não linearidades no desempenho dos sistemas de transmissão, e métodos usados de melhoria de linearidade. Como foi já referido a aproximação polinomial proporciona um modo de determinar os coeficientes do polinómio que melhor aproxima a característica de transferência do amplificador. São apresentados de seguida, métodos de aproximação polinomial que são normalmente utilizados para aproximar uma dada função polinomial desconhecida tendo como ponto de partida os valores dos extremos de um intervalo de valores dessa função. …

2.4.1 Introdução

O problema da aproximação consiste em determinar uma função que descreva o melhor possível o comportamento de um conjunto de pontos (x_0,f_0), (x_1,f_1), …, (x_n,f_n). Este conjunto de pontos pode ter surgido de observações efectuadas ou medições feitas durante a realização de uma experiência e, nesse caso, a função aproximação pode ser usada para prever valores em pontos intermédios (interpolar) ou para formular um modelo matemático que descreve o processo em causa [15, pp.159].

Existem diferentes métodos de aproximação polinomial e entre eles foram analisados os métodos de Lagrange, Newton e Chebychev por serem conhecidos e por já existirem diferentes implementações dos seus algoritmos em diversas linguagens de programação, para além disso, existe, para cada um deles, uma implementação em ambiente MATLAB [16] o que facilita o seu estudo. De entre estes métodos decidiu-se escolher o método de Lagrange para uma simulação em MATLAB. O método de Newton não foi considerado por ter um erro semelhante ao dado pelo método de Lagrange. Para a aplicação do método de Chebychev seria necessário conhecer antecipadamente a função a aproximar e esse não é o caso. Temos alguma informação sobre a função a aproximar mas não é suficiente.

Se existir uma quantidade significativa de erro nos valores tabelados da função então os métodos de aproximação de curvas ("*curve fitting*") deverão ser usados [16, pp.199] Como no caso em estudo, os valores são obtidos a partir de sensores com um limitado grau de precisão, contêm em si, já, um erro de medida, pelo que, os métodos dos mínimos quadrados e de Spline devem ser considerados. Escolheu-se implementar em MATLAB o algoritmo da Spline por dois motivos essenciais, o primeiro é que o método dos mínimos quadrados já está implementado na Linguagem MATLAB e, para além disso, o método da Spline tem um erro mais pequeno, pois é capaz de detectar com maior rapidez variações bruscas na curva da função.

2.4.2 Método de Lagrange

O método de interpolação de Lagrange é um método simples de encontrar o polinómio único de ordem L que passa pelas L+1 amostras distintas de um sinal. Uma vez conhecido o polinómio, o seu valor pode ser facilmente interpolado em qualquer ponto usando a equação do polinómio [39].

Dado o polinómio de ordem L, $P(x) = a_0 + a_1 x + \cdots + a_L x^L = \sum_{K=0}^{L}(a_K x^k)$ e os L+1 valores de $P(x_K)$ para diferentes x_K, com K∈ $\{0,1,\ldots,L\}, x_i \neq x_j, i \neq j$, o polinómio pode também ser escrito como $P(x) = \sum_{K=0}^{L}(P(x_K) \frac{(x-x_1)(x-x_2)\ldots(x-x_{k-1})(x-x_{k+1})(x-x_L)}{(x_K-x_1)(x_K-x_2)\ldots(x_K-x_{k-1})(x_K-x_{k+1})(x_K-x_L)}$. O valor deste polinómio para outro $x$pode ser calculado através da substituição nesta fórmula, ou através do desenvolvimento da mesma para determinar os coeficientes a_K [39].

A implementação em MATLAB deste método é dada pela listagem do "program 4.1 (Lagrange Approximation)" contida no livro "Numerical Methods Using Matlab" [16].

2.4.3 Comparação de dois métodos de Interpolação polinomial

Método de Lagrange V Spline cúbica para aproximar uma função continua de terceira ordem. Considerou-se uma curva característica representada pelo polinómio x^3+3x^2+2x e a partir deste polinómio foram calculados 5 pontos que serão usados depois como ponto de partida das diferentes interpolações polinomiais. No final compara-se os erros obtidos.

Implementaram-se dois algoritmos em MATLAB para fazer a interpolação polinomial a partir de um conjunto de pontos conhecidos de uma curva característica representada por um polinómio de 3ª ordem. Os algoritmos usados foram respectivamente o dado pela listagem do "program 4.1 (Lagrange Approximation)" e o dado pela listagem "program 5.3 (Camped Cubic Spline) contidas no livro "Numerical Methods Using Matlab" [16].

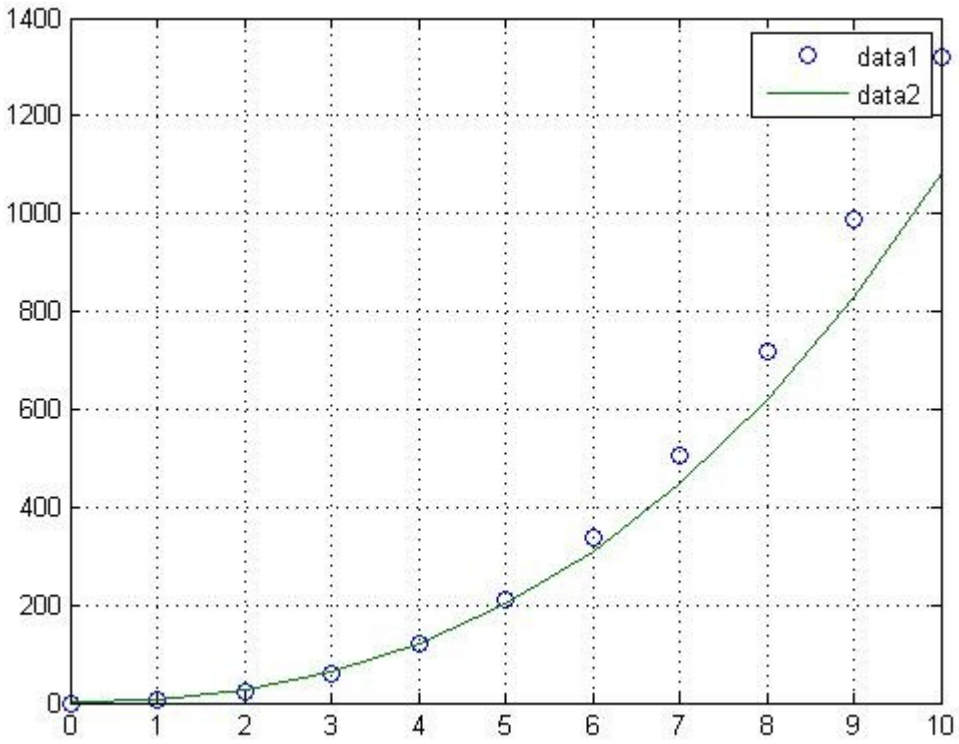


Figura 2-8 Aproximação polinomial de Lagrange

Legenda: data 1: pontos da curva característica de polinómio de 3ª ordem

data2: polinómio de lagrange

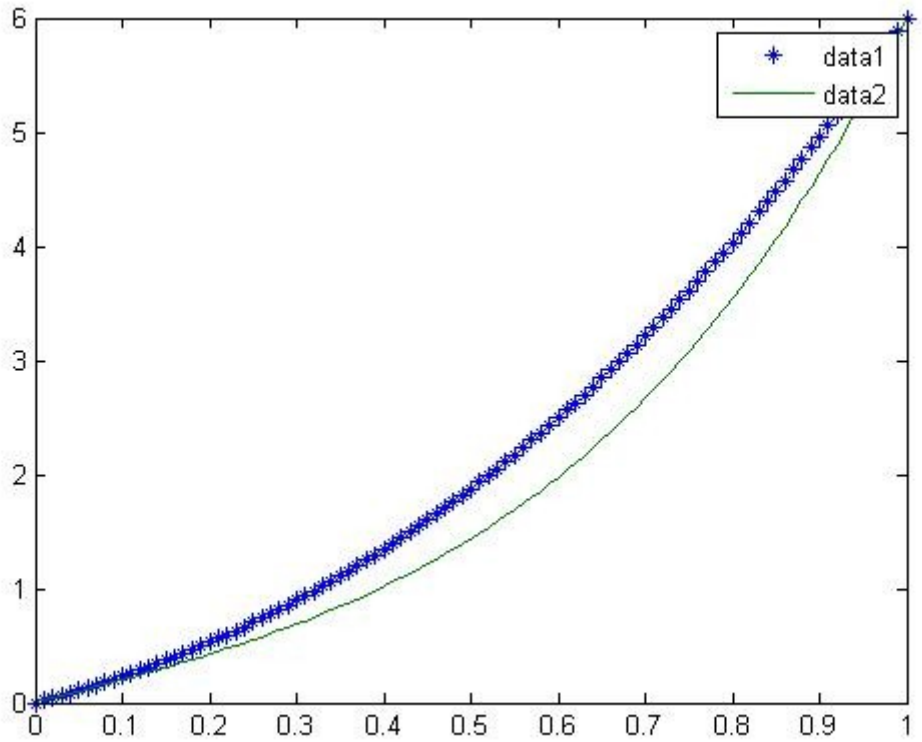


Figura 2-9 Polinómio de Lagrange de 3ª ordem versus polinómio de 3ª ordem original

Legenda: data1: polinómio lagrange; data2: curva característica de polinómio de 3ª ordem

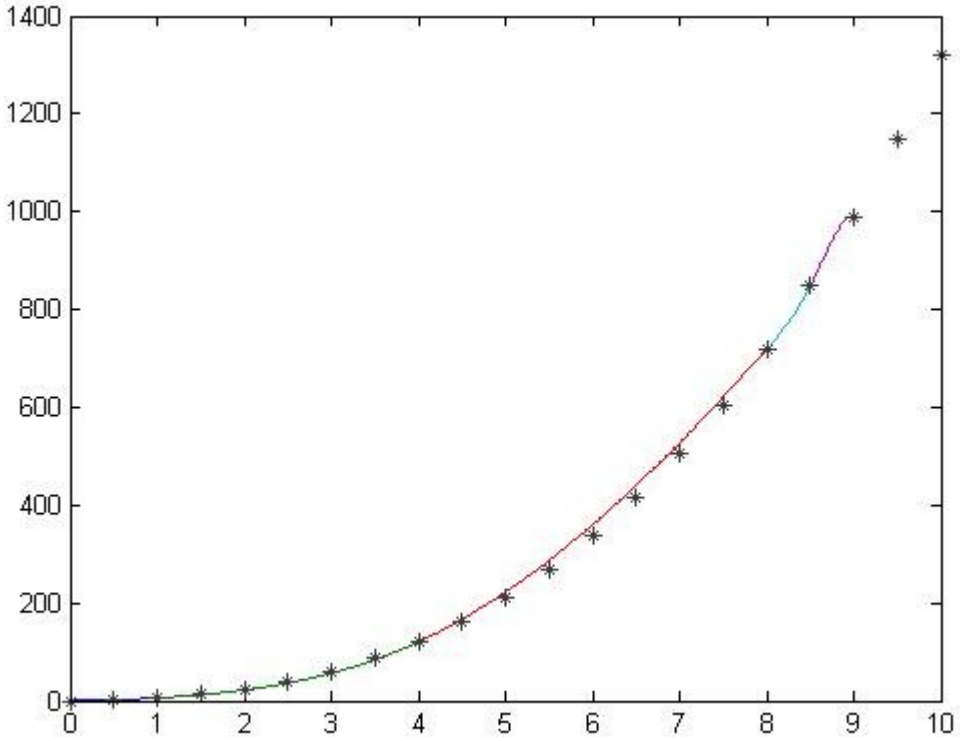


Figura 2-10 Spline cúbica

Legenda: os asteriscos '*' são pontos da curva característica representada por um polinómio de 3ª ordem. A linha contínua representa a spline cúbica calculada a partir dos mesmos pontos usados para gerar a figura 2-15.

Observando as figuras 2-15, 2-16 e 2-17, verifica-se que para valores mais elevados de x, a interpolação por spline cúbica traça uma curva mais próxima dos pontos dados pelo polinómio x^3+3x^2+2x do que a interpolação pelo método de Lagrange.

Quanto maior for o valor de x, mais influência no resultado do cálculo do polinómio têm as potências de x. Como foi descrito em 2.1.2 e 2.1.3 os termos de 2ª e 3ª ordem correspondem às não linearidades. Assim, como o nosso objecto de estudo são as não linearidades e o final do intervalo das amostras corresponde às potências de maior valor e portanto, à região de não linearidade de saturação do amplificador, consideramos o método de aproximação polinomial de spline cúbica o mais adequado para o nosso trabalho.

2.4.3.1 Definições de erro de truncatura

Além dos erros de arredondamento que se cometem durante os cálculos, outros erros chamados erros de truncatura surgem com a utilização de certos métodos numéricos para a

resolução de um problema matemático. Assim, cometem-se erros de truncatura quando se usam métodos de discretização e métodos interactivos.

Porque os nossos recursos são limitados, os processos iteractivos devem ser terminados ao fim de um número finito de operações, desde que se tenha a garantia de que a aproximação calculada se encontra muito perto da solução exacta. O erro de truncatura cometido, nestes casos, dá-nos a diferença entre o valor conseguido, quando da paragem do processo iteractivo, e o valor exacto que seria obtido no limite.

Os Limites superiores para os erros de truncatura da interpolação 'spline' cúbica e da sua primeira derivada são dados pelo teorema:

Seja f(x) uma função contínua com derivadas contínuas até à quarta ordem no intervalo [a,b], com a=x0<x1<…<xn-1<xn=b. Seja s3(x) uma função polinomial 'spline', composta pelos polinómios $s^i_3(x)$, i=1, …,n, cúbica completa para aproximar f(x) no intervalo [a,b]. Se max $_{\xi \in [a,b]}$ $|f^{(IV)}(\xi)| <= M_4$

Então, $|f(x)-s_3(x)| =< \frac{5}{384}h^4M_4$, e $\left|f'(x) - s'_3(x)\right| \leq \frac{1}{24}h^3M_4$, para $x \in [a,b]$, em que

$h = máx_{\ 0\leq i\leq n-1}(x_{i+1} - x_i)$

O erro de truncatura da aproximação polinomial dada pela fórmula interpoladora de Lagrange é dado por: $f(x) - p(x) = (x - x_0)(x - x_1) \ldots . (x - x_{n-1})(x - x_n)\frac{f^{(n+1)}(\xi)}{(n+1)!}$, em que f(x) é a função à qual queremos aproximar o polinómio p(x) dado pela fórmula interpoladora de Lagrange.

2.5 Sumário

Neste capítulo, verificamos que as não linearidades podem ser representadas pelos coeficientes de ordem mais elevada de um polinómio. Descrevemos métodos para caracterizar as não linearidades como o são o ponto de compressão de 1dB e o ponto de intersecção de terceira ordem. Apresentamos os métodos de medição de não linearidades tradicionais, o método de dois tons, bem como novos métodos de medição *on-chip*, por exemplo, um método de medida de não–linearidade em amplificador de potência integrado em tecnologia CMOS 0,35um. Discutiu-se ainda as diferentes formas de medir a potência eléctrica, como a medição indirecta por meio de detectores de pico e a medição directa que usa a correlação-cruzada. Finalmente descreve-se a interpolação polinomial como forma de caracterizar as não-linearidades através da descoberta das curvas de característica dos diferentes dispositivos de radiofrequência, mas, neste trabalho, apenas iremos considerar o amplificador de potência. É feita uma comparação de métodos de interpolação polinomial da qual se conclui que o método de Lagrange não é o ideal para a aplicação em causa. Já que se pode observar pela expressão do erro de truncatura ou de quantização do método de Lagrange que quanto maior for o valor de x maior é o erro e, para além disso, como não é conhecida a expressão da função das não-linearidades representada por um polinómio desconhecido existe a possibilidade de o fenómeno de Runge [16] ocorrer, pelo que, o método de Spline Cúbica surge como o indicado para a resolução do nosso problema. Para reforçar esta ideia foi realizada uma experiência em MATLAB na qual se tenta aproximar o mesmo polinómio pelos dois métodos de interpolação, Lagrange e Spline cúbica, e verifica-se que os resultados apresentados pela spline cúbica são os mais aproximados ao polinómio de referência.

Capítulo 3

Aproximação Polinomial por Meio de *Splines*

As Splines foram descritas pela primeira vez em 1946 pelo que são ligeiramente mais antigas do que o teorema da amostragem de Shannon. Em [21] Schoenberg apresenta os fundamentos matemáticos do tema e mostra como se pode usar splines para interpolar amostras igualmente espaçadas de uma função. Introduziu também o conceito de B-splines, os elementos atómicos básicos da construção de splines polinomiais. No entanto, o assunto dos splines esteve mais ou menos adormecido durante os anos cinquenta do século vinte, enquanto o processamento de sinal se desenvolveu a um ritmo acelerado dentro da moldura criada pelas funções de banda limitada e elegantes de Shannon. Os splines apenas arrancaram nos inícios dos anos sessenta do século passado quando matemáticos se aperceberam de que estas funções podem modelar processos físicos ou desenhar uma curva de curvatura mínima. Isto, criou um grande interesse no assunto e rapidamente as aplicações surgiram na teoria das aproximações [22], [23], análise numérica [24] e noutros ramos da matemática aplicada [25].

Com o advento dos computadores digitais, os splines chamaram a atenção dos engenheiros e tiveram um impacto tremendo no desenho assistido por computador [26], [27] e na computação gráfica [28]. No entanto houve pouca passagem para o processamento de sinal, talvez porque os investigadores deste campo ficaram tão acostumados a pensar em termos de funções de banda limitada que não analisaram esta hipótese. Recentemente, graças, em parte, a uma nova maneira de pensar, criada pela teoria de "*wavelet*" [29] a situação mudou significativamente.

3.1 Interpolação de Spline – Polinómios de Spline

Uma 'spline' é uma função segmentada e consiste na junção de várias funções definidas num intervalo, de tal forma que as partes estão ligadas umas às outras de uma maneira contínua e suave [15]. Os pontos de junção dos segmentos chamam – se nós. Para uma spline de grau n, cada segmento é um polinómio de grau n, o que sugere que são necessários n+1 coeficientes para descrever cada segmento. No entanto, há ainda uma condição de suavidade que impõem, a continuidade da spline e das suas derivadas até à ordem n-1 nos nós, de modo a que, efectivamente, apenas haja um grau de liberdade por segmento.

3.2 Spline Cúbico

3.2.1 Cálculo matricial de Spline Cúbico

Em cada segmento i, pretende-se construir um polinómio de grau 3, s_3^i, que passa pelos extremos do segmento, $[x_{i-1},x_i]$. A segunda derivada deste polinómio é uma forma linear, que pode ser construída a partir de polinómios de Lagrange, a passar pelos dois pontos x_{i-1} e x_i ,

$$s_3^{i''} = \frac{x-x_i}{x_{i-1}-x_i} M_{(x_{i-1})} + \frac{x-x_{i-1}}{x_i-x_{i-1}} M_{(x_i)} \quad (3.1)$$

em que $M_{(xi-1)}$ e $M_{(xi)}$ são os valores da segunda derivada da função naqueles pontos, embora desconhecidos. Integrando $s_3^{i''}$, duas vezes, em ordem a x, obtém-se uma forma polinomial cúbica que é precisamente a expressão da 'spline' cúbica do segmento i

$$s_3^i = \frac{M(x_{i-1})}{6(x_i-x_{i-1})}(x_i-x)^3 + \frac{M(x_i)}{6(x_i-x_{i-1})}(x-x_{i-1})^3 + \left[\frac{f(x_{i-1})}{(x_i-x_{i-1})} - \frac{M(x_{i-1})\,(x_i-x_{i-1})}{6}\right](x_i-x) + \left[\frac{f(x_i)}{(x_i-x_{i-1})} - \frac{M(x_i)\,(x_i-x_{i-1})}{6}\right](x-x_{i-1})\ para\ i = 1,2,\dots,n. \quad (3.2)$$

A continuidade nas primeiras derivadas nos n-1 pontos interiores, x1,x2,…,xn-1 define as seguintes n-1 condições

$$s_3^{i'}(x_i) = s_3^{i+1'}(x_i)\ para\ i = 1,2,\dots,n-1. \quad (3.4)$$

Pontos interiores:

$$(x_i - x_{i-1})M(x_{i-1}) + 2(x_{i+1} - x_{i-1})M(x_i) + (x_{i+1} - x_i)M(x_{i+1}) == \frac{6}{(x_{i+1}-x_i)}\big(f(x_{i+1}) - fxi\,6xi - xi{-}1fxi{-}1 - fxi\ para\ o\ nó\ i \quad (3.5)$$

Este sistema só é possível e determinado se lhe forem adicionadas mais duas equações. Estas dependem do tipo de 'spline' que se pretende construir.

Pontos exteriores:

$$2(x_1 - x_0)M(x_0) + (x_1 - x_0)M(x_1) = \frac{6}{(x_1 - x_0)}\big(f(x_i) - f(x_0)\big)\, 6f'(x_0) \quad (3.6)$$

$$2(x_n - x_{n-1})M(x_n) + (x_n - x_{n-1})M(x_{n-1}) = 6f'(x_n) - \frac{6}{(x_n - x_{n-1})}\big(f(x_n) - f(x_{n-1})\big) \quad (3.7)$$

O sistema de equações lineares, que no caso da spline natural é de ordem n-1 e no caso da spline completa é de ordem n+1, é sempre tridiagonal [15], isto é, um algoritmo de matriz tridiagonal (TDMA), também conhecido como o algoritmo de Thomas, é uma forma simplificada de eliminação Gaussiana que pode ser usada na resolução de sistemas de equações tridiagonais. Um sistema tridiagonal pode ser descrito como

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i,$$

onde $a_1 = 0$ e $c_n = 0$. Na forma matricial, este sistema é escrito da seguinte forma

$$\begin{bmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \cdot & \\ & & \cdot & \cdot & c_{n-1} \\ 0 & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \cdot \\ \cdot \\ d_n \end{bmatrix}.$$

Para estes sistemas a solução pode ser obtida em $O(n)$ operações em vez de $O(n^3)$ operações necessárias para a eliminação Gaussiana. Um primeiro varrimento elimina os a_i's, e depois a (abreviada) substituição para trás produz a solução [39] [40].

