Robust tools for weighted Chebyshev approximation and applications to digital filter design

Silviu-Ioan Filip


Numéro National de Thèse : 2016LYSEN063

THÈSE de DOCTORAT DE L'UNIVERSITÉ DE LYON
opérée par
l'École Normale Supérieure de Lyon

École Doctorale N° 512
École Doctorale en Informatique et Mathématiques de Lyon

Spécialité de doctorat :
Informatique

Soutenue publiquement le 07/12/2016, par :
Silviu-Ioan FILIP

Robust tools for weighted Chebyshev approximation and applications to digital filter design

Outils robustes pour l'approximation de Chebyshev pondérée et applications à la synthèse de filtres numériques

---

Devant le jury composé de :

Beckermann, Bernhard	Professeur	Université de Lille	Rapporteur
Sentieys, Olivier	Directeur de recherche	Université de Rennes ENSSAT Lannion	Rapporteur
Belfiore, Jean-Claude	Professeur	Télécom ParisTech	Examinateur
Leblond, Juliette	Directrice de recherche	Centre de recherche INRIA	Examinatrice
Trefethen, Lloyd Nicholas	Professeur	University of Oxford	Examinateur
Brisebarre, Nicolas	Chargé de recherche	ENS de Lyon	Co-directeur
Hanrot, Guillaume	Professeur	ENS de Lyon	Directeur de thèse

Contents

List of Figures

List of Tables

Notation

$\mathbb{N}$	The set of natural numbers $\{0, 1, \ldots\}$.
$\mathbb{N}^\star$	The set of natural numbers, without 0: $\{1, 2, \ldots\}$.
$\mathbb{Z}$	The set of integers $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$.
$\mathbb{R}$	The set of real numbers.
$\mathbb{C}$	The set of complex numbers.
$\mathbb{R}[x]$	The set of univariate polynomials with real coefficients.
$\mathbb{R}_n[x]$	The set of univariate polynomials with real coefficients of degree at most n.
$\mathbb{R}_n[x_1, \ldots, x_d]$	The set of d-variate polynomials with real coefficients of degree at most n.
$\mathcal{C}(X)$	The set of all continuous functions over the set X.
$\mathrm{sgn}(x)$	The sign function: returns 1 if $x > 0$, -1 if $x < 0$ and is undefined if $x = 0$.
$\mathbf{a}$	Small bold letters are used to represent column vectors.
$\mathbf{A}$	Capital bold letters are used to represent matrices.
$\mathbf{A}^t$	The transpose of the matrix $\mathbf{A}$.
$\mathbf{A}^\star$	The conjugate transpose of the matrix $\mathbf{A}$ (*i.e.,* $\overline{\mathbf{A}}^t$).
$\mathrm{diag}(\mathbf{a})$	The diagonal matrix whose diagonal is formed by taking the elements of $\mathbf{a}$.
$\ker(\mathbf{A})$	The kernel of the matrix $\mathbf{A}$, that is, the set of vectors $\mathbf{x}$ for which $\mathbf{Ax} = \mathbf{0}$.
$\mathcal{L}_2([a, b], w(x), \mathrm{d}x)$	The space of square-integrable functions on $[a, b]$ (*i.e.,* if $f \in \mathcal{L}_2([a, b], w(x), \mathrm{d}x)$, then $\int_a^b \lvert f(x)\rvert^2 w(x)\mathrm{d}x < \infty$).

Acronyms

FIR	finite impulse response
IIR	infinite impulse response
FPGA	Field Programmable Gate Array
DCT	Discrete Cosine Transform
FFT	Fast Fourier Transform
DTFT	Discrete-Time Fourier Transform
IDTFT	Inverse Discrete-Time Fourier Transform
LTI	linear-time invariant
CCDE	constant-coefficient difference equation
CSD	canonic signed digit
MR	maximal-ripple
ER	extra-ripple
AFP	approximate Fekete point
DLP	discrete Leja point
AM	admissible mesh
WAM	weakly admissible mesh
CPR	Chebyshev-proxy root-finder
DSP	digital signal processing
MILP	miced integer linear programming
CVP	closest vector problem
SVP	shortest vector problem
GSO	Gram-Schmidt orthogonalization
LLL	Lenstra-Lenstra-Lovász
HKZ	Hermite-Korkine-Zolotarev
BKZ	block Korkine-Zolotarev
LUT	look-up table
CLB	configurable logic block
IP	intellectual property
FD	frequency domain
TD	time domain
HW	hardware
SOPC	sum of products with constants
ASIC	application-specific integrated circuit

1 Introduction

All exact science is dominated by the idea of approximation.
Bertrand Russel

This is a thesis about univariate minimax *approximation* problems, most of which are derived from applications found in *Digital Signal Processing* (more specifically *digital filter design*). The focus is on the study and improvement of existing algorithmic solutions for these problems, together with corresponding implementations which are robust and convenient to use in practice.

A central topic of *Approximation Theory* is the constructive approximation of functions, which broadly speaking, revolves around the idea of *replacing* a given function f by a simpler one, having values which are not necessarily identical, but very close (with respect to an established error measure) to those of f. The ultimate goal of doing this is to have the ability to *manipulate* and *answer* nontrivial questions about the nature of f in a numerical context, where computational resources are finite. Having mathematically sound and computationally efficient *algorithms* to work with such *proxies* of f is hence crucial.

Choosing from a particular *class* $\mathcal{P}$ of functions a proxy for f raises some natural questions, both from a theoretical and practical implementation point of view:

- Does f have a *best* approximation (according to a given optimality criterion) $p^\star \in \mathcal{P}$ and is it algorithmically feasible to compute it?
- Consider $\mathcal{P}$ to be a finite dimensional vector space, for instance $\mathbb{R}_n[x]$. Then, if $p^\star$ exists, let us represent it in a particular basis $\{\varphi_0, \ldots, \varphi_n\}$ of $\mathbb{R}_n[x]$ (*i.e.,* $p^\star(x) = \sum_{k=0}^{n} a_k \varphi_k(x)$ is completely determined by $a_0, \ldots, a_n \in \mathbb{R}$). Since real numbers inside computers usually require that they be approximated by values from a finite set, how does this impact the quality of the approximation we will actually use?