3.2.2 Implementação do Algoritmo matricial de cálculo do Spline Cúbico

Decidiu-se implementar o algoritmo matricial de cálculo de spline cúbico em MATLAB de modo a poder aferir as suas virtualidades no caso da estimação da curva característica. Para este efeito foi usado o "Program 5.3 (Clamped Cubic Spline)" do livro "Numerical Methods Using Matlab" [16]. Este programa, implementa em Linguagem MATLAB as equações da resolução matricial do spline cúbico descritas em 3.1.1 e resolve-as através de funções recursivas. Esta implementação em MATLAB é uma função que toma como argumentos o vector das abcissas, X, o vector das ordenadas, Y, a derivada de 1ª ordem da função de spline para x0 e a derivada de 1ª ordem da função de spline para xn. Se se tratar de funções contínuas estes valores das derivadas tomam o valor de zero. Este programa tem como saída a matriz S dos coeficientes em ordem decrescente dos polinómios interpoladores da spline cúbica.

Apresentam-se a seguir, duas figuras com resultados da aplicação deste programa a dois conjuntos de pontos de amostra. Na figura 3-1 são considerados 36 pontos de amostra, a traço contínuo é representada a curva obtida pelo método dos mínimos quadrados e os asteriscos '*' correspondem aos pontos obtidos por interpolação de spline cúbica. O método dos mínimos quadrados é implementado através do comando 'polyfit' em MATLAB.

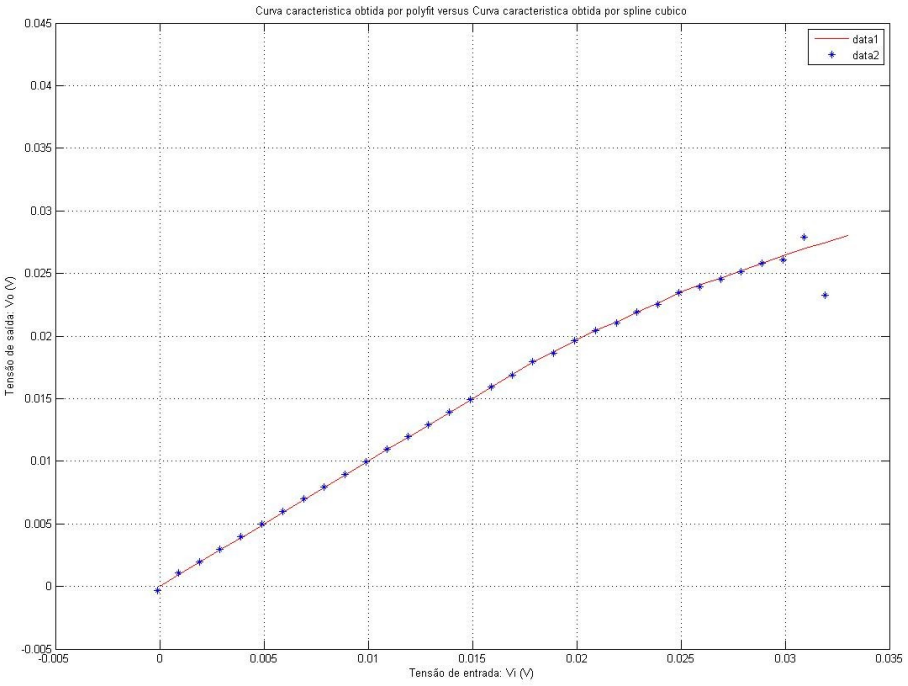


Figura 3-1 Pontos de calculo de spline cúbico v curva obtida por método dos mínimos quadrados

Verifica-se por observação da figura 3-1 que há uma boa aproximação do resultado da interpolação por spline cúbica ao resultado obtido pelo método dos mínimos quadrados.

Na figura 3-2 pode-se observar os diferentes segmentos que constituem a spline cúbica e que se situam entre cada dois pontos da amostra. Os pontos da amostra estão representados por círculos. Também, se observa que as distâncias entre os pontos da amostra não são constantes. Assim, se conclui que este método de interpolação não exige que os pontos da amostra sejam equidistantes, bem como, para a obtenção da curva de interesse não são necessários tantos pontos quanto os usados no cálculo da figura 3-1.

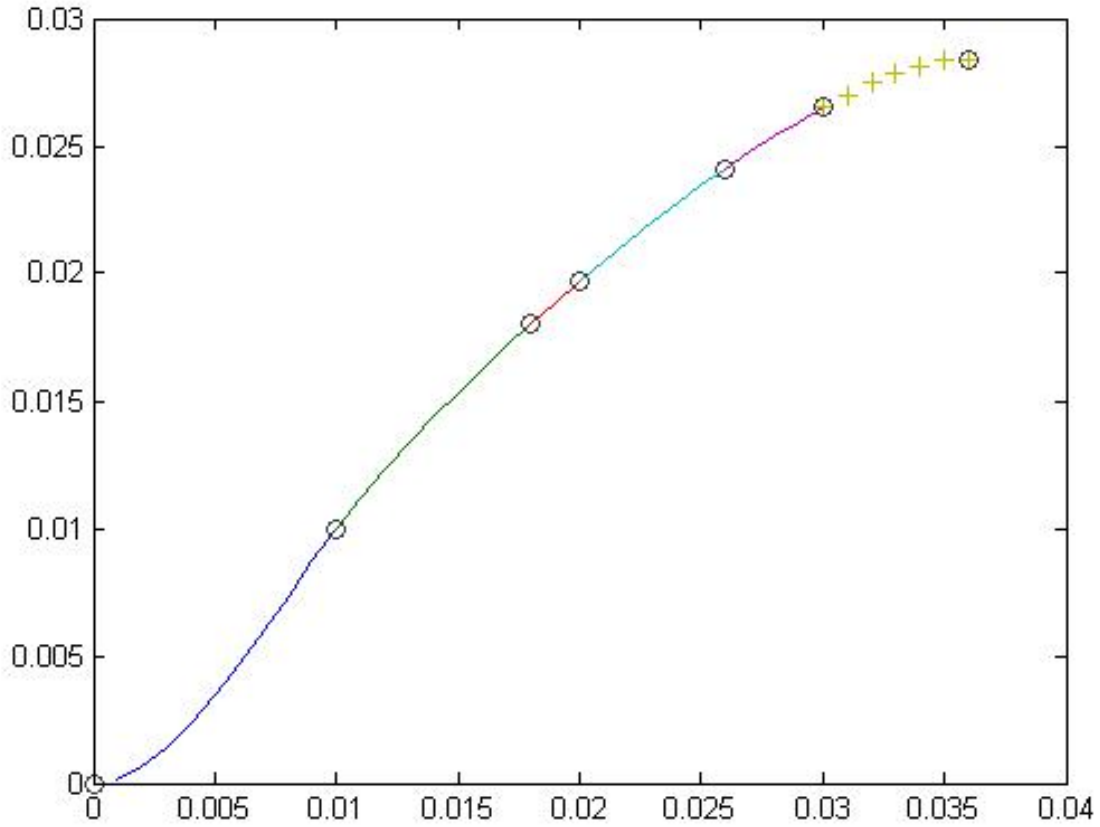


Figura 3-2 - Curva Característica Vi/Vo desenhada com base em 7 amostras e com o uso do método de interpolação de spline cúbico matricial

3.2.3 Interpolação de Spline – Polinómios de Spline – B-spline

As spline-B são splines com, nós uniformes e espaçamentos unitários. O resultado notável, devido a Schoenberg [21], é que estas splines são caracterizadas unicamente em termos da expansão de B-spline

$$s(x) = \sum_{k\in Z} c(k)\beta^n(x-k) \tag{3.8}$$

que envolve deslocamentos inteiros da B-spline central de grau n denotada por $\beta^n(x)$; os parâmetros do modelo são os coeficientes de B-spline c(k). As B-splines definidas, em baixo, são funções simétricas em forma de sino construídas a partir das n+1 convoluções do pulso rectangular β^0.

$$\beta^0(x) = \begin{cases} 1, & -\frac{1}{2} < x < \frac{1}{2} \\ \frac{1}{2}, & |x| = \frac{1}{2} \\ 0, & outro\ x \end{cases} \tag{3.9}$$

$$\beta^n(x) = \beta^0 * \beta^0 * \ldots * \beta^0 \ ((n+1)vezes)$$

- Convolução *

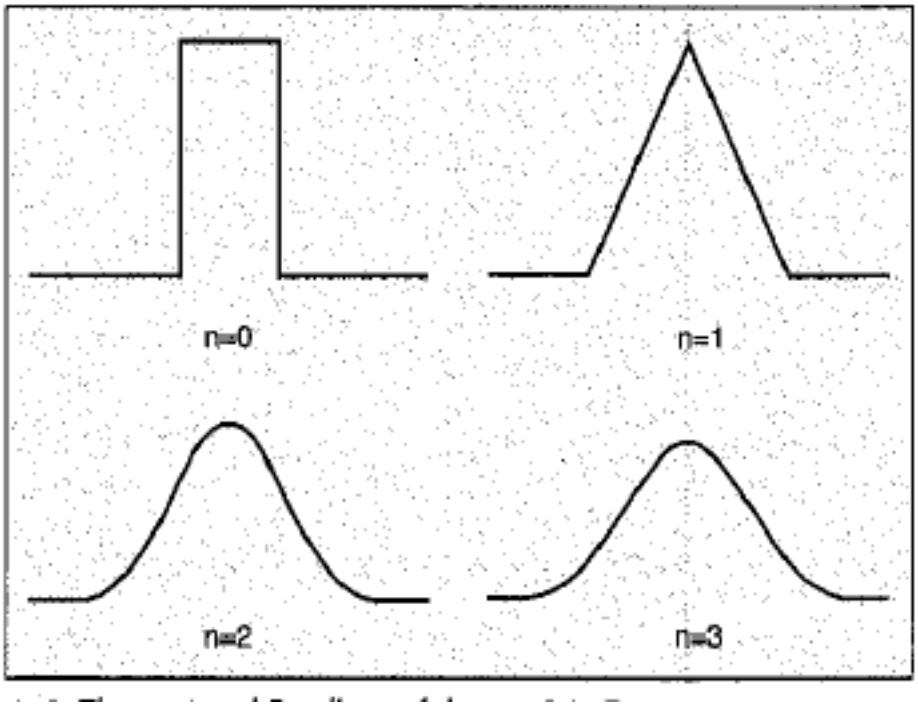


Figura 3-3 B-Splines

As B-splines de graus 0 a 3 estão representadas na figura 3-3. Uma vez que o modelo de B-spline é linear, estudar as características dos átomos básicos pode indicar-nos muito sobre as splines em geral. Graças a esta representação, cada spline é caracterizada de forma não ambígua pela sua sequencia de coeficientes de B-Spline c(k), que tem a conveniente estrutura de sinal discreto, apesar de na base estar um modelo contínuo (representação discreta/contínua).

Usando (3.9) obtemos a seguinte forma fechada de representar a B-spline que é frequentemente usada na obtenção de interpolações de grande qualidade.

$$\beta^3(x) = \begin{cases} \frac{2}{3} - |x|^2 + \frac{|x|^3}{2}, 0 \le |x| < 1 \\ \frac{(2-|x|^3)}{6}, \ 1 \le |x| < 2 \\ 0, \qquad 2 \le |x| \end{cases} \tag{3.10}$$

3.2.4 Interpolação de B-Spline via filtro digital

Pode parecer que a maioria do trabalho consiste em determinar o modelo de B-Spline de um dado sinal de entrada s(h). Consideremos agora o problema da interpolação com splines onde os coeficientes são determinados de tal forma que a função passa exactamente pelos pontos da amostra. (Fig2) Para splines de grau 0 (segmento constante) e splines de grau 1

(segmento linear), isto é um caso fácil já que os coeficientes da B-spline são idênticos às amostras de sinal: c(k) = s(h). Para graus mais elevados de spline a situação é mais complexa.

Tradicionalmente o problema da interpolação de B-spline tem sido abordado com o uso de uma matriz e de um sistema diagonal de equações que é depois solucionado através do uso de técnicas normais de métodos numéricos (substituição ou decomposição em LU) [30], [24].

No início dos anos noventa do século vinte foi reconhecido que este problema (assim como muitos outros relacionados) podiam ser também resolvidos usando técnicas mais simples de filtragem digital [31], [32], [33], [34].

Para descrever este tipo de algoritmo de processamento de sinal é necessário introduzir o conceito de B-spline kernel K discreto, que é obtido pela amostragem do B-spline de grau n expandido por um factor de m:

$$b_m^n(k) = \beta^n(x/m)|_{x=k} \overset{z}{\leftrightarrow} B_m^n(z) = \sum_{k\in Z} b_m^n(k) z^{-k} \tag{3.11}$$

Agora, dadas as amostras de sinal s(k), queremos determinar os coeficientes c(k) do modelo de B-spline (1) de tal modo que haja uma correspondência exacta nos valores inteiros, isto é, $\forall k \in Z$,

$$\sum_{l\in Z} c(l)\beta^n(x-l)|_{x=k} = s(k) \tag{3.12}$$

Usando B splines discretas esta condição pode ser reescrita na forma de convolução

$$s(k) = (b_1^n * c)(k) \tag{3.13}$$

Definindo o operador de convolução inversa

$$(b_1^n)^{-1}(k) \overset{z}{\leftrightarrow} {}^{1}/_{B_1^n}(z) \tag{3.14}$$

A solução é encontrada por filtragem inversa [97].

$$c(k) = (b_1^n)^{-1} * s(k). \tag{3.15}$$

Uma vez que b_1^né um filtro FIR (Finite Inverse Response), o chamado filtro directo de B-spline $(b_1^n)^{-1}$ é um sistema de pólos que pode ser implementado de forma muito eficiente usando filtros causais e anti-causais de primeira ordem [32] [33]. Este algoritmo é numericamente estável e é mais rápido e fácil de implementar do que qualquer outra técnica numérica.

3.2.5 Interpolação rápida por spline cúbica

Amostrando a B-spline cúbica (3.10) nas abcissas inteiras, descobrimos que

$$B_1^3(z) = \frac{(z+4+z^{-1})}{6} \tag{3.16}$$

Assim, o filtro a implementar é

$$(b_1^3)^{-1}(k) \overset{z}{\leftrightarrow} \frac{6}{z+4+z^{-1}} = 6\left(\frac{1}{1-z_1z^{-1}}\right)\left(\frac{-z_1}{1-z_1z}\right), \text{ com } z_1 = -2+\sqrt{3} \tag{3.17}$$

Dados os valores do sinal de entrada $\{s(k)\}_{k=0,\ldots,N-1}$ e definindo $c^-(k) = c(k)/6$, o lado direito da factorização leva ao seguinte algoritmo recursivo:

$$c^+(k) = s(k) + z_1c^+(k-1), (k = 1, \ldots, N-1) \tag{3.18}$$

$$c^-(k) = z_1(c^-(k+1) - c^+(k)), (k = N-2, \ldots, 0) \tag{3.19}$$

Com os valores iniciais do algoritmo dados por:

$c^+(0) = \sum_{k=0}^{k_0} s(k)z_1^k$, com $k_0 > log\varepsilon/log|z_1|$, sendo ε o nível de precisão

$$c^-(N-1) = \frac{z_1}{(1-z_1^2)}(c^+(N-1) + z_1c^+(N-2)) \tag{3.20}$$

[33] [35]

3.3 Algoritmo para cálculo do ponto de compressão 1dB

A partir de quatro vectores de entrada, dois com as tensões de entrada e de saída do amplificador e dois com as potências de entrada e saída em dBm estima-se o valor do ponto de compressão 1dB.

Para o conseguir faz-se uso da função spline definida na secção 3.1. Com essa função determinamos as ordenadas que correspondem aos valores de entrada da função. De seguida, determina-se o ganho da curva característica definida pelas tensões de entrada e saída do amplificador a partir de G= 20log10(Vo/Vi). Calcula-se para cada ponto da curva característica o ganho dado pela diferença entre a potência de saída e a potência de entrada calculadas em dBm. Podemos então determinar a função delta em cada ponto da curva característica que nos dá o nível de erro no cálculo do ponto. Com base na função delta iremos estimar onde começa e acaba a zona linear da curva característica.

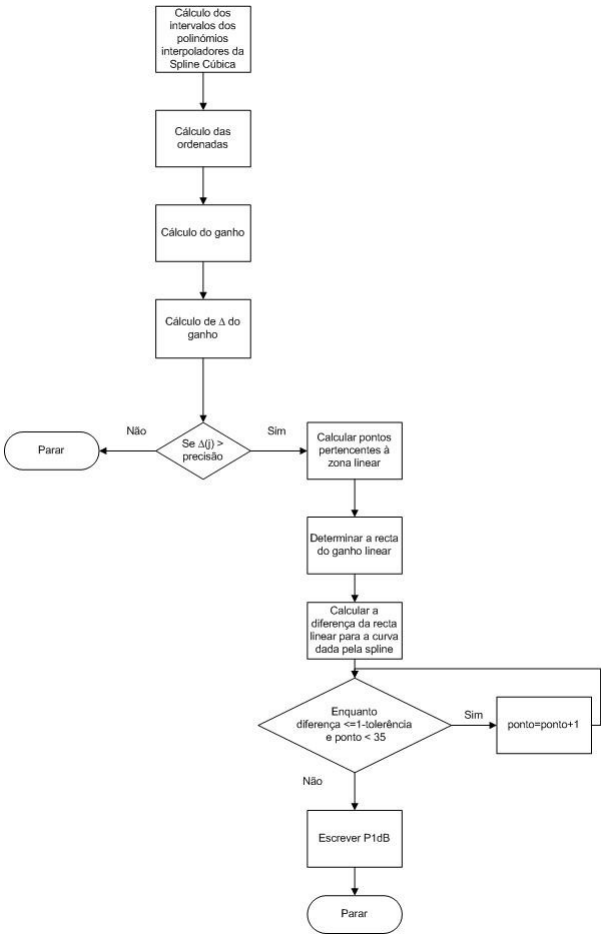


Figura 3-4 Algoritmo para cálculo de P1dB

Determina-se j como sendo o índice do vector de potências de entrada a partir do qual a curva característica deixa de ter ganho linear. De seguida, determina-se o ponto de saída adicionando à potência de entrada o valor do ganho linear, isto para todos os pontos até ao índice j. O passo seguinte consiste em determinar o declive da recta y que corresponde ao ganho linear, usando a equação $\frac{\mathrm{Pin}(2)-\mathrm{Pin}(1)}{\mathrm{Pout}(2)-\mathrm{Pout}(1)}$.

Conhecendo o segmento de recta linear determinam-se as suas ordenadas que são depois comparadas com as ordenadas obtidas pela função de spline para a mesma abcissa ou potência de entrada. Quando a diferença das ordenadas ou potências de saída for igual a 1dB foi encontrado o ponto de compressão de 1dB.

Este Algoritmo foi desenvolvido com base no artigo "Automatization of compression point 1dB (CP1dB) and input 3rd order intercept point (IIP3) measurements using lab VIEW" [19] e o trabalho principal consistiu na sua adaptação à linguagem usada no MATLAB.

3.3.1 Resultados

Com a utilização do algoritmo descrito em 3.3 e com o uso da função para calculo de spline cúbico descrita em 3.2 foram realizadas algumas simulações em MATLAB. De seguida apresentam-se alguns resultados obtidos.

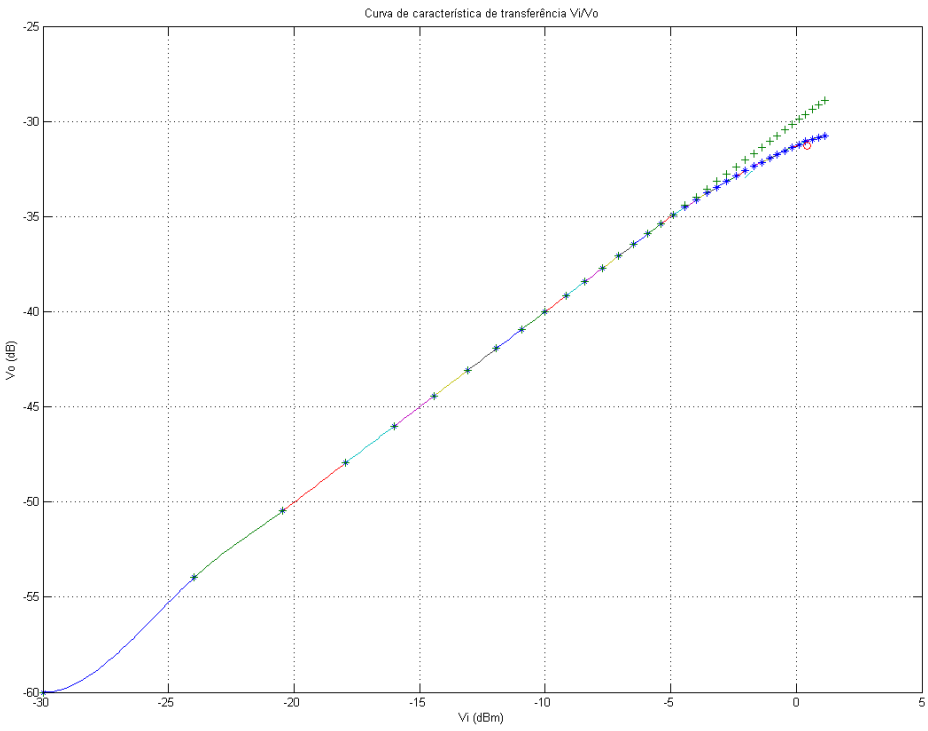


Figura 3-5 Curva característica das tensões do amplificador e recta de característica linear ideal desenhada com o uso das funções MATLAB descritas em 3.3 e 3.2

Com a finalidade de testar as funções atrás descritas usaram-se os dois vectores de tensões apresentados a seguir:

```
Vi=[0.001 0.002 0.003 0.004 0.005 0.006 0.007 0.008 0.009 0.010
0.011 0.012 0.013 0.014 0.015 0.016 0.017 0.018 0.019 0.020 0.021
0.022 0.023 0.024 0.025 0.026 0.027 0.028 0.029 0.030 0.031 0.032
0.033 0.034 0.035 0.036]; - vector amostra das tensões de entrada do amplificador
```

```
Vo=[0.001 0.002 0.003 0.004 0.005 0.006 0.007 0.008 0.009 0.010
0.011 0.012 0.013 0.014 0.015 0.016 0.017 0.018 0.0188 0.0197 0.0205
0.0212 0.0220 0.0227 0.0235 0.0241 0.0247 0.0253 0.0259 0.0265 0.0270
0.0275 0.0280 0.0284 0.0287 0.0290]; - vector amostra das tensões de saída do
```

amplificador

Com base nestes vectores e num script (disponível no anexo A3 cálculo_P1dB_IP3_ganho) construído com base nas fórmulas de referência [20] para cálculo do ponto de compressão 1dB e do ponto de intercepção de terceira ordem IP3 obtiveram-se os valores de referência.

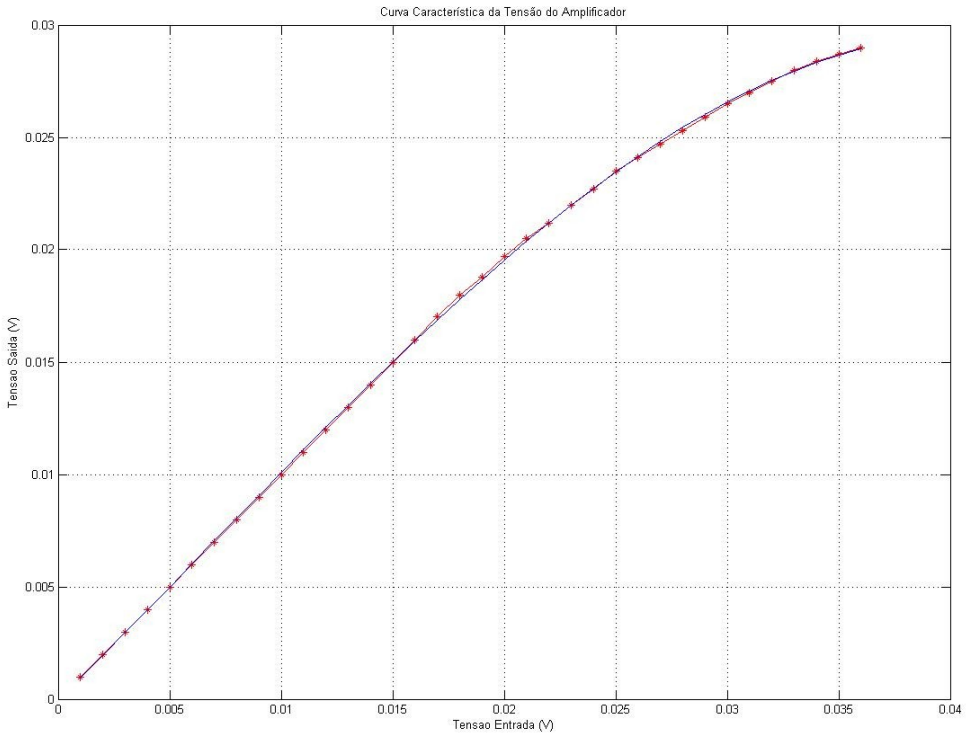


Figura 3-6 Curva da característica de tensão do Amplificador

Na figura 3-6 pode-se observar a curva característica de tensão do amplificador obtida a partir de 36 amostras e do comando polyfit do MATLAB.
Para calcular o ponto de compressão de 1dB usaram-se as fórmulas que se podem ver no excerto de código MATLAB que se apresenta, a seguir e os coeficientes do polinómio obtido como resultado do comando polyfit.

```
-------------------------------------------------------------------------
% Obtençao do polinomio de 3ª ordem que melhor aproxima a funçao
% de transferencia Vin vs Vout

[a,s]=polyfit(Vi,Vo,3);

a0=a(4)
a1=a(3)
a2=a(2)
a3=a(1)

Vy=a0+a1.*Vi+a2.*Vi.^2+a3.*Vi.^3;
-----------------------------------------------------------

-----------------------------------------------------------
V_1dB_in=sqrt(abs(((a1/(10^(1/20)))-a1)*(4/(3*a3))));
P_1dB_in=0.5*V_1dB_in.^2;
P_1dB_in=10*log10(P_1dB_in/1e-3)

V_1dB_out=a1.*V_1dB_in+3/4*a3.*V_1dB_in.^3;
P_1dB_out=0.5*V_1dB_out^2;
P_1dB_out=10*log10(P_1dB_out/1e-3)
-------------------------------------------------------------------
```

Para calcular o IIP3 também se recorreu a fórmulas e aos coeficientes do polinómio. As fórmulas usadas são apresentadas no excerto de código MATLAB que se segue.

```
-------------------------------------------------------------------------
ponto_intermodulacao_saida=abs((2*a1^3)/(3*a3));
ponto_intermodulacao_saida_dBm=10*log10(ponto_intermodulacao_saida/1e-3);
IP3_out=ponto_intermodulacao_saida_dBm

ponto_intermodulacao_entrada_tensao=sqrt(ponto_intermodulacao_saida/(0.5*a1^2)
);
ponto_intermodulacao_entrada_pot=0.5*ponto_intermodulacao_entrada_tensao^2;
```

```
ponto_intermodulacao_entrada_dBm=10*log10(ponto_intermodulacao_entrada_pot/1e-
3);
IP3_in=ponto_intermodulacao_entrada_dBm
--------------------------------------------------------------------------------
```

Deste modo, obtiveram-se os valores de P1dB e IP3 de referência.

Resultados Obtidos:

P_1dB_in =-6.0(2) (dBm)
P_1dB_out =-7.(1) (dBm)

IP3_out = 3.5(4) (dBm)
IP3_in = 3.6(2) (dBm)

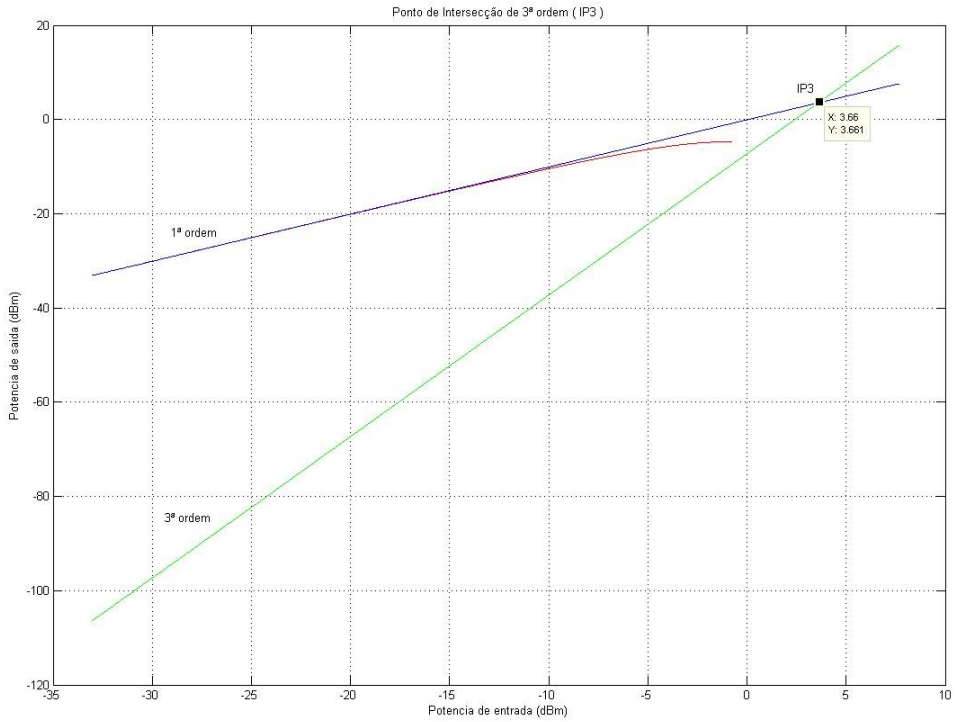


Figura 3-7 Ponto de Intersecção de 3ª ordem

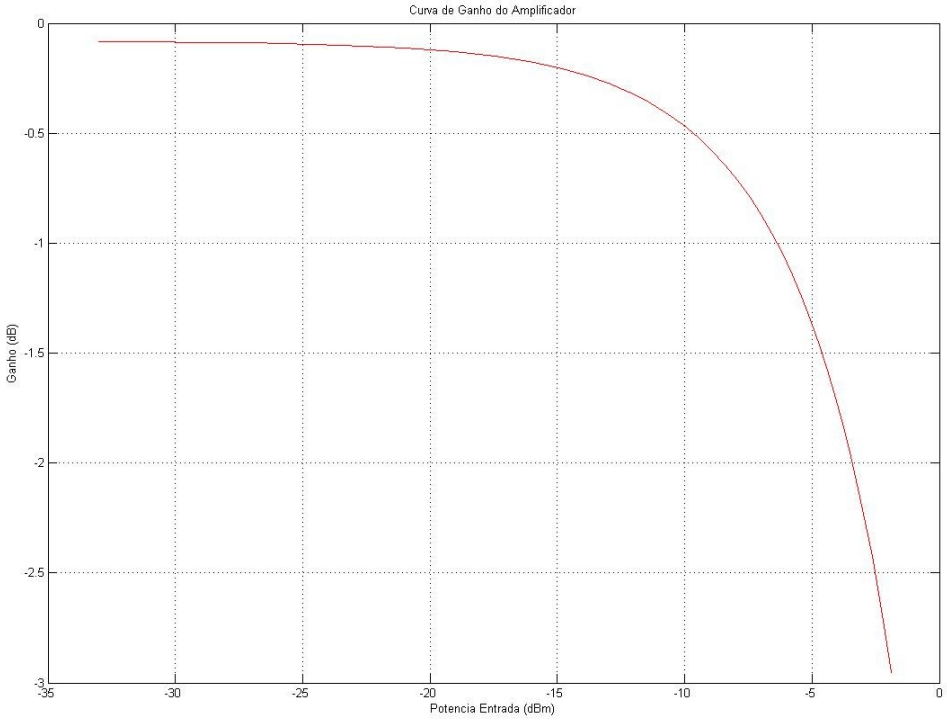


Figura 3-8 Curva do ganho do Amplificador

Os valores de referência podem ser comparados com os valores obtidos usando os algoritmos descritos em 3.3 "algoritmo para cálculo de P1dB de referência" e 3.2.1 "algoritmo de spline cúbica matricial" e, para além disso, registaram-se os valores obtidos para P1dB se às variáveis de entrada fosse adicionado um erro aleatório de 1 e 5%.

Tabela 0-1 Valores de P1dB estimados com interpolação de spline cúbico matricial e 36 amostras

P_1dB_in [dBm]	P_1dB_out [dBm]	% de erro
-3.4(7)	-4.5(5)	0
-3.4(8)	-4.5(1)	1
-3.3(2)	-4.3(4)	5

São apresentados na tabela 3-1 os resultados de cálculo de P1dB obtidos usando interpolação baseada em spline cúbico matricial com 36 pontos

Comparando estes resultados com os obtidos na figura 3.6, podemos afirmar que há uma diferença de cerca de 7,631 mV, o que corresponde a um erro de 21,197 % numa escala de 0 a 36 mV.

São apresentados a seguir resultados do cálculo de P1dB através de interpolação baseada em spline cúbico matricial com 6 pontos. As listagens dos *scripts* MATLAB que implementam cada uma das opções das tabelas de cima podem ser consultadas em anexo.

Tabela 0-2 Valores de P1dB estimados com interpolação de spline cúbico matricial e 6 amostras

P_1dB_in [dBm]	P_1dB_out [dBm]	% de erro
-3.4(7)	-4.5(4)	0
-3.4(8)	-4.5(0)	1
-3.3(2)	-4.3(7)	5

Comparando estes resultados com os obtidos na figura 3.6, podemos afirmar que há uma diferença de cerca de 7,631 mV, o que corresponde a um erro de 21,197 % numa escala de 0 a 36 mV. E comparando com os resultados para 36 amostras podemos ver que não há diferenças significativas consoante o número de amostras que são consideradas para a aplicação dos algoritmos.

De modo a poder comparar o método de interpolação dos mínimos quadrados com o método de interpolação por spline cúbico matricial com 36 amostras usou-se o algoritmo descrito em 3.3 e aplicou-se o comando "polyfit" de MATLAB que implementa este método. Os resultados obtidos são os seguintes: **P_1dB_in = -3.(8) dBm**; **P_1dB_out = -4,(7) dBm**

Estes resultados demonstram que não há diferença significativa entre o uso do método de interpolação por spline cúbico matricial de 36 amostras e o método de interpolação de mínimos quadrados, uma vez que o erro agora obtido é de cerca de 18,092%.

3.4 Implementação em MATLAB de Spline cúbica com recurso a filtros digitais

3.4.1 Regra de Horner

A curva característica de um amplificador de potência pode ser descrita por um polinómio de 3º grau como ficou demonstrado na secção 2.4. A regra de Horner diz que podemos representar esse polinómio apenas através de somas e multiplicações [16] [36].

3.4.1.1 Algoritmo de Horner

Suponhamos que queremos calcular o polinómio $p(x) = 4x^5 - 3x^4 + 7x^3 + 6x^2 + 3x + 9$ para x=3, o método normal de cálculo é calcular cada produto (4.3^5 ou 7.3^3) separadamente e por fim somar os valores. A desvantagem é que para calcular qualquer potência de x temos de calcular primeiro as anteriores. É claro, que uma abordagem mais simples seria compilar uma lista das potências de x, calculada recursivamente $x^n = x.x^{n-1}$, e depois usa-la.

O algoritmo de Horner é ainda mais eficiente que o método acima mencionado. Representemos um polinómio de grau n com a seguinte notação:

$$\begin{aligned} p(X) = {} & C_n X^n + C_{n-1} X^{n-1} \ldots + C_1 X + C_0 \; = \\ & = \big((C_n X + C_{n-1})X + \cdots\big)X + C_0 \end{aligned} \tag{3.20}$$

O propósito do algoritmo é calcular p(α) sendo α uma constante. O algoritmo funciona como se mostra a seguir:

1. Fazer u <- n , sendo n o grau do polinómio
2. Fazer resultado <- C_n
3. Se u = 0 parar. A resposta é resultado
4. Calcular resultado <- resultado x α + C_{u-1}
5. u <- u-1
6. Ir para o passo 3

Note-se que o algoritmo de Horner tem apenas o dobro do comprimento quando se calcula um polinómio de grau 20 do que quando se calcula um polinómio de grau 10 [21].

3.4.1.2 Implementação em MATLAB de Algoritmo de Horner

A implementação em MATLAB do algoritmo de Horner é bastante simples como se pode ver pela amostra de código que se apresenta a seguir:

```
-------------------------------------------------------------
function resultado = horner(grau,C,x)

%grau = 3;
resultado = C(grau);

for k=grau:-1:2

   resultado = resultado*x + C(k-1);

end
```

----------fim da listagem ----------

Como se pode ver trata-se da escrita da função recursiva "resultado = resultado*x + C(k-1);" num ciclo for. Consultar Anexos para ver listagem completa.

Na figura 3-11 pode-se observar o resultado desta implementação no desenho da curva dada pelo polinómio 2x^3 + 5x^2+9x +1 com o x a variar de 0 a 100. A implementação do algoritmo de horner é responsável pelos símbolos '+' e a linha corresponde ao resultado do comando MATLAB, polyfit.

Verifica-se observando a figura que a diferença entre um método e outro é nula. O que nos leva a concluir que a regra de Horner não introduz erro nos cálculos a realizar.

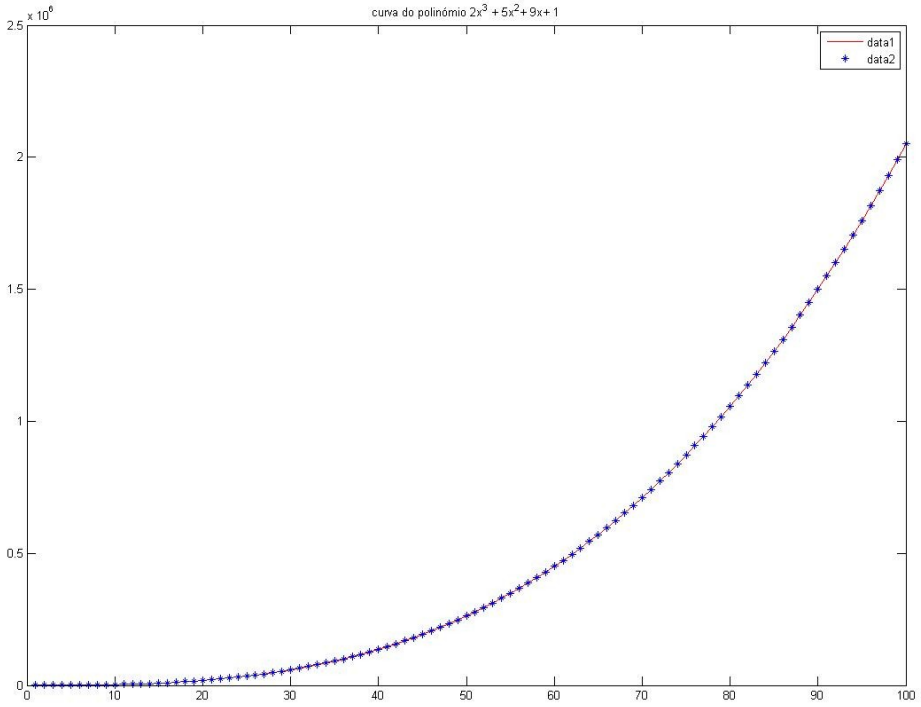


Figura 3-9 Polinómio calculado com a regra de horner

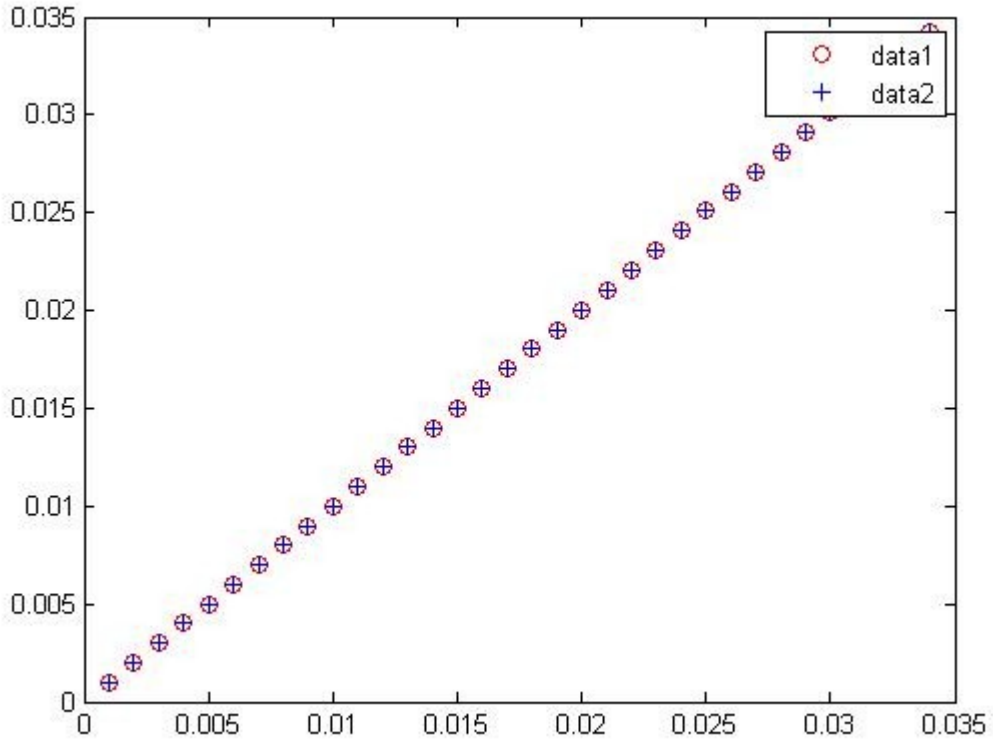


Figura 3-10(5x^3 + x) comando polyval versus regra de horner e filtro digital

A vermelho os pontos calculados com o comando de Matlab “polyval” e a azul os pontos calculados pelo filtro digital para x=[0.001:0.001:0.035] e para o polinomio 5x^3+x.
Em “Matlab” o erro obtido é inferior a 0,1%.

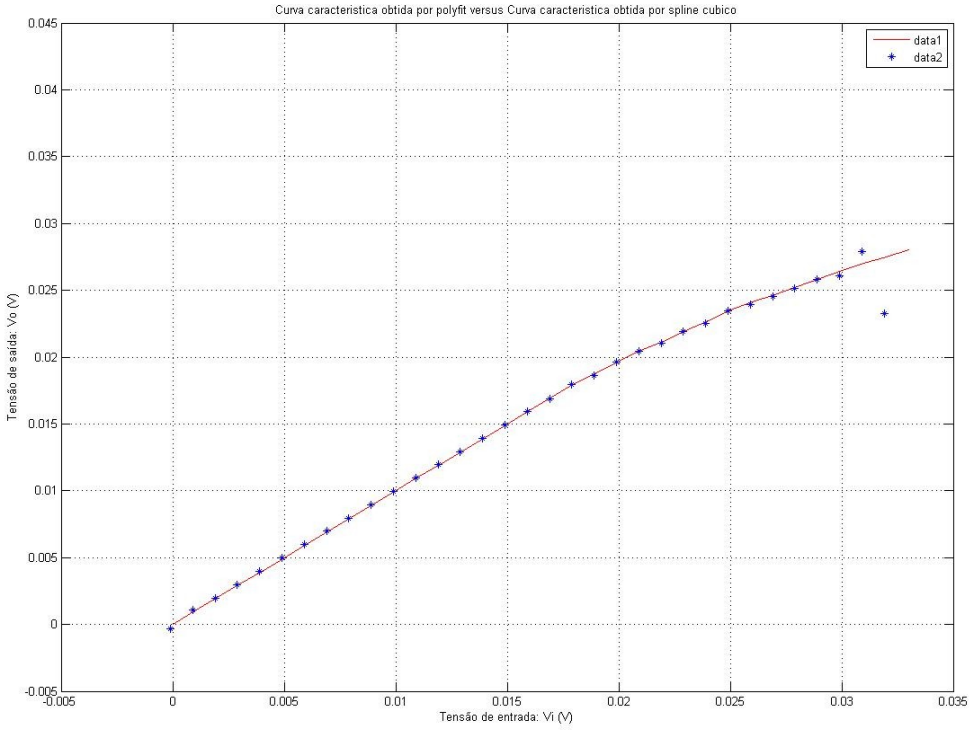


Figura 3-11 B spline cúbico implementado com filtro digital

Na Figura 3-11,a curva a vermelho representa a amostra e foi obtida através do comando “polyfit” do MATLAB. Os sinais de mais, a azul, representam o resultado da interpolação por spline cúbica para valores das abcissas diferentes dos valores das abcissas dos pontos da amostra.