Here are some concrete examples used throughout the text where such questions are relevant. An important constant in all of them are the Chebyshev polynomials of the first kind (treated in more detail in Section 2.1), which are the elements of $\mathbb{R}[x]$ defined by the recurrence

$$T_0(x) = 1, T_1(x) = x, T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x), n \geqslant 2. \tag{1.1}$$

(i) For $n = 62$, determine $p^\star \in \mathbb{R}_n[x]$ expressed in the basis $\{T_i\}_{0 \leqslant i \leqslant n}$, such that for $X = [-1, \cos(0.84\pi)] \cup [\cos(0.68\pi), \cos(0.4\pi)] \cup [\cos(0.24\pi), 1]$

$$f(x) = \begin{cases} 1, & x \in [-1, \cos(0.84\pi)] \cup [\cos(0.24\pi), 1], \\ 0, & x \in [\cos(0.68\pi), \cos(0.4\pi)], \end{cases} \quad \text{and} \quad w(x) = 1,$$

the weighted error $\delta = \max_{x\in X} |w(x)(f(x) - p^\star(x))|$ is minimal. What happens when we want the Chebyshev coefficients of our approximation to be stored using 21-bit wide fixed-point numbers? Can we still determine an optimal result efficiently?

Numerically, getting an approximation that is close to $p^\star$ in this case is not considered to be very complicated, with numerical software packages like MATLAB (their `firpm` command) having no problem outputting a result that would be qualified as good in practice. The default function call is

```
[h, err]=firpm(124, [0.0 0.24 0.4 0.68 0.84 1.0], [1 1 0 0 1 1], [1 1 1]);
```

The approximation to δ it outputs (the `err` variable) is about $1.2538 \cdot 10^{-2}$, which, based on computations we do with our own tools from Ch. 3, only seems to be accurate to two decimal digits (the actual δ is closer to $1.2785 \cdot 10^{-2}$). This is due mainly to the fact that the `firpm` command works on a discretized version of X containing about $11 \cdot 62 = 682$ points, instead of the actual domain. If we increase the discretization size by about a factor of four, meaning we execute

```
[h, err]=firpm(124, [0.0 0.24 0.4 0.68 0.84 1.0], [1 1 0 0 1 1], [1 1 1], {68});
```

four digits of accuracy are now available at the cost of doubling the computation time; the two calls took about 0.13s and 0.27s in MATLAB R2014B on my Linux machine equipped with an Intel i7-3687U CPU.

On the other hand, adding fixed-point format constraints on the polynomial coefficients complicates the problem considerably and a guaranteed optimal result is computationally hard to find, particularly in this case. The optimistic approach would then be to try and determine a quasi-optimal approximation in a short amount of time. For instance, using the tools in Ch. 4 we were able to obtain an approximation with error $1.56 \cdot 10^{-6}$, better than the best previously reported error $2.06 \cdot 10^{-6}$, computed under the same constraints [124].

(ii) For $n = 53248$, determine $p^\star \in \mathbb{R}_n[x]$ expressed in the basis of Chebyshev polynomials of the first kind such that for $X = [-1, \cos\left(\frac{3}{8192}\pi\right)] \cup [\cos\left(\frac{1}{8192}\pi\right), 1]$,

$$f(x) = \begin{cases} 0, & x \in [-1, \cos\left(\frac{3}{8192}\pi\right)], \\ 1, & x \in [\cos\left(\frac{1}{8192}\pi\right), 1], \end{cases} \quad \text{and} \quad w(x) = [(1+x)/2]^{1/2},$$

the weighted error $\max_{x\in X} |w(x)(f(x) - p^\star(x))|$ is minimal.

Whereas for **(i)** we stated that determining a sufficiently close approximation to $p^\star \in \mathbb{R}_n[x]$ is considered to be easy, this turns out to be much more difficult for **(ii)** using current software tools[1], in large part due to the large degree we are considering. We shall see however that most of the numerical issues that are characteristic to such problems can be alleviated (see Ch. 3).

Both **(i)** and **(ii)** are examples of (weighted) polynomial approximation questions that are specific to digital filter design (they give rise to a class of systems known as *finite impulse response* filters or FIR for short), an important branch of digital signal processing. They are very much the kind of problems that have motivated most of the work discussed in this text.

As an engineering discipline, digital signal processing is at the core of our modern digital world, enabling many of the communication technologies we are beginning to take for granted (one example is high-speed wireless data transmissions), medical devices used to diagnose and treat patients (examples include digital X-ray machines and digital hearing aid equipment) and can even be used to better understand and predict financial trends. Digital filtering becomes important when, for instance

- the data (*i.e.,* the signals) we have to process is inaccurate (the errors present in measured data are usually called noise in signal processing parlance) and we want, at least to a certain extent, to limit their impact (a classic example is noise cancellation in audio transmissions);
- in a more general setting, the signals we must process and interpret are saturated with more information than we actually need and some way of *filtering* out the non-required parts is desirable.

[1] MATLAB's `firpm` failed at the start of execution.

The information regarding univariate digital filtering we will need in this text is presented in Sec. 2.2.

(iii) Consider the approximation of $f(x) = e^x$ over $x \in (-\infty, 0]$ by rational functions with real coefficients of the same numerator and denominator degree n. For a fixed n, can we readily compute a degree (n, n) rational approximant $r^\star$ such that $\max_{x \leqslant 0} |f(x) - r^\star(x)|$ is (very close to) minimal?

This example is rather different than the first two and it is meant to give an idea of the kind of problems polynomials are not really suited for (in this case approximating a function with finite range over an unbounded domain). Rational approximation of the exponential function over $(-\infty, 0]$ was first looked at in a systematic manner by Cody, Meinardus and Varga [63] and has since been studied by quite a number of people, both for theoretical and practical reasons. Such approximations prove useful for instance when trying to find numerical solutions to stiff partial differential equations (see for instance [205, Ch. 23] and the references therein). We shall answer the question posed by this example in Ch. 6. Digital filters that map to rational approximation problems are also common and are known as *infinite impulse response* filters (IIR).

Our goal in this thesis is to, first of all, look at the existing algorithmic approaches used to solve problems like those given in **(i)**–**(iii)** and understand their strengths and limitations. Where possible, we try to introduce ideas that lead to more robust and efficient algorithms. Because of this, scalability of our approaches was always a big and constant driving factor if we ever planned to deal with such problems as those posed by **(ii)** or other practical instances where large degree polynomial approximations are needed.