Um outro exemplo de curva obtida através do spline cúbico com recurso a filtros digitais é a que está representada na figura a seguir:

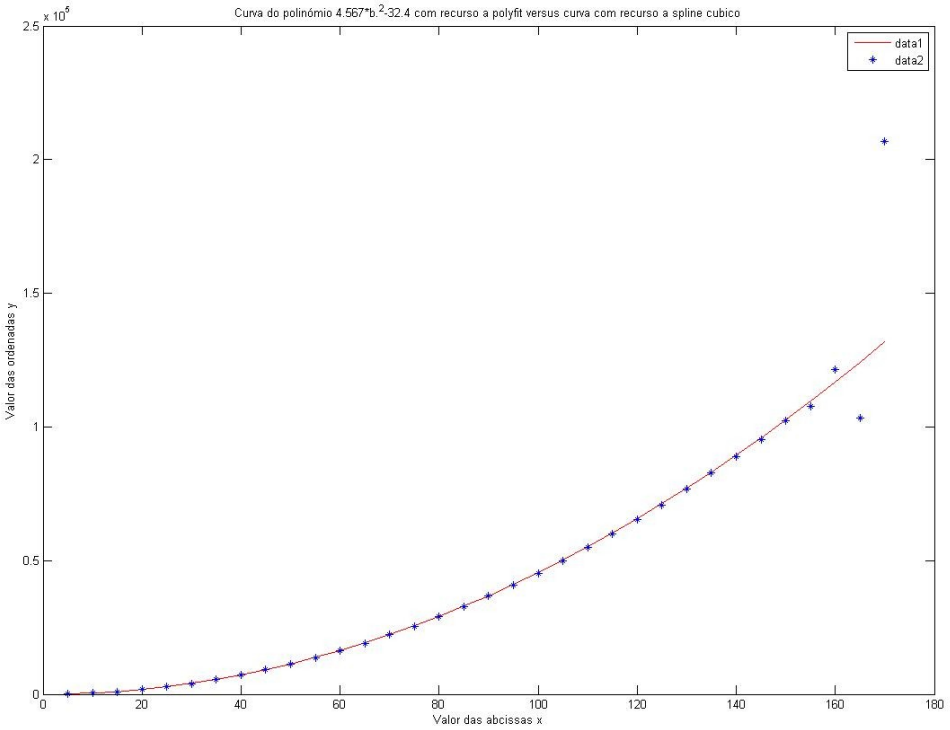


Figura 3-12 Curva do polinómio 4,567 * b^2 – 32,4 com recurso a polyfit versus com recurso a spline cúbico

3.4.2 Aplicação da regra de Horner na resolução da B - spline cúbica.

Com os algoritmos descritos em 3.2.5 para determinar os coeficientes c(k) da spline cúbica e com o algoritmo de horner descrito em 3.4.1 para implementar o polinómio da spline cúbica elaborou-se um script em MATLAB intitulado "Listagem de algoritmo de interpolação de spline cúbico com recurso a filtros digitais" que pode ser consultado em anexo. Este programa implementa filtros digitais para calcular os coeficientes c(k) da spline com base nas amostras de sinal e implementa, também, o polinómio de spline mas recorrendo a filtro digital para o implementar de acordo com a regra de horner. Finalmente, junta o resultado do polinómio com o resultado do cálculo dos coeficientes na expressão "S(k) = (horner(3,C,vector_abcissas_interpolacao-1.047)* -6*c0_menos(k)); " que nos dá o valor do "spline digital".

Com o uso desta listagem e da listagem descrita em 3.3 e as devidas adaptações, isto é, substituindo o spline matricial pelo spline "digital" obtiveram-se os seguintes resultados para um sinal com 6 amostras:

Tabela 0-3 Resultados do cálculo do ponto de compressão 1dB com recurso a interpolação por spline cúbica baseada em filtros digitais.

P_1dB_in	P_1dB_out	% de erro
-3.4(7)	-4.(5)	0
-3.(50)	-4.(5)	1
-3.(3)	-4.(4)	5

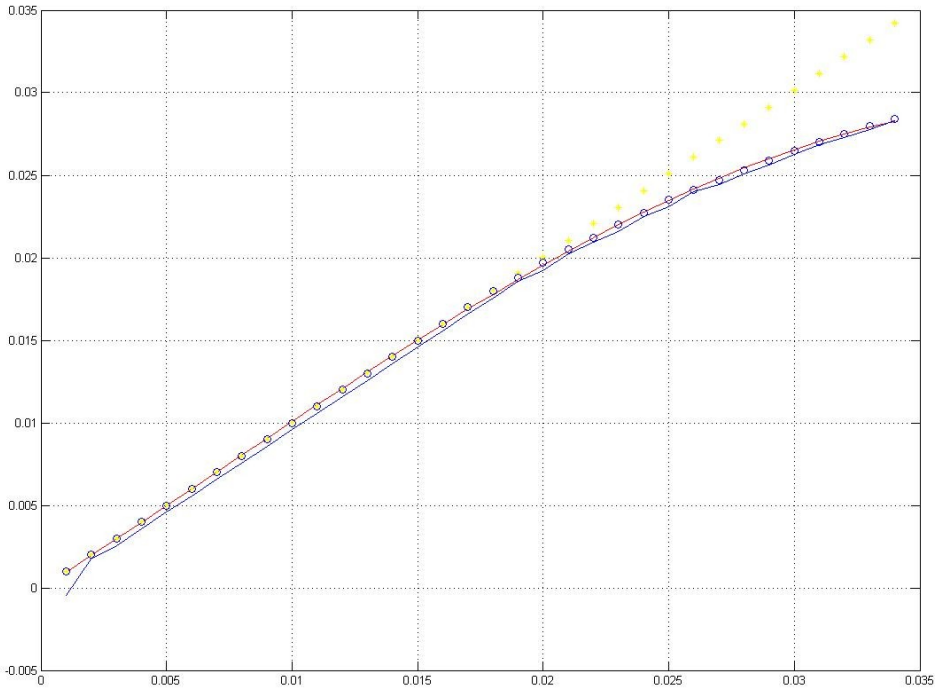


Figura 3-14 Curva caracteristica polyfit v spline IIR

Na figura 3-14 pode-se observar as amostras 'o', a vermelho o resultado da aplicação do algoritmo descrito em 3.3 ao comando polyfit e a azul o resultado se nesse mesmo algoritmo se usar o spline cúbico implementado com recurso a filtros digitais, a amarelo estão representados os pontos do ganho linear.

3.6 Algoritmo para cálculo de P1dB e IP3 a partir de uma característica de transferência conhecida

A partir de uma característica ideal do amplificador, medida em laboratório, determina-se a sua forma polinomial e respectivos coeficientes.

Após, a medição de alguns valores de tensões de entrada e respectivas tensões de saída do amplificador, obtém-se um polinómio de terceiro grau que representa a curva característica do amplificador através da interpolação pelo método dos mínimos quadrados. Em "Matlab" este processo é conseguido pelo uso do comando "polyfit".

O algoritmo para cálculo de P1dB e IP3 a partir destes dados é obtido pela execução dos seguintes passos:

- Com base nas amostras obtêm-se o polinómio de 3ª ordem que melhor aproxima a função de transferência Vin Vs Vout.
- Calcula-se P1dB e IP3 com base em fórmulas de referência.
- Calcula-se o ganho.
- Cálculo da diferença entre P1dB de referência e P1dB calculado com recurso a "Spline digital" com 0,1e 5% de erro aleatório.
- Ajusta-se os coeficientes do polinómio até se obter um P1dB dentro de uma determinada gama de valores de referência.
- Finalmente, calcula-se P1dB e IP3 com base nos coeficientes do polinómio obtidos e nas fórmulas de referência.

Após, a implementação em MATLAB deste algoritmo para FPGA obtiveram-se os seguintes resultados para o cálculo dos pontos:

P_1dB_in_final = -3.4(6)
P_1dB_out_final = -1.5(4)

IP3_out_final = 11.1(8)
IP3_in_final = 6.1(7)

Estes resultados estão ainda incompletos, pois apenas foi ajustado o valor de entrada do ponto de compressão 1dB, é possível, pelo mesmo processo, ajustar também o valor de saída, P1_dB_out. O código MATLAB usado pode ser consultado no anexo A6.

Capítulo 4

Implementação em Hardware

4.1 Introdução

No capítulo 3 desenvolveu-se a teoria de *splines* e a forma como estas podem ser implementadas através do recurso a filtros digitais, pois isso permite diminuir o número de componentes físicos necessários para a implementação do cálculo em FPGA. Com o objectivo de implementar fisicamente o algoritmo de cálculo estudado, implementaram-se em "System Verilog", uma série de blocos físicos que correspondem às funções anteriormente descritas e implementadas em ambiente MATLAB cujo código se encontra nos anexos, em.A.5 "Spline cúbico".

Como um dos objectivos do trabalho desenvolvido é obter de uma forma mais expedita o resultado de medições *on-chip*, para além da FPGA teremos também de considerar a inclusão no sistema de um ADC. Assim, temos detectores de pico e o respectivo ADC que adquirem as amostras. Amostras, essas, que depois são processadas na FPGA que, por sua vez, comunica para uma memória o resultado das 36 amostras de uma curva de transferência. Assim, a FPGA lê de uma memória os dados adquiridos pelo ADC, processa-os e escreve noutra memória o resultado da interpolação de *spline* cúbico.

A numeração usada nesta implementação é uma numeração de binária de 27 dígitos fixa. Em que o primeiro dígito corresponde ao sinal, os 11 dígitos seguintes à parte inteira do número e os 15 últimos dígitos, à parte fraccionária do número. Esta numeração foi escolhida devido à necessidade de representar uma gama de números que vai desde 0.001 até 1024.000. Isto, acontece porque, por um lado, temos amostras de 0.001 até 0.036 e, por outro lado, necessitamos de usar coeficientes de valor elevado como 1024.

4.2 Módulos Verilog

O sistema é composto por 4 blocos principais, o bloco de conversão, o bloco de cálculo do polinómio da *spline*, o bloco de cálculo dos coeficientes do polinómio de *spline* e por fim, o bloco de cálculo da *spline*. O bloco de cálculo de polinómio da spline é composto por três sub-blocos que implementam cada um, um filtro digital. O código destes blocos pode ser consultado em ANEXOS.

Em primeiro lugar é necessário digitalizar as amostras da curva de transferência do Amplificador de Potência, isto é feito por um ADC de 10 bits. O resultado desta digitalização é guardado em ficheiros binários na memória da FPGA. Definiu-se que o ficheiro de entrada do sistema que guarda os valores das tensões ou potências de entrada depois de digitalizadas pelo ADC de 10bits teria a denominação de "alfa.dat" e o ficheiro que guarda os valores da saída digitalizada teria a denominação de "beta.dat". Os ficheiros de dados, "alfa.dat" e "beta.dat" que deverão ser usados para testar o sistema estão em ANEXOS.

De seguida, é necessário ler os ficheiros "alfa.dat" e "beta.dat" de modo a poder calcular a curva de saída. Para esse fim, desenvolveram-se alguns módulos *verilog* que constituem o sistema que podemos observar na figura:

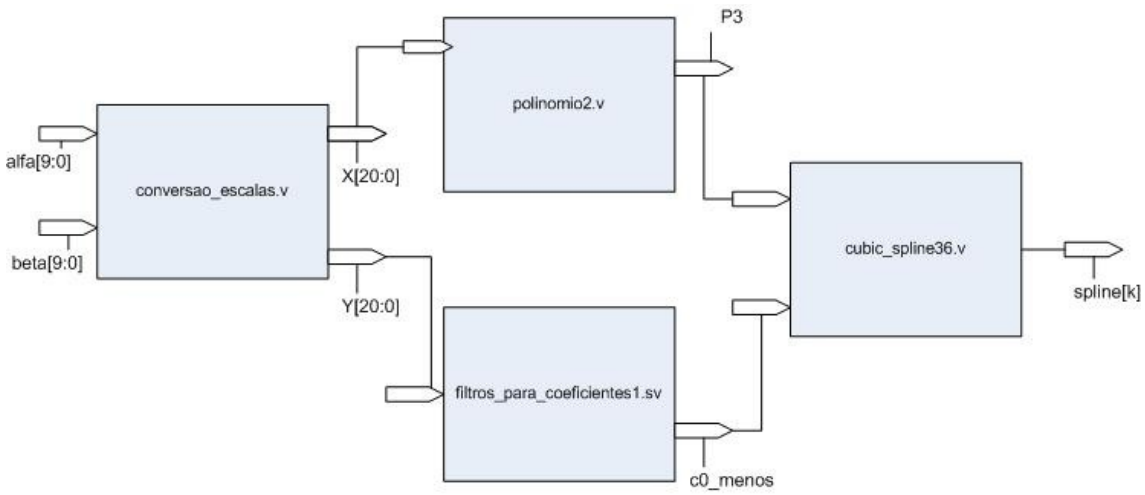


Diagrama de blocos de topo do sistema de cálculo de Spline Cúbica

Figura 4-1 Sistema Implementado

4.2.1 Módulo Verilog "conversão_escalas.v"

O objectivo deste bloco é converter a numeração binária resultante da digitalização dos valores da amostra pelo ADC de 10 bits na numeração binária escolhida de 27 bits descrita em 4.1 Para esse fim, concatena-se o resultado do ADC com o bit de sinal e com os 11 bits que representam a parte inteira do número e acrescentam-se mais cinco bits no final do número que vem do ADC.

4.2.2 Módulo de Verilog "polinómio2.v"

Este bloco é composto por 3 sub-blocos que implementam, cada um deles, o filtro digital dado pela regra de Horner. Isto, é assim, porque o verilog95 não suporta funções recursivas e uma forma de contornar esse problema consiste em criar cadeias de blocos iguais. Implementando, dessa forma, uma função recursiva, pois os sinais vão passando pela mesma função sucessivamente. Para além destes blocos estão inseridas no código constantes que correspondem aos coeficientes do polinómio que queremos implementar. Estes coeficientes foram determinados durante as simulações realizadas em software MATLAB e são baseadas no artigo científico [35] onde a função polinomial parametrizada da *spline* cúbica é descrita.

4.2.3 Módulo de System Verilog "filtros_para_coeficientes1.sv"

Para implementar o cálculo dos coeficientes a aplicar à função polinomial da *spline* cúbica foi criado o módulo "filtros_para_coeficientes1.sv" em System Verilog, porque esta versão de Verilog permite a implementação de funções recursivas. Na construção deste módulo optamos pelo uso de funções recursivas, pois desse modo, a escrita do código é mais simples e a verificação, também. As funções usadas foram baseadas no artigo em referência [35] e nas simulações realizadas com o software MATLAB descritas no capítulo 3.

4.2.4 Módulo "cubic_spline36.v"

O resultado do cálculo da *spline* cúbica é dado por este bloco, nele é implementada a multiplicação do polinómio de *spline*, P3, pelos coeficientes c0_menos e pela constante 6 [35]. Para além disso, como as funções descritas em [35] estão parametrizadas é feita uma divisão por um factor de escala.

4.3 Resultados da Implementação em Verilog

Os resultados da implementação em Verilog podem ser consultados no anexo B, e constituem os ficheiros de dados 'X.dat', 'Y.dat' que resultam do bloco de conversão de escalas, o ficheiro 'P3.dat' que reúne os valores calculados pelo bloco 'polinómio2.v', o ficheiro 'c0_menos3.dat' que contém o resultado da implementação dos filtros digitais que geram os coeficientes para o polinómio de spline e finalmente, o *script* resultante do teste do *spline* cúbico implementado.

Converteram-se 36 amostras em base decimal para uma numeração binária de 27 bits em que o bit mais à esquerda é o de sinal, os 11 seguintes correspondem à parte inteira do número e os 15 últimos dígitos correspondem à parte fraccionária do número. Como resultado da conversão temos os ficheiros 'alfa.dat' para os valores das abcissas e 'beta.dat' para os valores das ordenadas. Esta conversão introduz um erro de truncatura. Os números usados estão entre 0.001 e 0.036. Por exemplo, o erro cometido na representação de 0.001 é de $|0.001\text{-}2^{-10}| = 23.438$ µ. Ao longo das operações numéricas esses erros vão-se propagando. Daí ser necessário comparar os resultados finais do cálculo da *spline* em Hardware com os valores obtidos em Software.

De seguida, apresentam-se os resultados na numeração binária escolhida do cálculo do spline.

spline [1] = 000000000001 001010110110101, spline_signal[1]0
spline [2] = 000000000010 001101111001111, spline_signal[2]0
spline [3] = 000000000011 001001011111011, spline_signal[3]0
spline [4] = 000000000011 111101111110000, spline_signal[4]0
spline [5] = 000000000100 101011101110110, spline_signal[5]0
spline [6] = 000000000101 010011000101000, spline_signal[6]0
spline [7] = 000000000101 110100011001011, spline_signal[7]0
spline [8] = 000000000110 010000000011101, spline_signal[8]0
spline [9] = 000000000110 100110010111101, spline_signal[9]0
spline [10] = 000000000110 110111101110001, spline_signal[10]0
spline [11] = 000000000111 100011101100011, spline_signal[11]0
spline [12] = 000000000111 001100111110010, spline_signal[12]0
spline [13] = 000000000111 010001100011111, spline_signal[13]0
spline [14] = 000000000111 101101101000110, spline_signal[14]0
spline [15] = 000000001000 110100011000001, spline_signal[15]0
spline [16] = 000000001001 111101111100010, spline_signal[16]0
spline [17] = 000000001011 001001111011101, spline_signal[17]0
spline [18] = 000000001100 011000000110100, spline_signal[18]0

spline [19] = 000000001101 101000000001011, spline_signal[19]0
spline [20] = 000000001110 100001011101101, spline_signal[20]0
spline [21] = 000000001111 100110101100110, spline_signal[21]0
spline [22] = 000000010000 010001110110011, spline_signal[22]0
spline [23] = 000000010010 000001001010000, spline_signal[23]0
spline [24] = 000000010011 010001101011111, spline_signal[24]0
spline [25] = 000000010100 000111010110001, spline_signal[25]0
spline [26] = 000000010101 001001100111100, spline_signal[26]0
spline [27] = 000000010101 110101100001100, spline_signal[27]0
spline [28] = 000000010110 110001001100100, spline_signal[28]0
spline [29] = 000000010111 110100001010101, spline_signal[29]0
spline [30] = 000000011000 011100001111101, spline_signal[30]0
spline [31] = 000000011001 010010110011111, spline_signal[31]0
spline [32] = 000000000000 000000000000000, spline_signal[32]0

Para se estimar o erro deste cálculo é necessário converter a numeração para decimal como se pode ver a seguir:

Spline[0]	0
Spline[1]	1,169586
Spline[2]	2,217255
Spline[3]	3,148285
Spline[4]	3,968262
Spline[5]	4,683289
Spline[6]	5,298096
Spline[7]	5,818695
Spline[8]	6,250885
Spline[9]	6,599518
Spline[10]	6,870636
Spline[11]	7,557709
Spline[12]	7,202698
Spline[13]	7,274384
Spline[14]	7,213074
Spline[15]	8,81839
Spline[16]	9,967834
Spline[17]	11,155182
Spline[18]	12,376587
Spline[19]	13,647797

Spline[20]	14,522858
Spline[21]	15,604675
Spline[22]	16,2789
Spline[23]	18,252441
Spline[24]	19,026337
Spline[25]	20,114777
Spline[26]	21,150269
Spline[27]	21,836304
Spline[28]	22,768677
Spline[29]	23,815094
Spline[30]	24,441315
Spline[31]	25,293915
Spline[32]	0

Capítulo 5

Conclusão

No decurso deste trabalho obtiveram-se vários resultados importantes, primeiro construiu-se um método para estimar as não-linearidades baseado no cálculo do ponto de compressão 1dB; em segundo lugar analisaram-se diferentes métodos de interpolação e os seus algoritmos de cálculo tendo sido encontrado o mais preciso, rápido e fácil de implementar, o B-spline; em terceiro lugar, desenvolveu-se e implementou-se um algoritmo que permite calcular o ponto de compressão 1dB, bem como, o ponto de intercepção de 3ª ordem e, ainda, ajustar os coeficientes do polinómio da curva característica do amplificador. Finalmente, ficou demonstrado que é possível implementar em Hardware, em FPGA, o algoritmo de cálculo de B-spline.

Verificou-se que na numeração utilizada, para a implementação em *hardware* da B-spline, o aumento do número de bits da parte fraccionária do número de, 10 para 15 bits faz diminuir o erro máximo no cálculo da B-spline. Conclui-se, portanto, que existe um compromisso entre o nível de precisão desejado e a área ocupada (número de bits usados na numeração).

Demonstrou-se ainda, em termos de precisão que o algoritmo B-Spline com recurso a filtros digitais apresenta resultados muito semelhantes aos obtidos através do uso de interpolação de mínimos quadrados. Para além disso, ficou, também, provado que a introdução de um erro aleatório de 1 a 5 % não altera em muito os resultados obtidos através da interpolação por B-Spline e por Spline Matricial no cálculo do ponto de compressão de 1dB. Podendo-se, pois, concluir que estes métodos de interpolação são suficientemente robustos para a aplicação em causa.

O método de interpolação dos mínimos quadrados, embora, apresentando um elevado grau de precisão, não foi considerado adequado, por exigir o cálculo de matrizes e a utilização de numeração de vírgula flutuante. Pois, isso impede a sua implementação em FPGA dado consumir muitos recursos, nomeadamente, área de silício.

Quanto aos resultados do cálculo da spline em Hardware, verifica-se a existência de erro de aproximação provocado pela digitalização dos números da amostra, bem como pela

digitalização dos diferentes coeficientes utilizados no cálculo. Na implementação em Hardware a simplificação do cálculo permite ocupar menos área à custa de um maior número de operações que contudo são operações simples que podem ser realizadas rapidamente

Na implementação em "System Verilog" realizada verifica-se a existência de um erro máximo de 18,8 % na zona linear da curva que diminui para valores da ordem dos 10% na zona não linear da curva.