As a result, the content of this text is a mix of numerical and symbolic ideas, used to build arithmetic algorithms. The most obvious thing that individualizes our work is the heavy focus on signal processing and digital filter design issues. What we get in the end is, we believe, an interesting and efficient blend of approaches mixing approximation theory, computer arithmetic, algorithmic number theory and digital signal processing.

Of course, we should always strive to walk the talk. That is why all the approaches described in the sequel come with open-source implementations which are available for testing. In the long run, we hope to integrate each of them in a more complete package for FIR and IIR filter design and synthesis.

Let us begin by formulating the approximation problems we wish to deal with in a more rigorous setting and outline what our contributions are in addressing them.

1.1 Minimax approximation - outline of the thesis

An *approximation problem*, as built by the examples in the previous paragraphs, consists of three major components:

- An element f to approximate that will belong to a *function class* Θ, which usually is a vector space (or linear space). We will only consider real functions of one real variable when discussing applications (although some of the theoretical results are also valid in multivariate settings). At the lowest level, we will be considering functions continuous on a certain compact subset X of $\mathbb{R}$, which we will henceforth denote by $\mathcal{C}(X)$. When extra properties for Θ are required, they will be explicitly stated.

- The type of approximation (*i.e.,* from what $\mathcal{P}$) we are searching for. We will be considering mainly spaces of *polynomials* and to a slightly lesser extent, those of certain *rational functions*. In both contexts, these types will be subspaces of Θ.

- A norm which will be used to measure the quality of a certain approximation to f. When not referring to a particular norm, we will denote it with $\|\cdot\|$. The closeness of a certain approximation p to f will then be measured by the scalar quantity $\|f - p\|$.

  Definition 1.1. Let V be a $\mathbb{R}$-vector space. A *norm* $\|\cdot\|$ on V is any map from V to the non-negative real numbers which verifies the axioms:

  1. $\|v\| \geqslant 0$, with equality if and only if $v = 0$;
  2. $\|u + v\| \leqslant \|u\| + \|v\|$ (the triangle inequality);

3. $\|\alpha u\| = |\alpha| \, \|u\|$, for any scalar α.

There are several standard choices of norms which are used in practice, with two of the most popular ones being:

– L_∞ norm (also known as uniform, minimax or Chebyshev norm).

$$\|f\|_{\infty,X} = \sup_{x \in X} |f(x)| \, ;$$

– L_2 norm (also known as least-squares or Euclidean norm).

$$\|f\|_{2,X} = \sqrt{\int_X (f(x))^2 \, w(x) \mathrm{d}x},$$

where $w \in \mathcal{C}(X)$ is a weight function which is positive almost everywhere.[2].

As the title of the manuscript suggests, our focus will be on L_∞ problems, but L_2 results and ideas will also play an important role in our discussions. By settling on a norm, the question of what a *best* approximation to f is [146, Def. 3.2]:

Definition 1.2 (ε-good, best and near-best approximations)**.**

- An element $p \in \mathcal{P}$ is said to be *ε-good* if $\|f - p\| \leqslant \varepsilon$, where ε is a prescribed target accuracy.
- An element $p^\star \in \mathcal{P}$ is said to be *best* if, for any other $p \in \mathcal{P}$, $\|f - p^\star\| \leqslant \|f - p\|$. A best approximation is not necessarily unique.
- An element $p \in \mathcal{P}$ is said to be *near-best* within a relative distance ρ if, $\|f - p\| \leqslant (1 + \rho) \, \|f - p^\star\| \, ,$ where $p^\star$ is a best approximation to f from $\mathcal{P}$.

All three types of approximations from Definition 1.2 play a role in practice. An example is the filter synthesis flow we consider in Ch. 5; the problems are initially posed in terms of ε-good polynomial approximations with real coefficients that satisfy certain error constraints. Also, it seems natural to try and compute ideal mathematical objects such as best approximations, as we will do in Ch. 3. And yet, best approximations are, apart from certain situations, very hard or impossible to compute exactly due to the finite nature of *machine representable* numerical formats. We are, in fact, forced to use near-best approximations with ρ close to zero (hopefully!), as we will exemplify in Ch. 4; near-best approximations also appear at each iteration of the algorithms of Ch. 3.

An application domain for minimax approximations we have not mentioned yet is that of implementing *elementary mathematical functions* for the design of mathematical libraries (like C's `libm`). We shall not mention them much in this text and instead refer the reader to the in-depth presentation made by Sylvain Chevillard in his PhD thesis [58].

Our main contributions, summarized over the next paragraphs, concentrate on providing efficient algorithmic solutions that deal with weighted minimax polynomial approximation problems over multi-interval domains, starting from initial specification and going on to actual hardware synthesis of the final result (the entirety of Chapters 3–5). The signal processing applications of such problems run very deep, and a considerable amount of work went into "filtering" the sometimes overwhelmingly large engineering literature on the subject.

[2]In the sequel, the context will make it clear what w weight function is being used.

Weighted minimax polynomial approximation on compact subsets of $\mathbb{R}$

Chapter 3 looks at the problem of determining best weighted polynomial approximations with real coefficients in the L_∞ setting. We perform a thorough study of so-called *exchange* algorithms (a classic topic in approximation theory literature and one which represents the "workhorse of digital filter design methods" [232]) that allow us to iteratively compute near-best approximations of increasingly better quality, converging to the best approximation (*i.e.,* $\rho \to 0$). For practical purposes, these approximations will be almost indistinguishable from the best ones, which is why we will also use the term minimax when referring to them, even though most of the time it will not be theoretically accurate. As it will quickly become apparent, the practical difficulties with implementing such algorithms in the filter design setting are very closely linked to the multi-interval nature of the approximation domain X. Despite the immense engineering literature on the exchange algorithm (see Section 3.5 for a comprehensive review of the state of the art regarding this), current implementations are not as robust as they could be, a fact we intend to change.

We introduce effective heuristics for initializing such algorithms in practice, ones which work extremely well even for problems when the target degree n is in the order of tens of thousands. While analyzing their impact, we also perform a thorough discussion on how one should *diagnose*, *debug*, but most of all, *avoid* numerical problems in polynomial exchange routines.

Machine-efficient weighted minimax FIR filter design

Chapter 4 takes into account the fact that the polynomial (and filter) coefficients will need to be represented using machine number (finite precision) coefficients (a problem known as *coefficient quantization* in the filter design literature). When these formats are small (as is frequently the case with applications targeting for example embedded circuits), they can have a huge impact on the quality of the approximation error. Exact approaches to solve such problems can be computationally demanding on moderately high degree instances (say, $n \geqslant 50$). Our heuristic approach for quantizing the filter coefficients is based upon previous work targeting machine-efficient polynomial approximations for `libm` design, namely the `fpminimax` algorithm (see [38] and [58, Ch. 2.3]). A key new element is the introduction of a suitable discretization of the approximation domain which makes it possible to extend this work to FIR applications. We also analyze its practical effectiveness and scalability, when compared to exact methods and other similar heuristics found in the literature (like for instance [124, 157]).

Tools for hardware synthesis of FIR filters

The goal of Chapter 5 is to offer a practical application to the ideas and tools we present in Chapters 3 and 4 in terms of synthesizing FIR filters on Field Programmable Gate Arrays (FPGAs). We describe an *automatic* tool capable of efficient filter synthesis, starting only from a very small number of user parameters. With guaranteed output accuracy properties, preliminary results show that our approach compares very favorably to other mainstream tools, where more of a trial and error strategy needs to be applied if we want to get anywhere near the same performance characteristics.

Minimax rational approximation problems

Chapter 6 offers a preliminary overview of our ongoing work on minimax rational approximation problems with real coefficients. Just like in Chapter 3, we give an account on the existing literature and numerical approaches, the spotlight being on the issues that limit practical robustness and what are some of the approaches we can try to alleviate them. Prototype implementations of rational exchange algorithms we are working on are also discussed, together with examples.

2 Technical prerequisites

To isolate mathematics from the practical demands of the sciences is to invite the sterility of a cow shut away from the bulls.
Pafnuty L. Chebyshev

We now review some basic facts regarding Chebyshev polynomials, digital filter design and Computer Arithmetic. We will rely on them in the sequel of the manuscript.

2.1 Chebyshev technology

We shall see that the process of determining minimax solutions to problems like **(i)**–**(iii)** from the previous chapter is quite involved. On the other hand, interpolating f is usually a much simpler operation, one that, if done well, can in many cases produce quality approximants. Interpolation is also at the core of many iterative procedures for computing minimax polynomial and rational approximations. Though natural, this relationship between interpolation and approximation is a turbulent one – in full generality, interpolation can fail to converge in any way as the number of points tend to infinity, even for very regular functions (the famous Runge phenomenon). It is in this context that Chebyshev points and polynomials are seen as an indispensable tool for performing numerical approximation in the computer era: using this set of points and under mild regularity assumption (Lipschitz-continuity), we can actually guarantee convergence and thus use interpolation as a tool for approximation.

A telling example of this is the Chebfun [78] MATLAB library, which, as its name implies, is built around Chebyshev-centered ideas. The same holds true for much of the work in this thesis. A more detailed presentation of the results in this section is [146] and many of them are also emphasized in [205]. Complementary results to the ones discussed here which are slightly more general are also presented in Section 3.6.1.

2.1.1 Basic facts about Chebyshev polynomials

Returning to the Chebyshev polynomials of the first kind we introduced with (1.1), the reader can check that they satisfy the following properties:

- for $x \in [-1, 1]$, we have
$$T_n(x) = \cos(n \arccos x), n \geqslant 0.$$
- $\{T_0, T_1, \ldots, T_n\}$ form a basis for $\mathbb{R}_n[x]$.

- T_{n+1} has $n+1$ distinct roots inside $[-1,1]$, called *Chebyshev nodes of the first kind*:

$$\mu_i = \cos\left(\frac{\left(n-i+\frac{1}{2}\right)\pi}{n+1}\right), \qquad i = 0,\dots,n.$$

- T_n has $n+1$ distinct local extrema over $[-1,1]$, called *Chebyshev nodes of the second kind*:

$$\nu_i = \cos\left(\frac{(n-i)\pi}{n}\right), \qquad i = 0,\dots,n.$$

The ν_i's also correspond to the zeros of the $n+1$-th Chebyshev polynomial of the second kind U_{n+1}, which can also be given recursively as

$$U_0(x) = 1, U_1(x) = 2x, U_n(x) = 2xU_{n-1}(x) - U_{n-2}(x), n \geqslant 2.$$

We also have $\dfrac{\mathrm{d}T_n}{\mathrm{d}x} = nU_{n-1}$, for $n \geqslant 1$.

From a minimax perspective, Chebyshev polynomials are quite interesting, one of the reasons being:

Lemma 2.1 (The minimax property of T_n)**.** *The minimax approximation on* $[-1,1]$ *to the zero function by a monic polynomial of degree* n *with real coefficients is* $2^{1-n}T_n(x)$.

Proof. See for instance [146, Ch. 3.2]. □

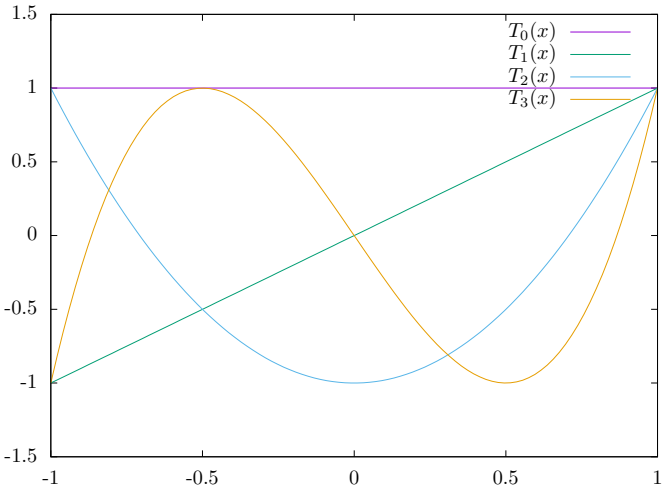

Figure 2.1: T_0 to T_3 on $[-1,1]$.

The equioscillatory nature of Chebyshev polynomials is also visible in Figure 2.1.