5.1 Trabalhos Futuros

Face aos resultados obtidos e às dificuldades encontradas na execução do trabalho, propõem-se o desenvolvimento de uma ferramenta automática de conversão da numeração binária escolhida em numeração decimal.

Para além disso, no desenvolvimento do código RTL deverá ser necessário, numa primeira fase simular o sistema sem ter em consideração o "range" ou alcance dos ADCs que deverão ser depois usados no protótipo, de modo a simplificar os cálculos.

Finalmente, quando o código RTL tiver já sido validado contra simulações MATLAB poder-se-á então considerar os ADCs e o seu range, bem como outras gamas de valores para a curva característica, ou seja, amplificadores com outro ganho. No entanto, há que realçar que para cada gama de valores diferentes é necessário ajustar os coeficientes do polinómio da spline cúbica visto que ele é dado por uma função normalizada e por consequência alterar o código RTL.

Para a construção de um protótipo, será ainda necessário sintetizar o código RTL para uma FPGA alvo o que, também, irá permitir saber com exactidão qual a área de silício ocupada por este processo de cálculo.

ANEXO A – Listagens de MATLAB

A.1 P1dB_Spline

```
--------------------------------------------------------------------------
function [ P_1dB_in, P_1dB_out ] = P1dB_Spline(Vi,Vo, P_in_dBm, P_out_dBm)

% Inputs : Vi - sample 36 point vector abscisses
%        : Vo - sample 36 point vector ordinates
% Outputs : P_1dB_in - Compression point 1dB abscisse
%         : P_1dB_out - Compression point 1dB ordinate
%-----------------------------------------------------------
% Author : Joana Azevedo Braga    Date : 29-04-2010

% Cálculo dos intervalos do Spline

x = 10*log10((0.5*Vi.^2)/1e-3);

for n=4:1:34
    aux_x(n) = x(n);
end

% Cálculo das ordenadas
y = spline_cubico(P_out_dBm,P_in_dBm,0,36);

for n=4:1:34
    aux_y(n) = y(n);
end

% Calculo do ganho

for n=1:1:36
g(n) = 20*log10(Vo(n)/Vi(n));
end

for i=1:1:36
    g_pot(i) = P_out_dBm(i) - P_in_dBm(i);
end

for i=1:1:36;

delta(i) = abs( g_pot(i) - g(i));
end

for j=1:1:36
if ( delta(j) > 0.0041*g(j) ) % precisão para o grau de linearidade do ganho

   A = zona_linear(j, P_in_dBm, P_out_dBm, g);
   for n=1:1:j
       P_in_dBm_linear(n) = P_in_dBm(n);
   end

   y1 = [(P_in_dBm(2) - P_in_dBm(1))/((A(2)-A(1))) 0]

   plot(aux_x,aux_y,'r-*', P_in_dBm_linear,A,'b',aux_x,polyval(y1,aux_x));
   grid on


   diferenca = polyval(y1,aux_x)-aux_y;

   m=1;
   while ( diferenca(m) <= 1.00-0.00041 && m < 35 )

   m=m+1;

   end
```

```
   P_1dB_in = P_in_dBm(m)
   P_1dB_out = P_out_dBm(m)
   break
end
end
---------Fim da Listagem do Algoritmo de cálculo de P1dB------------------
```

A.2 zona_linear

```
------------------------------ Função de Cálculo da Zona Linear ---------------
% em função de j, índice onde o ganho deixa de ser linear, calcular a porção
% linear da curva característica
% Author: Joana Braga                    7-05-2010
%--------------------------------------------------------------------------

function  [P_out_dBm_linear] = zona_linear(j, P_in_dBm, P_out_dBm, g)

% Entrada da função zona_linear: j é o índice do vector de potências de
entrada a partir do qual a curva
% deixa de ter ganho linear

for n=1:1:j
P_out_dBm_linear(n) = P_in_dBm(n) + g(n);
P_in_dBm_linear(n) = P_in_dBm(n);
end

% Cálculo da zona não linear da curva caracteristica

for n=j:1:36

    P_in_dBm_nao_linear(n) = P_in_dBm(n);
    P_out_dBm_nao_linear(n) = P_in_dBm(n);
end
--------------fim da listagem da função de cálculo da zona linear-----------
```

A.3 calculo_P1dB_IP3_ganho

```
-----------------Listagem do script de cálculos de referência ----------------
function calculo_P1dB_IP3_ganho

format long

Vi=[0.001  0.002  0.003  0.004  0.005  0.006  0.007  0.008  0.009  0.010  0.011  0.012
...
    0.013  0.014  0.015  0.016  0.017  0.018  0.019  0.020  0.021  0.022  0.023  0.024
...
    0.025 0.026 0.027 0.028 0.029 0.030 0.031 0.032 0.033 0.034 0.035 0.036];

Vo=[0.001  0.002  0.003  0.004  0.005  0.006  0.007  0.008  0.009  0.010  0.011  0.012
...
    0.013  0.014  0.015  0.016  0.017  0.018  0.0188  0.0197  0.0205  0.0212  0.0220
0.0227 ...
    0.0235  0.0241  0.0247  0.0253  0.0259  0.0265  0.0270  0.0275  0.0280  0.0284
0.0287 0.0290];

P_in=0.5*Vi.^2;
P_in_dBm=10*log10(P_in/1e-3);
P_out=0.5*Vo.^2;
P_out_dBm=10*log10(P_out/1e-3);


% Obtençao do polinomio de 3ª ordem que melhor aproxima a funçao
% de transferencia Vin vs Vout do LNA.

[a,s]=polyfit(Vi,Vo,3);

a0=a(4)
```

```
a1=a(3)
a2=a(2)
a3=a(1)

Vy=a0+a1.*Vi+a2.*Vi.^2+a3.*Vi.^3;

plot(Vi,Vo,'r-*',Vi,Vy,'b')
grid on
ylabel('Tensao Saida (V)')
xlabel('Tensao Entrada (V)')
pause


% Representaçao grafica de P1dB e IP3

num=length(Vi);
Vi_min=Vi(1);
Vi_max=Vi(num);

Vii=(Vi_min:0.00001:3*Vi_max);
aux=length(Vii);

Vy=a0+a1.*Vii+a2.*Vii.^2+a3.*Vii.^3;

Pii_dBm=10*log10(0.5*Vii.^2/1e-3);

Po_linear=0.5*a1.^2.*Vii.^2;
Po_linear_dBm=10*log10(Po_linear/1e-3);

Po_real=0.5*(a1.*Vii+3/4*a3.*Vii.^3).^2;
Po_real_dBm=10*log10(Po_real/1e-3);

Po_intermod=(9/32)*a3.^2.*Vii.^6;
Po_intermod_dBm=10*log10(Po_intermod/1e-3);

plot(Pii_dBm(1:4000),Po_real_dBm(1:4000),'r-',Pii_dBm,Po_linear_dBm,'b-
',Pii_dBm,Po_intermod_dBm,'g-')
grid on
ylabel('Potencia de saida (dBm)')
xlabel('Potencia de entrada (dBm)')
pause


% Calculo de P1dB e IP3

V_1dB_in=sqrt(abs(((a1/(10^(1/20)))-a1)*(4/(3*a3))));
P_1dB_in=0.5*V_1dB_in.^2;
P_1dB_in=10*log10(P_1dB_in/1e-3)

V_1dB_out=a1.*V_1dB_in+3/4*a3.*V_1dB_in.^3;
P_1dB_out=0.5*V_1dB_out^2;
P_1dB_out=10*log10(P_1dB_out/1e-3)

ponto_intermodulacao_saida=abs((2*a1^3)/(3*a3));
ponto_intermodulacao_saida_dBm=10*log10(ponto_intermodulacao_saida/1e-3);
IP3_out=ponto_intermodulacao_saida_dBm

ponto_intermodulacao_entrada_tensao=sqrt(ponto_intermodulacao_saida/(0.5*a1^2)
);
ponto_intermodulacao_entrada_pot=0.5*ponto_intermodulacao_entrada_tensao^2;
ponto_intermodulacao_entrada_dBm=10*log10(ponto_intermodulacao_entrada_pot/1e-
3);
IP3_in=ponto_intermodulacao_entrada_dBm


% Calculo do ganho

ganho=a1+(3/4)*a3.*Vi.^2;
```

```
ganho_dB=20*log10(ganho)

plot(P_in_dBm,ganho_dB,'r-')
grid on
ylabel('Ganho (dB)')
xlabel('Potencia Entrada (dBm)')
----------fim da listagem do scritp de referência---------------------------
```

A.4 horner

```
--------------Listagem do algoritmo de Horner em MATLAB ----------------------
% Método de Horner
% entradas: grau do polinómio, coeficientes do polinómio pela ordem inversa no
vector C e abcissas da funcao x .
% saidas: valor da função dada pelo polinómio nos pontos das abcissas
%-------------------------------------------------------------------
% Autor: Joana Azevedo Braga  Data:21-04-2010

function resultado = horner(grau,C,x)

%grau = 3;
resultado = C(grau);

for k=grau:-1:2

    resultado = resultado*x + C(k-1);

end
-------------------------------fim da listagem -------------------------------
```

A.5 spline_cubico

```
-----Listagem de algoritmo de interpolação de spline cúbico com recurso a
filtros digitais ------
function S = spline_cubico( vector_amostra_ordenadas, vector_amostra_abcissas,
delta_abcissa, N )

%Input  - vector_amostra_abcissas is the 1xn abscissa sample vector
%       - vector_amostra_ordenadas is the 1xn ordinate sample vector
%       - delta_abcissas is the distance from the sample vector abscissa to
the new abscissa were we want
%       to interpolate
%       - N number of points from the sample vector
%Output - S: vector das ordenadas, resultante da interpolação por spline
cúbico. Ordinate vector resulting from cubic spline interpolation
% ------------------------------------------------------------------------
-------------------------------------------------------------
%
% Author: Joana Azevedo Braga                       Date:29-04-2010
%

z1 = -2+sqrt(3);

erro = 1e-6; % nível de precisão

k0 = log(erro) / log( abs(z1));

soma = 0; %% nota vector_amostra_ordenadas(0) = 0

for k = 1:1:k0
```

```
    soma = vector_amostra_ordenadas(k)*(z1)^(k) + soma;
end

c0_mais_zero = soma;


c0_mais(1) = vector_amostra_ordenadas(1) + z1*c0_mais_zero;

for k=2:1:N-1

    c0_mais(k) = vector_amostra_ordenadas(k) + z1 * c0_mais(k-1);

end



c0_menos(N-1) = (z1/(1-z1^2))*( c0_mais(N-1) + z1* c0_mais(N-2));


for k=N-2:-1:1

    c0_menos(k) = z1 * (c0_menos(k+1) - c0_mais(k));
end


c0_menos_zero = z1 * (c0_menos(1) - c0_mais_zero);


%C = [ 0.6666666666 0 -1 0.5 0 0 ];
C = [ 0.6666666666 0 -1 0.5 ];

for k=1:1:N-2


vector_abcissas_interpolacao = ((0.001-delta_abcissa)*k  /55.556 );  %0.0001

S(k) = (horner(3,C,vector_abcissas_interpolacao-1.047)*  -6*c0_menos(k) ); %-
vector_amostra_ordenadas(n)

vector_amostra_abcissas_aux(k) = vector_amostra_abcissas(k);
vector_amostra_ordenadas_aux(k) = vector_amostra_ordenadas(k);
S_aux(k) = S(k);

end
------------------fim de listagem-----------------------------------------------
------
```

A.6 calculo_P1dB_IP3_ganho para FPGA

```
---------inicio da listagem MATLAB do script de Algoritmo para FPGA ----------
function calculo_P1dB_IP3_ganho

format long

Vi=[0.001  0.002  0.003  0.004  0.005  0.006  0.007  0.008  0.009  0.010  0.011  0.012
...
    0.013  0.014  0.015  0.016  0.017  0.018  0.019  0.020  0.021  0.022  0.023  0.024
...
    0.025 0.026 0.027 0.028 0.029 0.030 0.031 0.032 0.033 0.034 0.035 0.036];

Vo=[0.001  0.002  0.003  0.004  0.005  0.006  0.007  0.008  0.009  0.010  0.011  0.012
...
    0.013  0.014  0.015  0.016  0.017  0.018  0.0188  0.0197  0.0205  0.0212  0.0220
0.0227 ...
    0.0235  0.0241  0.0247  0.0253  0.0259  0.0265  0.0270  0.0275  0.0280  0.0284
0.0287 0.0290];

P_in=0.5*Vi.^2;
P_in_dBm=10*log10(P_in/1e-3);
P_out=0.5*Vo.^2;
P_out_dBm=10*log10(P_out/1e-3);


% Obtencao do polinomio de 3ª ordem que melhor aproxima a funcao
% de transferencia Vin vs Vout do LNA.

[a,s]=polyfit(Vi,Vo,3);

a0=a(4)
a1=a(3)
a2=a(2)
a3=a(1)
```

```
Vy=a0+a1.*Vi+a2.*Vi.^2+a3.*Vi.^3;

plot(Vi,Vo,'r-*',Vi,Vy,'b')
grid on
ylabel('Tensao Saida (V)')
xlabel('Tensao Entrada (V)')
pause


% Representacao grafica de P1dB e IP3

num=length(Vi);
Vi_min=Vi(1);
Vi_max=Vi(num);

Vii=(Vi_min:0.00001:3*Vi_max);
aux=length(Vii);

Vy=a0+a1.*Vii+a2.*Vii.^2+a3.*Vii.^3;

Pii_dBm=10*log10(0.5*Vii.^2/1e-3);

Po_linear=0.5*a1.^2.*Vii.^2;
Po_linear_dBm=10*log10(Po_linear/1e-3);

Po_real=0.5*(a1.*Vii+3/4*a3.*Vii.^3).^2;
Po_real_dBm=10*log10(Po_real/1e-3);

Po_intermod=(9/32)*a3.^2.*Vii.^6;
Po_intermod_dBm=10*log10(Po_intermod/1e-3);

plot(Pii_dBm(1:4000),Po_real_dBm(1:4000),'r-',Pii_dBm,Po_linear_dBm,'b-
',Pii_dBm,Po_intermod_dBm,'g-')
grid on
ylabel('Potencia de saida (dBm)')
xlabel('Potencia de entrada (dBm)')
pause
```

```
% Calculo de P1dB e IP3

V_1dB_in=sqrt(abs(((a1/(10^(1/20)))-a1)*(4/(3*a3))));
P_1dB_in=0.5*V_1dB_in.^2;
P_1dB_in=10*log10(P_1dB_in/1e-3)

V_1dB_out=a1.*V_1dB_in+3/4*a3.*V_1dB_in.^3;
P_1dB_out=0.5*V_1dB_out^2;
P_1dB_out=10*log10(P_1dB_out/1e-3)

ponto_intermodulacao_saida=abs((2*a1^3)/(3*a3));
ponto_intermodulacao_saida_dBm=10*log10(ponto_intermodulacao_saida/1e-3);
IP3_out=ponto_intermodulacao_saida_dBm

ponto_intermodulacao_entrada_tensao=sqrt(ponto_intermodulacao_saida/(0.5*a1^2)
);
ponto_intermodulacao_entrada_pot=0.5*ponto_intermodulacao_entrada_tensao^2;
ponto_intermodulacao_entrada_dBm=10*log10(ponto_intermodulacao_entrada_pot/1e-
3);
IP3_in=ponto_intermodulacao_entrada_dBm


% Calculo do ganho

ganho=a1+(3/4)*a3.*Vi.^2;
ganho_dB=20*log10(ganho)

plot(P_in_dBm,ganho_dB,'r-')
grid on
ylabel('Ganho (dB)')
xlabel('Potencia Entrada (dBm)')

% Cálculo das diferenças do ponto de P1dB de referência para P1dB calculado
% pela Spline implementada com filtro digital

P_1_dB_Spline = P1dB_Spline(Vi,Vo,P_in_dBm, P_out_dBm); % chamada do calculo
da spline
```

```
delta_P1db = P_1dB_in - P_1_dB_Spline(1)

% com erro de 1%
P_1_dB_Spline1 = P1dB_Spline1(Vi,Vo,P_in_dBm, P_out_dBm); % chamada do calculo
da spline
delta_P1db1 = P_1dB_in - P_1_dB_Spline1(1)

% com erro de 5%
P_1_dB_Spline5 = P1dB_Spline5(Vi,Vo,P_in_dBm, P_out_dBm); % chamada do calculo
da spline
delta_P1db5 = P_1dB_in - P_1_dB_Spline5(1)

% ajuste dos coeficientes

while ( abs(delta_P1db) > 0.02 )

    if abs(delta_P1db1) > 0.02
        if abs(delta_P1db5) > 0.02
            if delta_P1db5 < 0
                a1 = a1 - a1*0.01
               % a3 = a3 + a3*0.01
                V_1dB_in_final=sqrt(abs(((a1/(10^(1/20)))-a1)*(4/(3*a3))));
                P_1dB_in_final=0.5*V_1dB_in_final.^2;
                P_1dB_in_final=10*log10(P_1dB_in_final/1e-3)
                delta_P1db = P_1dB_in_final - P_1_dB_Spline(1)
            else
               if delta_P1db5 > 0
                a1 = a1 + a1*0.01
                %a3 = a3 - a3*0.01
                V_1dB_in_final=sqrt(abs(((a1/(10^(1/20)))-a1)*(4/(3*a3))));
                P_1dB_in_final=0.5*V_1dB_in_final.^2;
                P_1dB_in_final=10*log10(P_1dB_in_final/1e-3)
                delta_P1db = P_1dB_in - P_1_dB_Spline(1)
               end
            end
        else
               if delta_P1db1 < 0
               a1 = a1 - a1*0.01
```

```
                %a3 = a3 + a3*0.01
                V_1dB_in_final=sqrt(abs(((a1/(10^(1/20)))-a1)*(4/(3*a3))));
                P_1dB_in_final=0.5*V_1dB_in_final.^2;
                P_1dB_in_final=10*log10(P_1dB_in_final/1e-3)
                delta_P1db = P_1dB_in - P_1_dB_Spline(1)
                else
                   if delta_P1db1 > 0
                    a1 = a1 + a1*0.01
                 %   a3 = a3 - a3*0.01
                    V_1dB_in_final=sqrt(abs(((a1/(10^(1/20)))-a1)*(4/(3*a3))));
                    P_1dB_in_final=0.5*V_1dB_in_final.^2;
                    P_1dB_in_final=10*log10(P_1dB_in_final/1e-3)
                    delta_P1db = P_1dB_in - P_1_dB_Spline(1)
                   end
                end
        end
    else
        if delta_P1db < 0
                a1 = a1 - a1*0.01
                %a3 = a3 + a3*0.01
                V_1dB_in_final=sqrt(abs(((a1/(10^(1/20)))-a1)*(4/(3*a3))));
                P_1dB_in_final=0.5*V_1dB_in_final.^2;
                P_1dB_in_final=10*log10(P_1dB_in_final/1e-3)
                delta_P1db = P_1dB_in - P_1_dB_Spline(1)
        else
                   if delta_P1db > 0
                    a1 = a1 + a1*0.01
                 %   a3 = a3 - a3*0.01
                    V_1dB_in_final=sqrt(abs(((a1/(10^(1/20)))-a1)*(4/(3*a3))));
                    P_1dB_in_final=0.5*V_1dB_in_final.^2;
                    P_1dB_in_final=10*log10(P_1dB_in_final/1e-3)
                    delta_P1db = P_1dB_in - P_1_dB_Spline(1)
                   end
        end


    end
end

  % Cálculo final do ponto de compressão de 1dB e de IIP3 com base nos novos
     % coeficientes
```

```
    V_1dB_in_final=sqrt(abs(((a1/(10^(1/20)))-a1)*(4/(3*a3))));
    P_1dB_in_final=0.5*V_1dB_in_final.^2;
    P_1dB_in_final=10*log10(P_1dB_in_final/1e-3)

    V_1dB_out_final=a1.*V_1dB_in+3/4*a3.*V_1dB_in.^3;
    P_1dB_out_final=0.5*V_1dB_out_final^2;
    P_1dB_out_final=10*log10(P_1dB_out_final/1e-3)

    ponto_intermodulacao_saida_final=abs((2*a1^3)/(3*a3));

ponto_intermodulacao_saida_dBm_final=10*log10(ponto_intermodulacao_saida_final
/1e-3);
    IP3_out_final=ponto_intermodulacao_saida_dBm_final


ponto_intermodulacao_entrada_tensao_final=sqrt(ponto_intermodulacao_saida_fina
l/(0.5*a1^2));

ponto_intermodulacao_entrada_pot_final=0.5*ponto_intermodulacao_entrada_tensao
_final^2;

ponto_intermodulacao_entrada_dBm_final=10*log10(ponto_intermodulacao_entrada_p
ot_final/1e-3);
    IP3_in_final=ponto_intermodulacao_entrada_dBm_final



delta_P1db = P_1dB_in_final - P_1_dB_Spline(1)

% com erro de 1%
delta_P1db1 = P_1dB_in_final - P_1_dB_Spline1(1)

% com erro de 5%
delta_P1db5 = P_1dB_in_final - P_1_dB_Spline5(1)
end
  ------------------------------ fim da listagem ----------------------------
```

A.7 P1dB_Spline

```
-----------------------------------------P1dB_SPline-----------------------
---------------------------------
function [ P_1dB_in, P_1dB_out ] = P1dB_Spline(Vi,Vo, P_in_dBm, P_out_dBm)

% Inputs : Vi - sample 36 point vector abscisses
%        : Vo - sample 36 point vector ordinates
% Outputs : P_1dB_in - Compression point 1dB abscisse
%         : P_1dB_out - Compression point 1dB ordinate
%-------------------------------------------------------
% Author : Joana Azevedo Braga    Date : 29-04-2010

% Cálculo dos intervalos do Spline

x = 10*log10((0.5*Vi.^2)/1e-3);

for n=4:1:34
    aux_x(n) = x(n);
end

% Cálculo das ordenadas
y = spline_cubico(P_out_dBm,P_in_dBm,0,36);

for n=4:1:34
    aux_y(n) = y(n);
```

```
end

% Calculo do ganho

for n=1:1:36
g(n) = 20*log10(Vo(n)/Vi(n));
end

for i=1:1:36
    g_pot(i) = P_out_dBm(i) - P_in_dBm(i);
end

for i=1:1:36;

delta(i) = abs( g_pot(i) - g(i));
end

for j=1:1:36
if ( delta(j) > 0.0041*g(j) ) % precisão para o grau de linearidade do ganho

   A = zona_linear(j, P_in_dBm, P_out_dBm, g);
   for n=1:1:j
       P_in_dBm_linear(n) = P_in_dBm(n);
   end

   y1 = [(P_in_dBm(2) - P_in_dBm(1))/((A(2)-A(1))) 0]

   plot(aux_x,aux_y,'r-*', P_in_dBm_linear,A,'b',aux_x,polyval(y1,aux_x));
   grid on


   diferenca = polyval(y1,aux_x)-aux_y;

   m=1;
   while ( diferenca(m) <= 1.00-0.00041 && m < 35 )

   m=m+1;

   end
   P_1dB_in = P_in_dBm(m)
   P_1dB_out = P_out_dBm(m)
   break
end
end
-------------------fim da listagem -----------------------------------------
```

A.8 P1dB_Spline

```
----------------------------------------
function [ P_1dB_in, P_1dB_out ] = P1dB_Spline1(Vi1,Vo1,P_in_dBm1,P_out_dBm1)

% Inputs : Vi1 - sample 36 point vector abscisses
%        : Vo1 - sample 36 point vector ordinates
% Outputs : P_1dB_in - Compression point 1dB abscisse
%         : P_1dB_out - Compression point 1dB ordinate
%-------------------------------------------------------
% Author : Joana Azevedo Braga    Date : 29-04-2010


% erro aleatório de 1%
Vi = (Vi1*1.005)*(1-rand(1,1)/100);
Vo = (Vo1*1.005)*(1-rand(1,1)/100);

% erro aleatório de 1%
```

```
P_in_dBm = (P_in_dBm1*1.005)*(1-rand(1,1)/100);
P_out_dBm = (P_out_dBm1*1.005)*(1-rand(1,1)/100);


% Cálculo dos intervalos do Spline

x = 10*log10((0.5*Vi.^2)/1e-3);

for n=4:1:34
    aux_x(n) = x(n);
end

% Cálculo das ordenadas
y = spline_cubico(P_out_dBm,P_in_dBm,0,36);

for n=4:1:34
    aux_y(n) = y(n);
end

% Calculo do ganho

for n=1:1:36
g(n) = 20*log10(Vo(n)/Vi(n));
end

for i=1:1:36
    g_pot(i) = P_out_dBm(i) - P_in_dBm(i);
end

for i=1:1:36;

delta(i) = abs( g_pot(i) - g(i));
end

for j=4:1:36    % é assumido que nos primeiros 3 pontos o comportamento da
curva é não linear
if ( delta(j) > 0.0041*g(j) )

   A = zona_linear(j, P_in_dBm, P_out_dBm, g);
   for n=4:1:j
       P_in_dBm_linear(n) = P_in_dBm(n);
   end

   y1 = [(P_in_dBm(2) - P_in_dBm(1))/((A(2)-A(1))) 0]

   plot(aux_x,aux_y,'r-*', P_in_dBm_linear,A,'b',aux_x,polyval(y1,aux_x));
   grid on

   diferenca = polyval(y1,aux_x)-aux_y;

   m=1;
   while ( diferenca(m) <= 1.00-0.00041 && m < 35 )

   m=m+1;

   end
   P_1dB_in = P_in_dBm(m)
   P_1dB_out = P_out_dBm(m)
   break
end
end


--------------fim da listagem----------------------------------------------------
```

A.9 P1dB_Spline5

```
-------------------------------------P1_dB - 5% de erro aleatório-------------
------------------------------
function [ P_1dB_in, P_1dB_out ] = P1dB_Spline5(Vi1,Vo1,P_in_dBm1,P_out_dBm1)

% Inputs : Vi1 - sample 36 point vector abscisses
%        : Vo1 - sample 36 point vector ordinates
% Outputs : P_1dB_in - Compression point 1dB abscisse
%         : P_1dB_out - Compression point 1dB ordinate
%-------------------------------------------------------
% Author : Joana Azevedo Braga    Date : 29-04-2010

% erro aleatório de 5%
Vi = (Vi1*1.025)*(1-rand(1,1)/100);
Vo = (Vo1*1.025)*(1-rand(1,1)/100);

P_in_dBm = (P_in_dBm1*1.025)*(1-rand(1,1)/100);
P_out_dBm = (P_out_dBm1*1.025)*(1-rand(1,1)/100);

% Cálculo dos intervalos do Spline

x = 10*log10((0.5*Vi.^2)/1e-3);

for n=4:1:34
    aux_x(n) = x(n);
end

% Cálculo das ordenadas
y = spline_cubico(P_out_dBm,P_in_dBm,0,36);

for n=4:1:34
    aux_y(n) = y(n);
end

% Calculo do ganho

for n=1:1:36
g(n) = 20*log10(Vo(n)/Vi(n));
end

for i=1:1:36
    g_pot(i) = P_out_dBm(i) - P_in_dBm(i);
end

for i=1:1:36;

delta(i) = abs( g_pot(i) - g(i));
end

for j=4:1:36    % assume-se que os 3 primeiros pontos representam uma
caracteristica nao linear
if ( delta(j) > 0.0041*g(j) )

   A = zona_linear(j, P_in_dBm, P_out_dBm, g);
   for n=4:1:j
       P_in_dBm_linear(n) = P_in_dBm(n);
   end

   y1 = [(P_in_dBm(2) - P_in_dBm(1))/((A(2)-A(1))) 0]

   plot(aux_x,aux_y,'r-*', P_in_dBm_linear,A,'b',aux_x,polyval(y1,aux_x));
   grid on

   diferenca = polyval(y1,aux_x)-aux_y;

   m=1;
   while ( diferenca(m) <= 1.00-0.00041 && m < 35 )

   m=m+1;
```

```
   end
   P_1dB_in = P_in_dBm(m)
   P_1dB_out = P_out_dBm(m)
   break
end
end
------------------fim de listagem ------------------------------------------
```

A.10 Spline Matricial - listagem de "csfit" retirada do livro [16]

```
-------------------------spline matricial-------------------------------------
---------------------------------
function S=csfit(X,Y,dx0,dxn)

%Input  - X is the 1xn abscissa vector
%       - Y is the 1xn ordinate vector
%       - dx0 = S'(x0) first derivative boundary condition
%       - dxn = S'(xn) first derivative boundary condition
%Output - S: rows of S are coefficients, in descending
%         order, for the cubic interpolants

N=length(X)-1;
H=diff(X);
D=diff(Y)./H;
A=H(2:N-1);
B=2*(H(1:N-1)+H(2:N));
C=H(2:N);
U=6*diff(D);

%Clamped spline endpoint constraints
B(1)=B(1)-H(1)/2;
U(1)=U(1)-3*(D(1)-dx0);
B(N-1)=B(N-1)-H(N)/2;
U(N-1)=U(N-1)-3*(dxn-D(N));

for k=2:N-1
    temp=A(k-1)/B(k-1);
    B(k)=B(k)-temp*C(k-1);
    U(k)=U(k)-temp*U(k-1);
end

M(N)=U(N-1)/B(N-1);

for k=N-2:-1:1
    M(k+1)=(U(k)-C(k)*M(k+2))/B(k);
end

M(1)=3*(D(1)-dx0)/H(1)-M(2)/2;
M(N+1)=3*(dxn-D(N))/H(N)-M(N)/2;

for k=0:N-1
    S(k+1,1)=(M(k+2)-M(k+1))/(6*H(k+1));
    S(k+1,2)=M(k+1)/2;
    S(k+1,3)=D(k+1)-H(k+1)*(2*M(k+1)+M(k+2))/6;
    S(k+1,4)=Y(k+1);
end
-----------------------------fim da listagem------------------------------[16]
```

A.11 Cálculo de P1dB com recurso a spline matricial

```
--------------cálculo de P1db com recurso a spline matricial -----------------
------------------------
format long

Vi=[0.001 0.002 0.003 0.004 0.005 0.006 0.007 0.008 0.009 0.010 0.011 0.012
...
    0.013 0.014 0.015 0.016 0.017 0.018 0.019 0.020 0.021 0.022 0.023 0.024
...
    0.025 0.026 0.027 0.028 0.029 0.030 0.031 0.032 0.033 0.034 0.035 0.036];

Vo=[0.001 0.002 0.003 0.004 0.005 0.006 0.007 0.008 0.009 0.010 0.011 0.012
...
    0.013 0.014 0.015 0.016 0.017 0.018 0.0188 0.0197 0.0205 0.0212 0.0220
0.0227 ...
    0.0235 0.0241 0.0247 0.0253 0.0259 0.0265 0.0270 0.0275 0.0280 0.0284
0.0287 0.0290];

P_in=0.5*Vi.^2;
P_in_dBm=10*log10(P_in/1e-3);
P_out=0.5*Vo.^2;
P_out_dBm=10*log10(P_out/1e-3);

% Cálculo da Matriz de Splines
S = csfit(Vi,Vo,0,0);

% Definição dos polinómios de Spline
p1 =S(1,1:1:4);
p2 =S(2,1:1:4);
p3 =S(3,1:1:4);
p4 =S(4,1:1:4);
p5 =S(5,1:1:4);
p6=S(6,1:1:4);
p7=S(7,1:1:4);
p8=S(8,1:1:4);
p9=S(9,1:1:4);
p10=S(10,1:1:4);
p11=S(11,1:1:4);
p12=S(12,1:1:4);
p13=S(13,1:1:4);
p14=S(14,1:1:4);
p15=S(15,1:1:4);
p16=S(16,1:1:4);
p17=S(17,1:1:4);
p18=S(18,1:1:4);
p19=S(19,1:1:4);
p20=S(20,1:1:4);
p21=S(21,1:1:4);
p22=S(22,1:1:4);
p23=S(23,1:1:4);
p24=S(24,1:1:4);
p25=S(25,1:1:4);
p26=S(26,1:1:4);
p27=S(27,1:1:4);
p28=S(28,1:1:4);
p29=S(29,1:1:4);
p30=S(30,1:1:4);
p31=S(31,1:1:4);
p32=S(32,1:1:4);
p33=S(33,1:1:4);
p34=S(34,1:1:4);
p35=S(35,1:1:4);

% Definição dos intervalos de cada polinómio do spline
x1=[Vi(1):0.1:Vi(2)]; y1=polyval(p1,x1-Vi(1));
x2=[Vi(2):0.1:Vi(3)]; y2=polyval(p2,x2-Vi(2));
x3=[Vi(3):0.1:Vi(4)]; y3=polyval(p3,x3-Vi(3));
x4=[Vi(4):0.1:Vi(5)]; y4=polyval(p4,x4-Vi(4));
x5=[Vi(5):0.1:Vi(6)]; y5=polyval(p5,x5-Vi(5));
```

```
x6=[Vi(6):0.1:Vi(7)]; y6=polyval(p6,x6-Vi(6));
x7=[Vi(7):0.1:Vi(8)]; y7=polyval(p7,x7-Vi(7));
x8=[Vi(8):0.1:Vi(9)]; y8=polyval(p8,x8-Vi(8));
x9=[Vi(9):0.1:Vi(10)]; y9=polyval(p9,x9-Vi(9));
x10=[Vi(10):0.1:Vi(11)]; y10=polyval(p10,x10-Vi(10));
x11=[Vi(11):0.1:Vi(12)]; y11=polyval(p11,x11-Vi(11));
x12=[Vi(12):0.1:Vi(13)]; y12=polyval(p12,x12-Vi(12));
x13=[Vi(13):0.1:Vi(14)]; y13=polyval(p13,x13-Vi(13));
x14=[Vi(14):0.1:Vi(15)]; y14=polyval(p14,x14-Vi(14));
x15=[Vi(15):0.1:Vi(16)]; y15=polyval(p15,x15-Vi(15));
x16=[Vi(16):0.1:Vi(17)]; y16=polyval(p16,x16-Vi(16));
x17=[Vi(17):0.1:Vi(18)]; y17=polyval(p17,x17-Vi(17));
x18=[Vi(18):0.1:Vi(19)]; y18=polyval(p18,x18-Vi(18));
x19=[Vi(19):0.1:Vi(20)]; y19=polyval(p19,x19-Vi(19));
x20=[Vi(20):0.1:Vi(21)]; y20=polyval(p20,x20-Vi(20));
x21=[Vi(21):0.1:Vi(22)]; y21=polyval(p21,x21-Vi(21));
x22=[Vi(22):0.1:Vi(23)]; y22=polyval(p22,x22-Vi(22));
x23=[Vi(23):0.1:Vi(24)]; y23=polyval(p23,x23-Vi(23));
x24=[Vi(24):0.1:Vi(25)];  y24=polyval(p24,x24-Vi(24));
x25=[Vi(25):0.1:Vi(26)]; y25=polyval(p25,x25-Vi(26));
x26=[Vi(26):0.1:Vi(27)]; y26=polyval(p26,x26-Vi(26));
x27=[Vi(27):0.1:Vi(28)]; y27=polyval(p27,x27-Vi(27));
x28=[Vi(28):0.1:Vi(29)]; y28=polyval(p28,x28-Vi(28));
x29=[Vi(29):0.1:Vi(30)]; y29=polyval(p29,x29-Vi(29));
x30=[Vi(30):0.1:Vi(31)]; y30=polyval(p30,x30-Vi(30));
x31=[Vi(31):0.1:Vi(32)]; y31=polyval(p31,x31-Vi(31));
x32=[Vi(32):0.1:Vi(33)]; y32=polyval(p32,x32-Vi(32));
x33=[Vi(33):0.1:Vi(34)]; y33=polyval(p33,x33-Vi(33));
x34=[Vi(34):0.1:Vi(35)]; y34=polyval(p34,x34-Vi(34));
x35=[Vi(35):0.1:Vi(36)]; y35=polyval(p35,x35-Vi(35));

% Desenho da curva no ecran
plot(x1,y1,x2,y2,x3,y3,x4,y4,x5,y5,x6,y6,x7,y7,x8,y8,x9,y9,x10,y10,x11,y11,x12
,y12,x13,y13,x14,y14,x15,y15,x16,y16,x17,y17,x18,y18,x19,y19,x20,y20,x21,y21,x
22,y22,x23,y23,x24,y24,x25,y25,x26,y26,x27,y27,x28,y28,x29,y29,x30,y30,x31,y31
,x32,y32,x33,y33,x34,y34,x35,y35);
pause



% Calculo do ganho

for n=1:1:36
g(n) = 20*log10(Vo(n)/Vi(n));
end



for i=1:1:36
    g_pot(i) = P_out_dBm(i) - P_in_dBm(i);
end

for i=1:1:36;

delta(i) = abs( g_pot(i) - g(i));
end
for j=1:1:36
if ( delta(j) > 0.0001*g(j) )

   A = zona_linear(j, P_in_dBm, P_out_dBm, g);
   for n=1:1:j
       P_in_dBm_linear(n) = P_in_dBm(n);
   end
   for n=j:1:36

    P_in_dBm_nao_linear(n) = P_in_dBm(n);
    P_out_dBm_nao_linear(n) = P_out_dBm(n);
   end
```

```
   plot(P_in_dBm_linear, A,'b',P_in_dBm_nao_linear,  P_out_dBm_nao_linear,'g')
   grid on
   j



   break % chamar outro script para calcular a recta linear funcao(j)

end
end
-----------------------fim da listagem ------------------------------------
```

Anexo B – Listagens de Verilog

B.1 Conversão de escalas

```
--------início da listagem conversão de escalas -----------------------------
`timescale 1ns/100ps

module conversao_escalas ( X, Y, Clk_Sys, reset);

parameter Max_bit = 27;
parameter bits_fraccao = 15;
parameter Max_ADC_bit = 10;

input Clk_Sys, reset;

output [Max_bit-1:0] X,Y;

//wire [9:0] alfa, beta;          // comment this line if alfa and beta are to
be read from memory
// the following lines are to be uncommented if alfa and beta are to be read
from memory
 reg [Max_ADC_bit-1:0] alfa [35:0];
 reg[Max_ADC_bit-1:0] beta [35:0];
 reg [Max_bit -1:0] aux3, aux4, aux5;

reg [Max_ADC_bit-1:0] teste_alfa, teste_beta;

reg [Max_bit-1:0] X,Y, X_aux, Y_aux;
reg signal;
reg [bits_fraccao-1:0] fraccao;
reg [2*Max_bit:0] aux_1, aux_2;
wire [Max_bit -1:0] X_wire, Y_wire;

integer n, fd2, fd3, fd4;

parameter factor_escala = 27'b000000000000000000001001001110; //0.018(b10) ~
0.018005
parameter aux_inteiro = 11'd0;
parameter positivo = 1'b0;
parameter negativo = 1'b1;

assign X_wire = X_aux;
assign Y_wire = Y_aux;
 // the following lines are to be uncommented if alfa and beta are to be read
from memory


always @ (posedge Clk_Sys) begin

if (reset == 1'b1)
begin
X <= 27'd0;
Y <= 27'd0;
X_aux <= 27'd0;
Y_aux <= 27'd0;
signal <= 1'd0;
fraccao <= 15'd0;
aux_1 <= 41'd0;
aux_2 <= 41'd0;
teste_alfa <= 10'd0;
teste_beta <= 10'd0;
end

else begin
```

```
signal <= positivo;

$readmemb("../alfa1.dat", alfa);
$readmemb("../beta1.dat", beta);



for(n=0;n<36;n=n+1)
begin



$display("alfa[%d] = %b", n[5:0], alfa[n]);
$display("beta[%d] = %b", n[5:0], beta[n]);

teste_alfa = alfa[n];


// escrita de dados para o polinómio de spline Cúbico


$display(" teste_alfa = %b, teste_beta = %b", teste_alfa, teste_beta );

teste_beta = beta[n];



// escrita de dados para os filtros digitais que implementam os coeficientes a
multiplicar pelo polinómio de spline cúbico
fd2 = $fopen("../Y_teste1.dat", "a+");
                   $fwrite(fd2, "\n %b%b%b%b", signal, 1'b0, teste_beta,
15'b0);
                $fclose(fd2);

// escrita de dados para o polinómio de spline Cúbico
fd3 = $fopen("../X_teste1.dat", "a+");
                   $fwrite(fd3, "\n %b%b%b%b", signal, 1'b0, teste_alfa,
15'b0);
                $fclose(fd3);



end



end
end
endmodule
```

----------fim da listagem----------

B.2 Filtro Digital baseado na regra de Horner

```
------------------inicio da listagem de horner_filter3.v----------------------
`timescale 1ns/100ps

module horner_filter3(B,X,A,P,reset,clock);

input [26:0] B,A,X;
input reset, clock;
output [26:0] P;

wire [26:0] B,A,X, aux4, aux5;
```

```
wire [53:0] aux3, aux2;
reg [26:0] aux_p, aux_p1;
reg [26:0] P;



assign aux2 = A[25:0] * X[25:0]; // Ax
assign aux2_sinal = A[26] ^ X[26];
assign aux4 = {aux2_sinal, aux2[41:14]};
assign aux5 = {B[26],B[25:0]};

always @(posedge clock)                 //Ax + B
begin
if (reset == 1'b1)
begin
P <= 27'd0;
aux_p <= 27'd0;
aux_p1 <= 27'd0;
end
else
begin
if((aux4[25:0] > aux5[25:0]) && (aux4[26] == 1'b1) && (aux5[26] == 1'b0) )
 begin
 aux_p[26:0] <= aux4[25:0] - aux5[25:0];
P[25:0] <= aux_p[25:0];
 P[26] <= aux4[26];
end
else
begin

  if((aux4[25:0] > aux5[25:0]) && (aux4[26] == 1'b0) && (aux5[26] == 1'b1) )
  begin
  aux_p[26:0] <= aux4[25:0] - aux5[25:0];
  P[25:0] <= aux_p[25:0];

 // sinal
 P[26] <= aux4[26];
end
else
  begin

    if ((aux4[25:0] < aux5[25:0]) && (aux4[26] == 1'b0) && (aux5[26] == 1'b1)
)
             begin
             aux_p[25:0] <= aux5[25:0] - aux4[25:0];
             // converter complemento para 2 para unsigned bits
             //aux_p1[20:0] <= aux_p[20:0] - 21'd1;
             //P[20:0] <= ! aux_p1[20:0];
             P[25:0] <= aux_p[25:0];
             // sinal
             P[26] <= aux5[26];
             end
       else
       begin

         if ((aux4[25:0] < aux5[25:0]) && (aux4[26] == 1'b1) && (aux5[26] ==
1'b0) )
             begin
             aux_p[25:0] <= aux5[25:0] - aux4[25:0];

             P[25:0] <= aux_p[25:0];
             // sinal
             P[26] <= aux5[26];
             end
       else
       begin
       if ( (aux4[26] == 1'b1) && (aux5[26] == 1'b1) )
                   begin
```

```
                    aux_p[25:0] <= aux4[25:0] + aux5[25:0];

                    P <= {aux4[26],aux_p[25:0]};
                    end
      else
            begin
            if ( (aux4[26] == 1'b0) && (aux5[26] == 1'b0) )
                        begin
                        aux_p[25:0] <= aux4[25:0] + aux5[25:0];
                        P <= {aux4[26],aux_p[25:0]};
                        end
            end
      end
end
end
end
end

// duplo zero
if( P[25:0] == 26'd0 && P[26] == 1'b1 )
begin
P[26] <= 1'b0;
end


end
endmodule
-----------------------------------fim de listagem -----------------------------
```

B.3 Polinómio de Spline cúbica - polinomio2.v

```
----------------------inicio da listagem polinomio2.v ------------------------
`timescale 1ns/100ps

module polinomio2(P1,P2,P3,reset,Clk_Sys);

//input [20:0] X;
reg [26:0] X [35:0];
input reset,Clk_Sys;
output [26:0] P1;
output[26:0] P2;
output[26:0] P3;
wire [26:0] P1;
wire [26:0] P2;
wire [26:0] P3;

reg [26:0] P1_aux, P2_aux, P3_aux, ler_aux5;

integer index,n,fd1;

parameter C0 = 27'b000000000000101010101010101; // 0.6666
parameter C1 = 27'd0; // 0
parameter C2 = 27'b100000000001000000000000000; // -1
parameter C3 = 27'b000000000000100000000000000; // 0.5


// X[32]
horner_filter3 a(C0,X[32],C1,P1,reset,Clk_Sys);
horner_filter3 b(C2,X[32],P1,P2,reset,Clk_Sys);
horner_filter3 c(C3,X[32],P2,P3, reset,Clk_Sys);



// The following lines are to be uncommented if X is to be read from memory
and the port module must be changed to acomodate the difference
initial
 begin
$readmemb("../aux5.dat", X);


$display("X = %b", X[32]);

// escrita de dados para o polinómio de spline Cúbico

fd1 = $fopen("../P3_aux5.dat", "a+");
                    $fwrite(fd1, "\n %b", P3);
                 $fclose(fd1);







end
endmodule
----------------------------------fim da listagem ----------------------------
```