Equally noteworthy is that both types of Chebyshev polynomials are part of a larger class of functions called *orthogonal polynomials*, which, among other things, play an important role in numerical quadrature and least squares problems (see [88] for a recent in depth computational-centric account of orthogonal polynomials). To get a rough idea of some of their properties, we recall:

Definition 2.2. Given a weight $w \in \mathcal{C}([a,b])$ strictly positive almost everywhere over $[a,b]$, for any two functions $f, g \in \mathcal{L}_2([a,b], w(x), \mathrm{d}x)$ let their inner product with respect to w be[1]

$$\langle f, g\rangle = \int_a^b \overline{f(x)}g(x)w(x)\mathrm{d}x.$$

[1] Again, in the sequel, the context will make it clear to which w we are referring to.

f and g are said to be *orthogonal* to one another over $[a,b]$ with respect to the weight function w if $\langle f,g\rangle = 0$.

This inner product defines the L_2-type norm $\|f\|_{2,[a,b]} := \sqrt{\langle f,f\rangle}$. A family of orthogonal polynomials is any sequence of nonzero elements $(\varphi_n)_{n\geqslant 0}$, where $\varphi_n \in \mathbb{R}_n[x]$, that verifies:

$$\langle \varphi_i, \varphi_j\rangle = 0, (i \neq j).$$

Remark 2.3. The reader can check that both $(T_n)_{n\geqslant 0}$ and $(U_n)_{n\geqslant 0}$ form families of orthogonal polynomials over $[-1,1]$ with respect to the weight functions $(1-x^2)^{-\frac{1}{2}}$ and $(1-x^2)^{\frac{1}{2}}$. Also $\|T_0\|_2^2 = \pi, \|T_i\|_2^2 = \pi/2, i \geqslant 1$ and $\|U_i\|_2^2 = \pi/2, i \geqslant 0$.

Remark 2.4. The weight functions for Chebyshev polynomials are special cases of the *Jacobi weight function* $w(x) = (1-x)^\alpha(1+x)^\beta$, with $\alpha,\beta > -1$ as parameters. The orthogonal polynomials they generate over $[-1,1]$ are known as *Jacobi polynomials* and are denoted with $(P_n^{(\alpha,\beta)})_{n\geqslant 0}$. Maybe the most well-known such family is that of the *Legendre polynomials* which are common in physics and correspond to $\alpha = \beta = 0$ and $w(x) = 1$.

Orthogonal polynomials possess L_2 optimality characteristics analogous to the minimax property of Chebyshev polynomials in Lemma 2.1 (see [146, Ch. 4.3]):

Lemma 2.5. *If $(\varphi_n)_{n\geqslant 0}$ is a family of orthogonal polynomials over $[a,b]$, then:*

1. *the zero function is the best L_2 polynomial approximation of degree $n-1$ to φ_n over $[a,b]$;*
2. *φ_n is the best L_2 approximation to zero over $[a,b]$ among polynomials of degree n with the same leading coefficient.*

Moreover, it can be shown that:

Lemma 2.6. *The best L_2 polynomial approximation f_n of degree n to f may be expressed in terms of the orthogonal polynomial family $\{\varphi_i\}_{0\leqslant i\leqslant n}$ in the form*

$$f_n(x) = \sum_{i=0}^{n} a_i\varphi_i(x),$$

where

$$a_i = \frac{\langle f,\varphi_i\rangle}{\langle \varphi_i,\varphi_i\rangle}.$$

Thus, f_n is the projection of f onto the linear span of $\{\varphi_i\}_{0\leqslant i\leqslant n}$.

2.1.2 Chebyshev series and Chebyshev interpolants

If a given function f can be expanded in a (suitable convergent) series based on an orthogonal family $(\varphi_n)_{n\geqslant 0}$ over $[a,b]$, then we may write

$$f(x) = \sum_{k=0}^{\infty} a_k\varphi_k(x),$$

and notice that f_n inside Lemma 2.6 is in fact the partial sum of the orthogonal series expansion of f. We recall that a series is *absolutely convergent* if it remains convergent when each term is replaced by its absolute value (in which case one can reorder the series coefficients without changing the result). With respect to Chebyshev polynomials $(T_n)_{n\geqslant 0}$, if f is *Lipschitz continuous* over $[-1,1]$ (*i.e.,* there exists $C > 0$ such that $|f(x) - f(y)| < C\,|x-y|$ for all $x,y \in [-1,1]$), an assumption which is true in many cases (*e.g.,* as soon as f is continuously differentiable over $[-1,1]$), then

Theorem 2.7 (Chebyshev series)**.** *If f is Lipschitz continuous over $[-1,1]$, it has a unique representation as an absolutely and uniformly convergent series*

$$f(x) = \sum_{k=0}^{\infty} a_k T_k(x), \tag{2.1}$$

over $[-1,1]$*, with the coefficients* $a_k, k \geqslant 0$ *given by*

$$a_k = \frac{\langle f, T_k \rangle}{\langle T_k, T_k \rangle} = \frac{2}{\pi} \int_{-1}^{1} \frac{f(x) T_k(x)}{\sqrt{1-x^2}} \mathrm{d}x, \tag{2.2}$$

and the $2/\pi$ *coefficient is replaced with* $1/\pi$ *for* $k = 0$*.*

Proof. See for instance [205, Ch. 3]. □

Apart from the good behavior of truncated orthogonal series with respect to the L_2 norm (via Lemma 2.6), truncated Chebyshev series also behave well in the minimax setting, but we postpone the discussion on the speed of convergence of f_n to Section 3.6.1.

The use of truncated (or projected) Chebyshev series f_n in practice is mitigated by how we can retrieve the a_k coefficients. While for some functions they can be determined explicitly (see for instance [146, Ch. 5.2]), in general eq. (2.2) would need to be evaluated numerically.

Computationally, it usually turns out to be much more practical to instead work with $p_n \in \mathbb{R}_n[x]$ which *interpolates* f at the Chebyshev nodes of the second kind $(\nu_i)_{0 \leqslant i \leqslant n}$. It is generally known as a *Chebyshev interpolant* and it is also uniformly convergent if f is Lipschitz continuous (see discussion from Section 3.6.1). Expressed in the basis of Chebyshev polynomials of the first kind, we have

$$p_n(x) = \sum_{k=0}^{n} c_k T_k(x) \in \mathbb{R}_n[x], \tag{2.3}$$

which is very closely related to f_n. One way to see this is through the so-called *aliasing phenomenon*:

Theorem 2.8 (Aliasing formula for Chebyshev coefficients)**.** *Let* f *be Lipschitz continuous on* $[-1,1]$*. Using the notations from (2.1) and (2.3), we have*

$$\begin{aligned} c_0 &= a_0 + a_{2n} + a_{4n} + \cdots, \\ c_n &= a_n + a_{3n} + a_{5n} + \cdots, \end{aligned}$$

and for $1 \leqslant k \leqslant n-1$*,*

$$c_k = a_k + (a_{k+2n} + a_{k+4n} + \cdots) + (a_{-k+2n} + a_{-k+4n} + \cdots).$$

Proof. See for instance [205, Ch. 4]. □

In fact, the difference between the minimax approximation quality of f_n and p_n over $[-1,1]$ is usually very small in practice, a difference of a factor of 2 between them being frequent [205, Thm. 7.2, Thm. 8.2].

Another way to relate f_n to p_n is that the continuous orthogonality relationship stated in Definition 2.2 can be converted into a discrete orthogonality relationship when Chebyshev polynomials are concerned, by simply replacing the integral with a summation. Concretely, we have:

Proposition 2.9 (Discrete orthogonality of Chebyshev polynomials)**.** *The polynomials* $T_0, \ldots, T_n$ *are orthogonal over the set* $\{\nu_0, \ldots, \nu_n\}$ *of local extrema of* $T_n(x)$ *with respect to the discrete inner product* $\langle u, v \rangle = \frac{1}{2} u(\nu_0) v(\nu_0) + \sum_{k=1}^{n-1} u(\nu_k) v(\nu_k) + \frac{1}{2} u(\nu_n) v(\nu_n)$*, in the sense that*

$$\langle T_i, T_j \rangle = \begin{cases} 0, & i \neq j, \\ n, & i = j = 0 \text{ or } i = j = n, \\ n/2, & 0 < i = j < n. \end{cases}$$

Proof. See [146, Ch. 4.6]. A similar result can also be derived for the Chebyshev nodes of the first kind. □

From this result we can deduce

Proposition 2.10 (Chebyshev interpolation formulas). *If $p_n(x) = \sum_{k=0}^{n} c_k T_k(x)$ is the degree n Chebyshev interpolant of f (*i.e., *$p_n(\nu_k) = f(\nu_k), k = 0, \ldots, n$), then*

$$c_k = \frac{1}{n} f(\nu_0) T_k(\nu_0) + \frac{2}{n} \sum_{i=1}^{n-1} f(\nu_i) T_k(\nu_i) + \frac{1}{n} f(\nu_n) T_k(\nu_n), \qquad k = 0, \ldots, n. \tag{2.4}$$

The c_k coefficients in (2.4) can be evaluated in a numerically stable way with $\mathcal{O}(n^2)$ operations, by using Clenshaw's algorithm $n+1$ times (see Algorithm 1), or faster, in only $\mathcal{O}(n \log n)$ operations, by means of the Discrete Cosine Transform (DCT) [201] or the Fast Fourier Transform (FFT) [146, Ch. 4.7.1], provided that the values $f(\nu_i), i = 0, \ldots, n$ are already given. This ability to compute the coefficients of p_n in a fast and stable manner is one of the key elements which makes Chebfun such a robust numerical library. Working with continuous functions f in Chebfun means replacing them with so-called `chebfun`s, that is Chebyshev interpolants of sufficiently high degree that approximate f close to the unit roundoff of 64-bit `double` floating-point values (see Section 2.3 and the references therein for the relevant definitions and values concerning floating-point arithmetic).

Algorithm 1: CLENSHAW EVALUATION

Input: Chebyshev coefficients $c_0, \ldots, c_n$ and a point x
Output: $\sum_{k=0}^{n} c_k T_k(x)$

$b_{n+1} \leftarrow 0, b_n \leftarrow c_n$
for $k \leftarrow n-1$ ***down to*** 1 **do**
  $b_k \leftarrow 2x b_{k+1} - b_{k+2} + c_k$
return $c_0 + x b_1 - b_2$

Since polynomials expressed in the Chebyshev basis $(T_i)_{0 \leqslant i \leqslant n}$ are used extensively throughout the text, it is perhaps appropriate to also say a few words about evaluating them in practice. The idea goes back to Clenshaw [61] and consists of a recursive method to evaluate linear combinations of polynomials satisfying a three term recurrence. For Chebyshev polynomials, it works by taking advantage of (1.1). Given in Algorithm 1, it can be seen as an extension to Horner's rule for evaluating a polynomial as a sum of powers using nested multiplication and also runs in $\mathcal{O}(n)$ arithmetic operations. Indeed, by the definition of the b_k's, we have

$$\sum_{k=0}^{n} c_k T_k(x) = c_0 + (b_1 - 2x b_2 + b_3) T_1(x) + \ldots + (b_{n-1} - 2x b_n + b_{n+1}) T_{n-1}(x) + c_n T_n(x),$$

which simplifies to $c_0 + b_1 x + b_2 (T_2(x) - 2x T_1(x))$ by using (1.1) and the values of b_n and b_{n+1}. For remarks regarding its stable numerical behavior, we refer the reader to [193] and [146, Ch. 2.4.2].

2.2 Digital filter design

The content of this section is meant to give a very concise overview of some basic notions on digital filters we will be using throughout the text. It is by no means exhaustive and the reader is referred to standard digital signal processing texts like [11, 158] for a systematic and complete presentation of the topic.

Digital Signal Processing gives us tools to analyze, modify or otherwise manipulate information regarding phenomena evolving over time or space. It operates on sequences of complex values called *discrete-time signals*. In a one dimensional setting, such a signal, let us call it x, is usually denoted in signal processing literature as

$$x = \{x[n]\}, \qquad x[n] \in \mathbb{C}, -\infty < n < \infty,$$

where the integer index n is viewed as a measure of dimensionless time, imposing a chronological order on the values of the sequence. One signal that is central to describing digital systems is the *impulse* (or *unit*

sample sequence). It is defined as

$$\delta[n] = \begin{cases} 1, & n = 0, \\ 0, & n \neq 0. \end{cases}$$

Its importance stems from the fact that it can be used to capture the structure of an arbitrary signal x. Generally speaking, any sequence can be expressed as

$$x[n] = \sum_{k=-\infty}^{\infty} x[k]\delta[n-k], \tag{2.5}$$

a convolution operation between the sequence and the impulse, called the reproducing formula.