B.4 Módulo de teste do “polinómio2.v”

```
------------------------inicio da listagem do testbench de polinomio2---------
`timescale 1ns/100ps

module polinomio2_testbench;

reg reset, Clk_Sys;
//reg [20:0] X;
wire [26:0] P1;
wire [26:0] P2;
wire [26:0] P3;
wire[53:0] aux2;

always begin  // clock process

 Clk_Sys = 1'b0;
 #1
 Clk_Sys = 1'b1;
 #1
 #49
 Clk_Sys = 1'b0;
 #49
 Clk_Sys = 1'b0;
end

polinomio2 polinomio(P1,P2,P3,reset,Clk_Sys);


initial
 begin

 $monitor( $time, " P1 = %b, P2 = %b, P3 = %b, reset = %b, Clk_Sys = %b, aux2
= %b, aux4 = %b, ler_aux5_bloco1 = %b, ler_aux5_bloco2 = %b, ler_aux5_bloco3 =
%b ",
 P1,P2,P3, reset, Clk_Sys, polinomio2_testbench.polinomio.b.aux2,
polinomio2_testbench.polinomio.b.aux4, polinomio2_testbench.polinomio.a.X,
polinomio2_testbench.polinomio.b.X, polinomio2_testbench.polinomio.c.X );



reset = 1'b0;
#100
reset = 1'b1;
#100
reset = 1'b0;
#100


#950


   $stop;

  end
endmodule
----------------------------------------fim de listagem-------------------------
```

B.5 Módulo de Filtros para coeficientes em System Verilog

```
----inicio da listagem filtros_para_coeficientes2.sv-------------------------

`timescale 1ns/100ps
```

```
module filtros_para_coeficientes(clock); //Clk_Sys, reset



input clock;

integer k;

reg [26:0] vector_amostra_ordenadas[36:0];
reg [26:0] c0_mais[36:0], c0_menos[36:0]; // array de N elementos de 22 bits
reg [26:0] c0_mais_zero, c0_menos_zero;
//reg [25:0] soma;

// Nota: É necessário calcular o parâmetro soma em MATLAB, este depende do
valor das amostras em y, de k 0 e do erro ( nível de precisão desejado).
// z1 = -2+sqrt(3);
//erro = 1e-6; % nível de precisão
//k0 = ceil(log(erro) / log( abs(z1)));  Nota: para um erro de 1e-6, k0=11  ;
k0 = 8

parameter k0 = 27'b000000001011000000000000000; // k0 = 11
parameter z1 = 27'b100000000000010001001001100; // z1 =-0.267944 ~ -0.267949
parameter N = 22'd33; // numero de pontos do vector amostra ordenadas ex:
36(10)

initial
begin
reg [26:0] vector_amostra_ordenada [32:0];
reg [52:0] multiplicacao_res;
reg [26:0] mult_truncated;
reg [35:0] soma2 [32:0];
reg [27:0] aux_soma2;
reg [52:0] soma3 [32:0];
reg [26:0] c0_mais_zero;
reg [52:0] c0_mais_aux;
reg [26:0] c0_mais [32:0];
reg [26:0] c0_mais_truncated, vector_amostra_ordenada_aux, c0_mais_sinal_aux;
reg [26:0] aux_p, aux_p1;
```

```
reg [27:0] P;
reg [81:0] c0_menos_aux2;
reg [26:0] c0_mais_aux2, c0_mais_aux3, z1_aux;
reg [53:0] calc_1;
reg [26:0] calc_2;
reg [26:0] calc_3, aux_c0_menos_32, aux_c0_mais_k;
reg [53:0] c0_menos_k_aux;

           integer   index, k,l,fd1, fd2;

           begin

              $readmemb("../Y.dat", vector_amostra_ordenada);

                           soma3[0] = vector_amostra_ordenada[0];

              for(index = 0; index < 12; index = index + 1)
                      begin
              //$display("vector_amostra_ordenada[%d] = %b", index[5:0],
vector_amostra_ordenada[index]);
                      soma2[index] = vector_amostra_ordenada[index] *
z1^(21'd12) + soma3[0] ;
                      $display("\n soma2[%d] = %p", index[5:0], soma2[index]);

                      end

                      //multiplicacao_res = aux_soma2[26:0];
                      //mult_truncated = multiplicacao_res[26:0];

                      //soma3[index+1] = aux_soma2 + soma3[0];
                      //$display("\n soma2 = %p", soma2);
                      //$display("\n Apos somatorio: soma3[%d] = %b",
index[5:0], soma3[index]);
                      //$display("\n multiplicacao_res = %b",
multiplicacao_res);

                      end
```

```
                    //$display("\n mult_truncated = %b", mult_truncated); //
soma = mult_truncated

                     //c0_mais_zero = mult_truncated; // c0_mais_zero = soma
                     c0_mais_zero = soma2[10];

                     //for (k=2; k<=N-1; k=k+1)
                    //
                    //    c0_mais[k] <= vector_amostra_ordenadas[k] + z1 *
c0_mais[k-1];

                    c0_mais_aux = z1[25:0]*c0_mais_zero[25:0];

                    // resolução do sinal

                    if ( z1[26] == 1'b1 && c0_mais_zero[26] == 1'b0)
                    begin
                    c0_mais_aux[35] = 1'b1;
                    end
                    else begin
                                if ( z1[26] == 1'b1 && c0_mais_zero[26] ==
1'b1)
                                begin
                                c0_mais_aux[35] = 1'b0;
                                end
                           else begin
                                       if ( z1[26] == 1'b0 &&
c0_mais_zero[26] == 1'b1)
                                       begin
                                       c0_mais_aux[35] = 1'b1;
                                       end
                                       else begin
                                                   if ( z1[26] == 1'b0 &&
c0_mais_zero[26] == 1'b0)
                                                   begin
                                                   c0_mais_aux[35] = 1'b0;
```

```
                                        end
                                    end
                                end
                            end


                    c0_mais_truncated = c0_mais_aux[35:9];

                    $display("\n c0_mais_aux = %b, c0_mais_truncated = %b  ",
c0_mais_aux, c0_mais_truncated);

                    c0_mais[0] = c0_mais_truncated; //c0_mais_truncated;

                    $display("\n c0_mais[0] = %b  ", c0_mais[0]);

                    //sinal


                    for( k=0; k < 33; k = k+1 )
                    begin
                    // registos auxiliares
                    vector_amostra_ordenada_aux = vector_amostra_ordenada[k];
                    if ( k == 0)
                    begin
                    c0_mais_sinal_aux = 27'b100000000000000010010100000;
                    end
                    // somatório
                    //c0_mais[k] = vector_amostra_ordenada_aux[19:0] +
c0_mais_aux[19:0];
                    $display("\n c0_mais_sinal_aux = %b,
vector_amostra_ordenada_aux = %b  ", c0_mais_sinal_aux,
vector_amostra_ordenada_aux);
                    // sinal

                    if (c0_mais_sinal_aux == 27'b100000000000000000000000000 )
// caso de duplo 0 na notacao binaria escolhida
                    begin
```

```
                    c0_mais_sinal_aux = 27'b000000000000000000000;
                    end

                    if((vector_amostra_ordenada_aux[25:0] >
c0_mais_sinal_aux[25:0]) && (vector_amostra_ordenada_aux[26] == 1'b1) &&
(c0_mais_sinal_aux[26] == 1'b0) )
                          begin
                          aux_p[26:0] = c0_mais_sinal_aux[25:0] -
vector_amostra_ordenada_aux[25:0];
                          // converter complemento para 2 para unsigned bits
                          aux_p1[26:0] = aux_p[26:0] - 21'd1;
                          P[26:0] = ! aux_p1[26:0];
                          // sinal
                          P[27] = vector_amostra_ordenada_aux[26];
                          end
                    else
                          begin
                          if ((vector_amostra_ordenada_aux[25:0] <
c0_mais_sinal_aux[25:0]) && (vector_amostra_ordenada_aux[26] == 1'b1) &&
(c0_mais_sinal_aux[26] == 1'b0) )
                                begin
                                aux_p[26:0] = c0_mais_sinal_aux[25:0] -
vector_amostra_ordenada_aux[25:0];
                                // converter complemento para 2 para unsigned
bits
                                aux_p1[26:0] = aux_p[26:0] - 22'd1;
                                P[26:0] = ! aux_p1[26:0];
                                // sinal
                                P[27] = c0_mais_sinal_aux[26];
                                end
                          else
                          begin
                                if ((vector_amostra_ordenada_aux[25:0] >
c0_mais_sinal_aux[25:0]) && (vector_amostra_ordenada_aux[26] == 1'b0) &&
(c0_mais_sinal_aux[26] == 1'b1) )
                                      begin
                                      aux_p[26:0] =
vector_amostra_ordenada_aux[25:0] - c0_mais_sinal_aux[25:0];
```

```
                                        // converter complemento para 2 para
unsigned bits
                                        aux_p1[26:0] = aux_p[26:0] - 22'd1;
                                        P[26:0] = ! aux_p1[26:0];
                                        // sinal
                                        P[27] =
vector_amostra_ordenada_aux[26];
                                        end
                                  else
                                  begin
                                              if
((vector_amostra_ordenada_aux[25:0] < c0_mais_sinal_aux[25:0]) &&
(vector_amostra_ordenada_aux[26] == 1'b0) && (c0_mais_sinal_aux[26] == 1'b1) )
                                                    begin
                                                    aux_p[26:0] =
c0_mais_sinal_aux[25:0] - vector_amostra_ordenada_aux[25:0];
                                                    // converter complemento
para 2 para unsigned bits
                                                    aux_p1[26:0] =
aux_p[26:0] - 22'd1;
                                                    P[26:0] = ! aux_p1[26:0];
                                                    // sinal
                                                    P[27] =
c0_mais_sinal_aux[26];
                                                    end
                                              else
                                              begin
                                                    if (
(vector_amostra_ordenada_aux[26] == 1'b1) && (c0_mais_sinal_aux[26] == 1'b1) )
                                                    begin
                                                    aux_p[26:0] =
vector_amostra_ordenada_aux[25:0] + c0_mais_sinal_aux[25:0];

                                                    P =
{vector_amostra_ordenada_aux[26],aux_p};
                                                    end
                                                    else
                                                    begin
```

```
                                                        if (
(vector_amostra_ordenada_aux[26] == 1'b0) && (c0_mais_sinal_aux[26] == 1'b0) )
                                                        begin
                                                               aux_p[26:0]
= vector_amostra_ordenada_aux[25:0] + c0_mais_sinal_aux[25:0];
                                                               P =
{vector_amostra_ordenada_aux[26],aux_p};
                                                        end
                                                 end
                                          end
                                   end
                             end
                    end

                    c0_mais[k] = {P[27],P[26:1]};

                    if (c0_mais[k] == 27'b100000000000000000000000000 ) //
caso de duplo 0 na notacao binaria escolhida
                    begin
                    c0_mais[k] = 27'b000000000000000000000000000;
                    end

                    c0_mais_sinal_aux = c0_mais[k];

                    $display("\n c0_mais [%d] = %b, P = %b, aux_p = %b, aux_p1
= %b", k[5:0], c0_mais[k], P, aux_p, aux_p1);

                    end  // fim do ciclo for



                    //(z1/(1-(z1*z1))) = z1_aux

                    z1_aux = 27'b100000000000010010011100000; // z1_aux = -
0.288674895207275
```

```
                    //  *c0_menos[N-1] <= (z1/(1-(z1*z1)))*( c0_mais[N-1] +
z1* c0_mais[N-2]);

                    // registos auxiliares
                    c0_menos_aux2 = c0_menos[33-1];
                    c0_mais_aux2 = c0_mais[33-1];
                    c0_mais_aux3 = c0_mais[33-2];

                    //  *c0_menos[N-1] <= (z1/(1-(z1*z1)))*( c0_mais[N-1] +
z1* c0_mais[N-2]);

                    calc_1[52:0] = z1[25:0] * c0_mais_aux3[25:0];

                    //calc_2[19:0] = c0_mais_aux2[19:0] + calc_1[29:9];

                    // c0_menos_aux2[40:0] = z1_aux[19:0]*calc_2[19:0];

                    // sinal para a operação de multiplicacao
                    if ( z1[26] == 1'b0 && c0_mais_aux3[26] == 1'b1 )
                           begin
                           calc_1[53] = 1'b1;
                           end
                    else
                    begin
                           if ( z1[26] == 1'b1 && c0_mais_aux3[26] == 1'b0 )
                                  begin
                                  calc_1[53] = 1'b1;
                                  end
                           else
                           begin
                                  if ( z1[26] == 1'b1 && c0_mais_aux3[26] ==
1'b1 )
                                         begin
                                         calc_1[53] = 1'b0;
                                         end
                                  else
                                  begin
```

```
                                        if ( z1[26] == 1'b0 &&
c0_mais_aux3[26] == 1'b0 )
                                                begin
                                                calc_1[53] = 1'b0;
                                                end
                              end
                    end
                end

                        // limpar registos auxiliares
                        aux_p = 27'd0;
                        aux_p1 = 27'd0;
                        P = 28'd0;

                        // sinal para a operação de adição e resultado da mesma
                        if((c0_mais_aux2[25:0] > calc_1[40:14]) &&
(c0_mais_aux2[26] == 1'b1) && (calc_1[53] == 1'b0) )
                                begin
                                aux_p[26:0] = calc_1[40:14] - c0_mais_aux2[25:0];
                                // converter complemento para 2 para unsigned bits
                                aux_p1[26:0] = aux_p[26:0] - 27'd1;
                                P[26:0] = ! aux_p1[26:0];
                                // sinal
                                P[27] = c0_mais_aux2[26];
                                end
                        else
                                begin
                                if ((c0_mais_aux2[25:0] < calc_1[40:14]) &&
(c0_mais_aux2[26] == 1'b1) && (calc_1[53] == 1'b0) )
                                        begin
                                        aux_p[26:0] = calc_1[40:14] -
c0_mais_aux2[25:0];
                                        // converter complemento para 2 para unsigned
bits
                                        aux_p1[26:0] = aux_p[26:0] - 27'd1;
                                        P[26:0] = ! aux_p1[26:0];
                                        // sinal
                                        P[27] = calc_1[53];
```

```
                                end
                        else
                        begin
                                if ((z1[25:0] > calc_1[40:14]) &&
(c0_mais_aux2[26] == 1'b0) && (calc_1[53] == 1'b1) )
                                        begin
                                        aux_p[26:0] = c0_mais_aux2[25:0] -
calc_1[40:14];
                                        // converter complemento para 2 para
unsigned bits
                                        aux_p1[26:0] = aux_p[26:0] - 27'd1;
                                        P[26:0] = ! aux_p1[26:0];
                                        // sinal
                                        P[27] = c0_mais_aux2[26];
                                        end
                                else
                                begin
                                                if ((c0_mais_aux2[25:0] <
calc_1[40:14]) && (c0_mais_aux2[26] == 1'b0) && (calc_1[53] == 1'b1) )
                                                        begin
                                                        aux_p[26:0] =
calc_1[40:14] - c0_mais_aux2[25:0];
                                                        // converter complemento
para 2 para unsigned bits
                                                        aux_p1[26:0] =
aux_p[26:0] - 27'd1;
                                                        P[26:0] = ! aux_p1[26:0];
                                                        // sinal
                                                        P[27] = calc_1[53];
                                                        end
                                                else
                                                begin
                                                        if ( (c0_mais_aux2[26] ==
1'b1) && (calc_1[53] == 1'b1) )
                                                        begin
                                                        aux_p[26:0] =
c0_mais_aux2[25:0] + calc_1[40:14];
```

```
                                                P =
{c0_mais_aux2[26],aux_p};
                                                end
                                                else
                                                begin
                                                      if (
(c0_mais_aux2[26] == 1'b0) && (calc_1[53] == 1'b0) )
                                                      begin
                                                            aux_p[26:0]
= c0_mais_aux2[25:0] + calc_1[40:14];
                                                            P =
{c0_mais_aux2[26],aux_p};
                                                      end
                                                end
                                          end
                              end
                        end
                  end

                  calc_2 = {P[27],P[26:1]};

                  //  *c0_menos[N-1] <= (z1/(1-(z1*z1)))*( c0_mais[N-1] +
z1* c0_mais[N-2]);

                  c0_menos_aux2[53:0] = z1_aux[25:0]*calc_2[25:0];

                  // sinal para a operação de multiplicacao
                  if ( z1_aux[26] == 1'b0 && calc_2[26] == 1'b1 )
                        begin
                        c0_menos_aux2[35] = 1'b1;
                        end
                  else
                  begin
                        if ( z1_aux[26] == 1'b1 && calc_2[26] == 1'b0 )
                              begin
                              c0_menos_aux2[35] = 1'b1;
                              end
                        else
```

```
                        begin
                                if ( z1_aux[26] == 1'b1 && calc_2[26] == 1'b1
)
                                        begin
                                        c0_menos_aux2[35] = 1'b0;
                                        end
                                else
                                begin
                                        if ( z1_aux[26] == 1'b0 && calc_2[26]
== 1'b0 )
                                                begin
                                                c0_menos_aux2[35] = 1'b0;
                                                end
                            end
                    end
                end

                    if (c0_menos_aux2[35:9] == 27'b100000000000000000000000000
) // caso de duplo 0 na notacao binaria escolhida
                    begin
                    c0_menos_aux2[35:9] = 27'b000000000000000000000000000;
                    end

                    c0_menos[32] = c0_menos_aux2[35:9];

                    $display("\n c0_menos_aux2 = %b, calc_1 = %b, calc_2 = %b,
c0_menos[35] = %b ", c0_menos_aux2, calc_1, calc_2, c0_menos[35]);

                    //for (k=N-2; k<=1; k=k-1)
                    //
                    //    c0_menos[k] <= z1 * (c0_menos[k+1] - c0_mais[k]);

                    // limpar registos auxiliares
                    aux_p = 27'd0;
                    aux_p1 = 27'd0;
                    P = 28'd0;

                    for( l = 31 ; l >= 0; l = l-1 )
```

```
                    begin
                    $display ("teste de codigo l= %d", l);
                    aux_c0_menos_32 = c0_menos[32];
                    aux_c0_mais_k = c0_mais[l];

                    //calc_3 = aux_c0_menos_32 - aux_c0_mais_k;

                    // Sinal e calculo auxiliar calc_3
                    // sinal para a operação de adição e resultado da mesma
                    if((aux_c0_menos_32[25:0] > aux_c0_mais_k[25:0]) &&
(aux_c0_menos_32[26] == 1'b1) && (aux_c0_mais_k[26] == 1'b0) )
                          begin
                          aux_p[26:0] = aux_c0_mais_k[25:0] -
aux_c0_menos_32[25:0];
                          // converter complemento para 2 para unsigned bits
                          aux_p1[26:0] = aux_p[26:0] - 27'd1;
                          P[26:0] = ! aux_p1[26:0];
                          // sinal
                          P[27] = aux_c0_menos_32[26];
                          end
                    else
                          begin
                          if ((aux_c0_menos_32[25:0] < aux_c0_mais_k[25:0]) &&
(aux_c0_menos_32[26] == 1'b1) && (aux_c0_mais_k[26] == 1'b0) )
                                begin
                                aux_p[26:0] = aux_c0_mais_k[40:14] -
aux_c0_menos_32[25:0];
                                // converter complemento para 2 para unsigned
bits
                                aux_p1[26:0] = aux_p[26:0] - 27'd1;
                                P[26:0] = ! aux_p1[26:0];
                                // sinal
                                P[27] = aux_c0_mais_k[26];
                                end
                          else
                          begin
```

```
                                if ((aux_c0_menos_32[25:0] >
aux_c0_mais_k[25:0]) && (aux_c0_menos_32[26] == 1'b0) && (aux_c0_mais_k[26] ==
1'b1) )
                                        begin
                                        aux_p[26:0] = aux_c0_menos_32[25:0] -
aux_c0_mais_k[25:0];
                                        // converter complemento para 2 para
unsigned bits
                                        aux_p1[26:0] = aux_p[26:0] - 27'd1;
                                        P[26:0] = ! aux_p1[26:0];
                                        // sinal
                                        P[27] = aux_c0_menos_32[26];
                                        end
                                else
                                begin
                                        if ((aux_c0_menos_32[25:0] <
aux_c0_mais_k[25:0]) && (aux_c0_menos_32[26] == 1'b0) && (aux_c0_mais_k[26] ==
1'b1) )
                                                begin
                                                aux_p[26:0] =
aux_c0_mais_k[25:0] - aux_c0_menos_32[25:0];
                                                // converter complemento
para 2 para unsigned bits
                                                aux_p1[26:0] =
aux_p[26:0] - 27'd1;
                                                P[26:0] = ! aux_p1[26:0];
                                                // sinal
                                                P[27] =
aux_c0_mais_k[26];
                                                end
                                        else
                                        begin
                                                if ( (aux_c0_menos_32[26]
== 1'b1) && (aux_c0_mais_k[26] == 1'b1) )
                                                begin
                                                aux_p[26:0] =
aux_c0_menos_32[25:0] + aux_c0_mais_k[25:0];
```

```
                                                P =
{aux_c0_menos_32[26],aux_p};
                                                end
                                                else
                                                begin
                                                      if (
(aux_c0_menos_32[26] == 1'b0) && (aux_c0_mais_k[26] == 1'b0) )
                                                      begin
                                                            aux_p[26:0]
= aux_c0_menos_32[25:0] + aux_c0_mais_k[25:0];
                                                            P =
{aux_c0_menos_32[26],aux_p};
                                                      end
                                                end
                                          end
                              end
                        end
                  end

                  calc_3 = {P[27],P[26:1]};

                  //c0_menos[k] = z1 * calc_3;

                  // Sinal

                  // sinal para a operação de multiplicacao
                  if ( z1[26] == 1'b0 && calc_3[26] == 1'b1 )
                        begin
                        c0_menos_k_aux[35] = 1'b1;
                        end
                  else
                  begin
                        if ( z1[26] == 1'b1 && calc_3[26] == 1'b0 )
                              begin
                              c0_menos_k_aux[35] = 1'b1;
                              end
                        else
                        begin
```

```
                                if ( z1[26] == 1'b1 && calc_3[26] == 1'b1 )
                                        begin
                                        c0_menos_k_aux[35] = 1'b0;
                                        end
                                else
                                begin
                                        if ( z1[26] == 1'b0 && calc_3[26] ==
1'b0 )
                                                begin
                                                c0_menos_k_aux[35] = 1'b0;
                                                end
                                end
                        end
                end

                c0_menos_k_aux[53:0] = z1[25:0] * calc_3[25:0];

                c0_menos[l] = c0_menos_k_aux[40:14];

                $display("\n c0_menos[%d] = %b ", l[5:0], c0_menos[l]);



                end // fim de ciclo for

                fd1 = $fopen("../testeblabla3.txt", "w");
                for( l=0; l<33; l = l+1)
                begin
                $fwrite(fd1, "\n c0_menos[%d] = %b ", l[5:0],
c0_menos[l]);
                end
                $fclose(fd1);

                fd2 = $fopen("../c0_menos3.dat", "w");
                for( l=0; l<33; l = l+1)
                begin
                $fwrite(fd2, "\n  %b ", c0_menos[l]);
                end
```

```
                    $fclose(fd2);

     end


endmodule
```