When working with signals, we have to take into account that, from a physical point of view, they are characterized as having finite energy. The energy of a signal x is given as $E_x = \|x\|_2^2 = \sum_{n=-\infty}^{\infty} |x[n]|^2$. Two spaces of sequences satisfying this property are of interest in practice:

- absolutely summable sequences (the space $\ell^1(\mathbb{Z})$), which are the sequences satisfying:

$$\|x\|_1 = \sum_{n\in\mathbb{Z}} |x[n]| < \infty;$$

- square-summable sequences (the space $\ell^2(\mathbb{Z})$), which have:

$$\|x\|_2^2 = \sum_{n\in\mathbb{Z}} |x[n]|^2 < \infty.$$

The reader can verify that $\ell^1(\mathbb{Z}) \subset \ell^2(\mathbb{Z})$. In the following we will deal only with these types of discrete time signals.

2.2.1 Frequency domain representation

Another important remark regarding real-world signals is that they are often characterized by some sort of oscillatory behavior. In most cases it is very hard to analyze it by studying the sequence representation we used to denote signals. This information can be captured by using Fourier analysis to transform the signal in its frequency domain representation. The Discrete-Time Fourier Transform (DTFT) maps $\ell^1(\mathbb{Z})$ sequences into the space of 2π-periodic functions of a real-valued argument.

Definition 2.11 (DTFT and IDTFT)**.** The **Discrete-Time Fourier Transform** of a signal x belonging to $\ell^1(\mathbb{Z})$ is defined as the 2π-periodic function

$$X(\omega) = \sum_{n=-\infty}^{\infty} x[n]e^{-i\omega n}, i \in \mathbb{C}, i^2 = -1 \qquad \text{and } w \in \mathbb{R}. \tag{2.6}$$

The inverse operation, called the **Inverse Discrete-Time Fourier Transform** (IDTFT) satisfies

$$x[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(\omega)e^{i\omega n}\mathrm{d}\omega.$$

Some changes are needed to make Definition 2.11 work in the $\ell^2(\mathbb{Z})$ setting. Since we will only be using the DTFT on $\ell^1(\mathbb{Z})$ sequences in Ch. 3–5, we refer the reader to [219, Ch. 3.4] for details on the $\ell^2(\mathbb{Z})$ specifics.

In the following, unless specified otherwise, we will usually denote the DTFT of a lowercase letter signal with its capital case equivalent. The usual interval chosen to represent the frequency content (called the *spectrum*) of a signal is $[-\pi, \pi]$, with low frequencies centered around 0 and high ones around $-\pi$ and π.

Energy conservation between the two representations is ensured by means of *Parseval's theorem*, which states the following equality:

$$\sum_{n=-\infty}^{\infty} |x[n]|^2 = \frac{1}{2\pi} \int_{-\pi}^{\pi} |X(\omega)|^2 \, \mathrm{d}\omega.$$

We can think of the magnitude of the DTFT at a certain frequency $\omega \in [-\pi, \pi]$ (the value $|X(\omega)|$) as the energy concentration of the signal at that particular frequency. Thus, by analyzing a signal's spectral magnitude we can decompose discrete-time signals into three broad categories:

- *lowpass* signals: the magnitude spectrum is concentrated around 0 and negligible elsewhere. We say that the passband for the signal is centered around 0, while the stopbands correspond to the remaining regions of $[-\pi, \pi]$;
- *highpass* signals: spectrum concentrated around $\pm\pi$ and negligible elsewhere;
- *bandpass* signals: spectrum concentrated around $\omega = \pm\omega_p$, $\omega_p \in (-\pi, \pi)$, and negligible elsewhere (around $\omega = 0$ and $\omega = \pm\pi$).

2.2.2 Discrete-time systems

We said that signal processing is interested in manipulating information stored in signals. The tool to do so is the *discrete-time system*. Mathematically, we define it as a transformation (or operator) that maps an input sequence with values $x[n]$ into an output sequence with values $y[n]$ ($y[n] = T\{x[n]\}$). A simple example consists of the ideal delay system, expressed using the equation

$$y[n] = x[n - n_d], \qquad -\infty < n < \infty,$$

where n_d is a fixed positive integer representing the delay of the system. The ideal delay operator falls under the category of *linear-time invariant*(LTI) systems (also called *filters*) and they are a core topic in digital signal processing.

Consider an arbitrary LTI discrete-time system $\mathcal{H}$. Its linearity is expressed by

$$\mathcal{H}\{\alpha x_1[n] + \beta x_2[n]\} = \alpha \mathcal{H}\{x_1[n]\} + \beta \mathcal{H}\{x_2[n]\}$$

for any two sequences $x_1[n]$ and $x_2[n]$ and any two scalars $\alpha, \beta \in \mathbb{C}$, while time-invariance implies the equivalence

$$y[n] = \mathcal{H}\{x[n]\} \Leftrightarrow \mathcal{H}\{x[n - n_0]\} = y[n - n_0], \forall n_0 \in \mathbb{Z}.$$

Taken together, these two properties have a very powerful consequence on the system's behavior: it can be completely characterized by its response to the signal $x[n] = \delta[n]$. The sequence $h[n] = \mathcal{H}\{\delta[n]\}$ is called the *impulse response* of the system and it is all we need to know in order to determine the system's response for any other input signal. Indeed, by using the reproducing formula (2.5) and the linearity and time-invariance properties, we have

$$y[n] = \mathcal{H}\{x[n]\} = \mathcal{H}\left\{\sum_{k=-\infty}^{\infty} x[k]\delta[n-k]\right\} = \sum_{k=-\infty}^{\infty} x[k]\mathcal{H}\{\delta[n-k]\} = \sum_{k=-\infty}^{\infty} x[k]h[n-k], \tag{2.7}$$

or, equivalently, in terms of the convolution operator, $y[n] = x[n] * h[n]$. LTI systems also make it easy to work with signals in the frequency domain by means of the *convolution theorem*: if x and h are in $\ell^1(\mathbb{Z})$, then the corresponding DTFTs are linked by $Y(\omega) = X(\omega)H(\omega)$. We call $H(\omega)$ the *frequency response*.