----------fim da listagem ----------

B.6 Módulo que implementa a spline cúbica para 36 amostras

```
------------------------------inicio da listagem de cubic_spline_36.v----------
`timescale 1ns/100ps
module cubic_spline_36( Clk_Sys);


input Clk_Sys;

reg [53:0] aux_mult;
reg [53:0] aux_spline;
reg [26:0] spline [36:0];
reg [26:0] c0_menos [36:0];
reg [26:0] aux_c0_menos;
reg [26:0] teste [36:0];
reg [26:0] aux_P3;
reg [32:0] spline_signal;
reg aux_mult_signal, aux_spline_signal;

parameter number = 27'b000000000110000000000000000; // número 6(10)
parameter menos_um = 27'b100000000001000000000000000;


integer k;

initial begin

$readmemb("../P3.dat", teste);



$readmemb("../c0_menos3.dat", c0_menos);

for(k=1;k<33;k=k+1)
begin

aux_P3 = teste[k];

$display("\n aux_P3[%d] = %b", k, aux_P3); //aux_div

aux_c0_menos = c0_menos[k];

$display("\n aux_c0_menos[%d] = %b", k, aux_c0_menos); //aux_div

aux_spline = aux_P3 * aux_c0_menos;

$display("\n aux_spline[%d] = %b", k, aux_spline);

aux_spline_signal = aux_P3[26] ^ aux_c0_menos[26];
```

```
spline[k] = aux_spline[40:13];

spline_signal[k] = aux_spline_signal;

$display("\n spline [%d] = %b, spline_signal[%d]", k[5:0], spline[k], k[5:0],
spline_signal[k]);

end

end

endmodule
----------------------------fim da listagem-----------------------------------
```

B.7 Módulo de teste da Spline cúbica

```
------------------------------inicio da listagem de
cubic_spline_36_testbench.v--------------
`timescale 1ns/100ps
module cubic_spline_36_testbench;

reg [32:0] P3;
reg reset, Clk_Sys;

always begin  // clock process

 Clk_Sys = 1'b0;
 #1
 Clk_Sys = 1'b1;
 #1
 #49
 Clk_Sys = 1'b0;
 #49
 Clk_Sys = 1'b0;
end



cubic_spline_36 cubic_spline_36_1( Clk_Sys);

initial
 begin

 $monitor( $time, " P3 = %b ", P3 );

#0
#100
#100

$stop;

end
endmodule
-----------------------------------fim de listagem-----------------------------
```

B.8 Módulo “Factores de escala.v”

```
-----------------------------inicio da listagem de factores de escala.v ------
`timescale 1ns/100ps
```

```
module factores_de_escala( reset, clock);


input reset, clock ;

parameter Max_bit = 27;
parameter factor1 = 27'b000000000000000001001001110; // 0.018005 ~0.018
parameter factor2 = 27'b100000000001000011001000110; // -1.049011 ~ -1.049

reg [Max_bit-1:0] X [36:0];
reg [Max_bit-1:0] aux6,aux_sub_compl2, aux_sub, teste_aux4;
reg [(2*(Max_bit-1)):0] aux_X;
reg aux2_sinal;
reg [Max_bit-1:0] aux4 [36:0];
reg [Max_bit-1:0] aux5;
reg [((Max_bit-1)*2):0] aux2_aux [36:0];
reg [((Max_bit-1)*2)+1:0] aux2 [36:0];

integer n,fd1, fd2, fd3;

// ler os valores de X



always @(posedge clock)                 //Ax + B
begin
if (reset == 1'b1)
begin
aux_sub_compl2 <= 27'd0;
aux_sub <= 27'd0;
aux6 <= 27'd0;
end
else
begin
$readmemb("../X.dat", X);

for (n=1; n<37 ; n=n+1)
begin

$display( "X[%d] = %p", n[5:0], aux_X);
// operação a realizar [(X*factor1) + factor2 ]
aux_X <= X[n] * factor1[Max_bit-2:0];   // X * factor1
#31
// escrita de dados
fd2 = $fopen("../aux_X.dat", "a+");
                    $fwrite(fd2, "\n %b", aux_X[41:15]);
                  $fclose(fd2);

// leitura de dados
$readmemb("../aux_X.dat", aux4);

teste_aux4 = aux4[n];

aux5 = factor2[(Max_bit-2):0] - teste_aux4[(Max_bit-2):0];
aux5[(Max_bit-1)] = 1'b0;

// escrita de dados
fd3 = $fopen("../aux5.dat", "a+");
                    $fwrite(fd3, "\n %b", aux5);
                  $fclose(fd3);


end

                    end
end
```

```
endmodule
```

----------fim do ficheiro----------

B.9 Ficheiro de dados de entrada “alfa.dat”

```
----------inicio de alfa1.dat ----------
0000000001 // 0.001
0000000010
 0000000011
 0000000100
 0000000101
 0000000110
 0000000111
 0000001000
 0000001001
 0000001010
 0000001011
 0000001100
 0000001101
 0000001110
 0000001111
 0000010000
 0000010001
 0000010010
 0000010011
 0000010100
 0000010101
 0000010110
 0000010111
 0000011000
 0000011001
 0000011010
 0000011011
 0000011100
 0000011101
 0000011110
 0000011111
 0000100000
 0000100001
 0000100010
 0000100011
 0000100100
----------fim de alfa.dat----------
```

B.10 Ficheiro de dados de entrada “beta.dat”

```
----------inicio do ficheiro beta1.dat----------
0000000001 // 0.001
 0000000010
 0000000011
 0000000100
 0000000101
 0000000110
```

```
 0000000111
 0000001000
 0000001001
 0000001010
 0000001011
 0000001100
 0000001101
 0000001110
 0000001111
 0000010000
 0000010001
 0000010010 // 0.0180
 0000010011 // 0.0188
 0000010100 // 0.0197
 0000010100 // 0.0205
 0000010101 // 0.0212
 0000010110 // 0.0220
 0000010111 // 0.0227
 0000011000 // 0.0235
 0000011000 // 0.0241
 0000011001 // 0.0247
 0000011001 // 0.0253
 0000011010 // 0.0259
 0000011011 // 0.0265
 0000011011 // 0.0270
 0000011100 // 0.0275
 0000011100 // 0.0280
 0000011100 // 0.0284
 0000011101 // 0.0287
 0000011110 // 0.0292
-----------------------------fim de beta1.dat----------------------------------
```

B.11 Ficheiro de dados X.dat

```
---------------------------------início do ficheiro de dados X.dat-------------------------------------
00000000000100000000000000
 00000000001000000000000000
 00000000001100000000000000
 00000000010000000000000000
 00000000010100000000000000
 00000000011000000000000000
 00000000011100000000000000
 00000000100000000000000000
 00000000100100000000000000
 00000000101000000000000000
 00000000101100000000000000
 00000000110000000000000000
 00000000110100000000000000
 00000000111000000000000000
 00000000111100000000000000
 00000001000000000000000000
 00000001000100000000000000
 00000001001000000000000000
 00000001001100000000000000
 00000001010000000000000000
 00000001010100000000000000
 00000001011000000000000000
 00000001011100000000000000
 00000001100000000000000000
 00000001100100000000000000
 00000001101000000000000000
 00000001101100000000000000
 00000001110000000000000000
```

```
 00000001110100000000000000
 00000001111000000000000000
 00000001111100000000000000
 00000010000000000000000000
 00000010000100000000000000
 00000010001000000000000000
 00000010001100000000000000
 00000010010000000000000000
```

----------fim do ficheiro X.dat ----------

B.12 Ficheiro de dados Y.dat

```
------------------inicio do ficheiro Y.dat ----------------------------------
00000000001000000000000000
 00000000001000000000000000
 00000000001100000000000000
 00000000010000000000000000
 00000000010100000000000000
 00000000011000000000000000
 00000000011100000000000000
 00000000100000000000000000
 00000000100100000000000000
 00000000101000000000000000
 00000000101100000000000000
 00000000110000000000000000
 00000000110100000000000000
 00000000111000000000000000
 00000000111100000000000000
 00000001000000000000000000
 00000001000100000000000000
 00000001001000000000000000
 00000001001100000000000000
 00000001010000000000000000
 00000001010000000000000000
 00000001010100000000000000
 00000001011000000000000000
 00000001011100000000000000
 00000001100000000000000000
 00000001100000000000000000
 00000001100100000000000000
 00000001100100000000000000
 00000001101000000000000000
 00000001101100000000000000
 00000001101100000000000000
 00000001110000000000000000
 00000001110000000000000000
 00000001110000000000000000
 00000001110100000000000000
 00000001111000000000000000
----------------------------------fim do ficheiro Y.dat------------------------
```

B.13 Ficheiro de dados C0_menos.dat

```
------------------------------inicio de C0_menos3.dat------------------------

  00000000000000000000000000
  00000000000001000100100100
  00000000000010000100100110000
  00000000000011001110111001100
  00000000000010001001001100000
  00000000000010011010010101100
  00000000000010101011011111000
  00000000000010111100101000100
  00000000000011001101110010000
  00000000000011011110111011100
  00000000000011110000000101000
  00000000001000000010011101000
  00000000001000100100110000000
  00000000001001000111000011100
  00000000001001101001010110000
  00000000001010001011101001000
  00000000001010011100110010100
  00000000001010110110100000011
  00000000001011010100100000101
  00000000001011110100101011000
  00000000011000101011110011000
  00000000011001001101000001100
  00000000011001111111111110111
  00000000011010011001011000101
  00000000011011001000011010000
  00000000011100000001000111000
  00000000011100011111100100000
  00000000011101001111111100000
  00000000000000000000000000000
------------------------------------fim de C0_menos3.dat----------------------
```

B.14 Ficheiro de dados P3.dat

```
-------------------------inicio de P3.dat-------------------------------------
----------------------------------
000000000001001001100110010
000000000001000101110101111
000000000001000010001100111
000000000000111110101010101
000000000000111011001111011
000000000000110111111011101
000000000000110100101110101
000000000000110001101000110
000000000000101110101010001
000000000000101011110010011
000000000000101001000001110
000000000000101001000001110
000000000000100011110101111
000000000000100001011010100
000000000000100000111001100
```

```
000000000000100011000110110
000000000000100101001100111
000000000000100111001011110
000000000000101001000011110
000000000000101010110100101
000000000000101100011110010
000000000000101110000001000
000000000000101110000001000
000000000000110000110001001
000000000000110001111110101
000000000000110011000101000
000000000000110100000100001
000000000000110100111100011
000000000000110101101101011
000000000000110110010111100
000000000000110110111010011
000000000000110111010110010
000000000000110111101011000
----------------------------------fim de P3.dat-----------------------
```

B.15 Script dos resultados do teste de spline cúbico

```
--------------------------------------inicio do script-------------------------
# Compile of cubic_spline_36.v was successful.
vsim work_spline15.cubic_spline_36
# vsim work_spline15.cubic_spline_36

#  spline [ 1] = 000000000001 001010110110101, spline_signal[ 1]0

#  spline [ 2] = 000000000010 001101111001111, spline_signal[ 2]0

#  spline [ 3] = 000000000011 001001011111011, spline_signal[ 3]0

#  spline [ 4] = 000000000011 111101111110000, spline_signal[ 4]0

#  spline [ 5] = 000000000100 101011101110110, spline_signal[ 5]0

#  spline [ 6] = 000000000101 010011000101000, spline_signal[ 6]0

#  spline [ 7] = 000000000101 110100011001011, spline_signal[ 7]0

#  spline [ 8] = 000000000110 010000000011101, spline_signal[ 8]0


#  spline [ 9] = 000000000110 100110010111101, spline_signal[ 9]0

#  spline [10] = 000000000110 110111101110001, spline_signal[10]0

#  spline [11] = 000000000111 100011101100011, spline_signal[11]0

#  spline [12] = 000000000111 001100111110010, spline_signal[12]0

#  spline [13] = 000000000111 010001100011111, spline_signal[13]0

#  spline [14] = 000000000111 101101101000110, spline_signal[14]0

#  spline [15] = 000000001000 110100011000001, spline_signal[15]0

#  spline [16] = 000000001001 111101111100010, spline_signal[16]0

#  spline [17] = 000000001011 001001111011101, spline_signal[17]0

#  spline [18] = 000000001100 011000000110100, spline_signal[18]0
```

```
#  spline [19] = 000000001101 101000000001011, spline_signal[19]0

#  spline [20] = 000000001110 100001011101101, spline_signal[20]0

#  spline [21] = 000000001111 100110101100110, spline_signal[21]0

#  spline [22] = 000000010000 010001110110011, spline_signal[22]0

#  spline [23] = 000000010010 000001001010000, spline_signal[23]0

#  spline [24] = 000000010011 010001101011111, spline_signal[24]0

#  spline [25] = 000000010100 000111010110001, spline_signal[25]0

#  spline [26] = 000000010101 001001100111100, spline_signal[26]0

#  spline [27] = 000000010101 110101100001100, spline_signal[27]0

#  spline [28] = 000000010110 110001001100100, spline_signal[28]0

#  spline [29] = 000000010111 110100001010101, spline_signal[29]0

#  spline [30] = 000000011000 011100001111101, spline_signal[30]0

#  spline [31] = 000000011001 010010110011111, spline_signal[31]0

#  spline [32] = 000000000000 000000000000000, spline_signal[32]0
----------------------------------fim do script-------------------------------
```

Referências

[1] Kenington, Peter B., “High-Linearity RF Amplifier Design.”, Artech House Publishers – Boston-London, 2000, ISBN 1-58053-143-1.

[2]Haykin, Simon, “Communication Systems”, Jonh Wiley and Sons, Inc., 2001, ISBN 0-471-17869-1.

[3] J. Machado da Silva, G. Pinho, and P. Mota, “A Built-In Self Test Approach for RF Power Amplifiers”, In Informal Digest of the IEEE 12th International Mixed Signals Testing Workshop, June 2006.

[4] Pedro Mota, José Machado da Silva, and John Long, "Estimation and adaptive correction of PA's nonlinearities", - 13th International Mixed Signals Testing Workshop and 3rd GHz/Gbps Test Workshop, Junho 2007.

[5]W.Schelmbauer, R.Weigel, B.Adler, J.Fenk, H.Pretl, and L.Maurer. Linearity Considerations of W-CDMA Front-Ends for UMTS, IEEE Transactions on Circuits and Systems I, 53(7):1468-1476, July 2006.

[6] Pedro Mota, José Silva, John Long,"An Adaptive Scheme for Estimating and Correcting RF Amplifiers' Non-Linearities", IEEE 9th of the biennial Asia Pacific Conference on Circuits and Systems (APCCAS 2008), Venetian Macao-Resort-Hotel, Macao, China, Dezembro 2008.

[7] Moreira, Silvio Abrantes; Corte-Real, Luís; “Apontamentos de Fundamentos de Telecomunicações I” – 2009/2010 – FEUP.

[8] Pedro Mota, José Machado da Silva, Ricardo Veiga, "Estimation of RF PA Non-Linearities After Cross-Correlating Current and Output Voltage", Journal of Electronic Testing - Theory and Applications – Special Issue on Analog, Mixed-Signal and RF Testing, January 2010.

[9]Valdes-Garcia, A.; Silva-Martinez, J.; Sanchez-Sinêncio, E.; “On-Chip Testing Techniques for RF Wireless Transceivers”, IEEE Design & Test of Computers, Vol. 23, No. 4, pp. 268 – 277, April 2006.

[10]F. Ellinger, Radio Frequency Integrated Circuits and Technologies, Springer-Verlag, Berlin, Heidelberg, 2007.

[11]Choongeol Cho, William R. Eisenstadt, Bob Stengel, and Enrique Ferrer, “IIP3 Estimation from the Gain Compression Curve”, IEEE Transactions On Microwave Theory and Techniques, Vol. 53, No. 4, April 2005.

[12] Calogero D. Presti, Francesco Carrara, Antonino Scuderi, and Giuseppe Palmisano, “Fast Peak Detector with Improved Accuracy and Linearity for High-Frequency Waveform Processing”, IEEE International Symposium on Circuits and Systems, 2007, ISCAS 2007.

[13] Francesco Carrara, Calodero D. Presti, Antonino Scuderi and Giuseppe Palmisano, “A Methodology for fast VSWR Protection Implemented in a Monolithic 3W 55% PAE RF CMOS Power Amplifier”, IEEE Journal of Solid State Circuits, Volume 43, Issue 9, 2008.

[14] N. Vasudev, “Measurement of a Filter Using Power Detector”, IEEE Transactions on Microwave Theory and Techniques, Volume 50, Issue 9, pp.2083-2089, 2002.
[15] Edite Manuela da G.P.Fernandes, Computação Numérica, 1996 Universidade do Minho, ISBN 972-96944-0-0.
[16] Mathews, John H., Kurtis D. Fink, “Numerical Methods Using MATLAB” –Third Edition – Prentice Hall, Inc.
[17] Podell, Allen, “RF and uW Power Amplifiers” – Webminar – May 2010.
[18] Fazel, K. and Kaiser, S. “Analysis of Non-Linear Distortions on MC-CDMA”, IEEE International Conference on Communications, 1998, ICC98, pp.1028-1034, vol. 2, 1998.
[19] Balashov, E.V.; Pasquet, D.; Korotkov, A.S.; Bourdel, E.; Giannini, F.; “Title Automatization of compression point 1dB (CP1dB) and input 3rd order intercept point (IIP3) measurements using lab VIEW”, Signals, Circuits and Systems, 2005. ISSCS 2005. International Symposium - Online Date26 Sep 2005.
[20] Razavi, Behzad, “RF Microelectronics”, Prentice Hall (November 16, 1997).
[21] Schoenberg, I. J., “Contribution To The Problem Of Approximation Of Equidistant Data By Analytic Functions”, Quart. Appl. Math, Vol. 4, pp. 45-99, 112-141, 1946.
[22] De Boor, C., “On calculating with B-splines”, J. Approximation Theory, Vol. 6, pp.50-62, 1972.
[23] Schumaker, L.L., “Spline Functions: Basic Theory”, New York: Wiley, 1981.
[24] Prenter, P.M., “Splines and Variational Methods”, New York: Wiley, 1975.
[25] Ahlberg, J.H., Nilson, E. N., and Walsh, J.L., “The Theory of Splines and Their Applications”, New York: Academic Press, 1967.
[26] Diercks, P., “Curve and Surface Fitting With Splines”, Oxford: Oxford University Press, 1995.
[27] Lancaster, P., and Salkauskas, K., “Curve and Surface Fitting: An Introduction”, London: Academic Press, 1986.
[28] Bartels, R.H., Beatty, J.C., and Barsky, B.A., “Splines for use in Computer Graphics”, Los Altos, CA: Morgan Kaufmann, 1987.
[29] Mallat, S.G., “A Theory of multiresolution signal decomposition: the wavelet representation”, IEEE Trans. Pattern Analysis Machine Intell, vol. PAMI-11, no.7, pp. 674-693, 1989.
[30] De Boor, C., “A Pratical Guide to Splines”, New York: Springer-Verlag, 1978.
[31] Goshtasby, A., Cheng, F., and Barsky, B.A., “B-spline curves and surfaces viewed as digital filters”, Computer Vision, Graphics and Image Processing, vol. 52, pp. 264-275, 1990.
[32] Unser, M., Aldroubi, A., and Eden, M., “Fast B-Spline transforms for continuous image representation and interpolation”, IEEE Trans. Pattern Analysis Machine Intell, vol.13 no.3 pp.227-285, 1991.

[33] Unser, M., Aldroubi, A., and Eden, M., “B-spline signal processing: Part II – efficient design and applications”, IEEE Trans. Signal Processing, Vol.41, no.2, pp. 834-848, 1993.
[34] Unser, M., Aldroubi, A., and Eden, M., “B-spline signal processing: Part I theory”, IEEE Trans. Signal Processing, vol.41, no.2,pp.821-833, 1993.
[35] Unser, M., “Splines: a perfect fit for signal and image processing”, Signal Processing Magazine, IEEE, pp.22-38, November 1999.
[36] Cargal, J.M., “Honer’s Algorithm”, Chapter 8 out of 37 from Discrete Mathematics for Neophytes: Number Theory, Probability, Algorithms, and Other Stuff.
[37] Naoki Mikami, Masaki Kobayashi, and Yukiko Yokoyama, “Roundoff-Error reduction for evaluation of a Function by Polynomial Aproximation with Error FeedBack in Fixed-Point Arithmetic”, IEEE Transactions on Signal Processing, VOL. 41, NO.5 May 1993, pp 1953-1954.
[38] Ciletti, Michael D., “Advanced Digital design with the Verilog HDL”, 1st edition, Prentice Hall Xilinx Design Series, 2003.
[39]http://cnx.org/content/m12812/latest/ Acesso em 27/06/2010.
[40]http://www.cfd-online.com/Wiki/Tridiagonal_matrix_algorithm_-_TDMA_%28Thomas_algorithm%29 Acesso em 27/06/2010.