By looking at the magnitude of the frequency response, we can split filters in the same lowpass, highpass and bandpass categories as before. Depending on the type of filter used, the outputs only contain those frequency components that are not eliminated by the filter via the convolution theorem. For example, in the case of highpass filters, only the frequency components centered around $\pm\pi$ are preserved from input to output. The phase content of $H(\omega)$ is also important, especially in data transmission applications. Linear

phase in the frequency domain corresponds to a delay in the time domain, which ensures a predictable behavior (time-wise) for the output.

Other properties which are needed in practice, but not necessarily valid for an LTI system, are: *causality* and *stability*. A causal system is one whose output does not depend on future input values. This is a crucial aspect, especially for systems that work in real time, for which the inputs are not known in advance. Stability, in this setting, states the fact that for any bounded filter input, the output should also be bounded. A necessary and sufficient condition for stability (see [158, Ch. 2.4]) is that the impulse response be absolutely summable.

For realizable filters (*i.e.,* each value of the output can be computed using a finite number of arithmetic operations), equation (2.7) can be reworked into a *constant-coefficient difference equation* (CCDE) like so:

$$y[n] = \sum_{k=0}^{N-1} b_k x[n-k] - \sum_{k=1}^{M-1} a_k y[n-k], \tag{2.8}$$

where the a_k and b_k coefficients are taken to be real (or complex) numbers.

Without going into much detail, equations like (2.8) can be analyzed in an algebraic manner by using the z-transform, which, for a sequence x, is defined as

$$X(z) = \mathcal{Z}\{x[n]\} = \sum_{n=-\infty}^{\infty} x[n] z^{-n}, z \in \mathbb{C}. \tag{2.9}$$

Applying it to (2.8), we get

$$\begin{aligned} Y(z) &= \sum_{k=0}^{N-1} b_k z^{-k} X(z) - \sum_{k=1}^{M-1} a_k z^{-k} Y(z) \\ &= \frac{\sum_{k=0}^{N-1} b_k z^{-k}}{1 + \sum_{k=1}^{M-1} a_k z^{-k}} X(z) \\ &= H(z) X(z). \end{aligned}$$

$H(z)$ is called the *transfer function* of the system described by equation (2.8).

Remark 2.12. The denominator of $H(z)$ is invertible as a power series expansion of z; this allows us to retrieve the $h[k]$ coefficients of (2.7).

A realizable filter is always defined in terms of a rational transfer function. Also, we can see that using $z = e^{i\omega}$ in (2.9) gives us the discrete-time Fourier transform (2.6) of x. The use of the letter z and ω for the variable should make it clear to which of the two transforms we are referring to. There are also different ways of rewriting the transfer function $H(z)$ and this translates to different hardware and software implementations [76, Ch. 12–13].

Based on the number of nonzero values of $h[n]$ (called *taps*), filters can be of two types:

- **FIR** (finite impulse response) filters: the number of taps is finite;
- **IIR** (infinite impulse response) filters: infinite number of taps.

There are different reasons as to why one should use a type of filter over the other (see for instance [169, Ch. 7.1.1]). In essence, the choice comes down to the following observations:

- FIR filters are unconditionally stable, while stability is not guaranteed in the IIR case;
- FIR filters offer precise control over their phase response (*i.e.,* can be easily designed to offer linear phase, as we shall see) whereas IIR filters are more difficult to control;
- IIR filters are computationally less intensive than FIR filters that are designed to have similar behavior.

2.2.3 Minimax FIR filter design

Many FIR filters with real-valued taps (or coefficients) $\{h[k]\}_{k=0}^{N-1}$ used in practice have a frequency response of the form

$$G(\omega) = \sum_{k=0}^{N-1} h[k]e^{-ik\omega} = H(\omega)e^{i\left(\frac{L\pi}{2} - \frac{N-1}{2}\omega\right)}, \tag{2.10}$$

where $H(\omega)$ is a real valued function and L is 0 or 1. The $h[k]$ values are usually required to be positive ($L = 0$) or negative ($L = 1$) symmetric around the middle tap(s), giving rise to so-called *linear-phase* filters. Depending on the type of symmetry and number of non-zero taps, there are 4 major types (I to IV) of such filters [158, Ch. 5.7.3]. We state some important formulas without proof (they can be easily verified) which will show that from a design point of view, all four types can generally be treated in the same way [11, Ch. 15.8–15.9], which is why we will usually focus our attention, without loss of generality, on type I FIR filters.

Type I – positive symmetry and odd number of taps

$$H(\omega) = \sum_{k=0}^{n} a_k \cos(k\omega), \tag{2.11}$$

$n = (N-1)/2, a_0 = h[n]$ and $a_k = 2h[n-k], 1 \leqslant k \leqslant n$.

Type II – positive symmetry and even number of taps

$$H(\omega) = \sum_{k=1}^{n} b_k \cos\left(\left(k - \frac{1}{2}\right)\omega\right), \tag{2.12}$$

$n = N/2$ and $b_k = 2h[n-k], k = 1, \ldots, n$. It can be shown that $H(\omega) = \cos\frac{\omega}{2}H'(\omega)$, where

$$H'(\omega) = \sum_{k=0}^{n-1} \tilde{b}_k \cos(k\omega),$$

$b_1 = \tilde{b}_0 + \frac{1}{2}\tilde{b}_1, b_k = \frac{1}{2}\left(\tilde{b}_{k-1} + \tilde{b}_k\right), k = 2, \ldots, n-1, b_n = \frac{1}{2}\tilde{b}_{n-1}$.

Type III – negative symmetry and odd number of taps

$$H(\omega) = \sum_{k=1}^{n} c_k \sin(k\omega), \tag{2.13}$$

$n = (N-1)/2, c_k = 2h[n-k], k = 1, \ldots, n$ and $h[n] = 0$. Similarly to before, we have $H(\omega) = \sin\omega H'(\omega)$, where

$$H'(\omega) = \sum_{k=0}^{n-1} \tilde{c}_k \cos(k\omega),$$

$c_1 = \tilde{c}_0 - \frac{1}{2}\tilde{c}_1, c_k = \frac{1}{2}\left(\tilde{c}_{k-1} - \tilde{c}_{k+1}\right), k = 2, \ldots, n-2, c_{n-1} = \frac{1}{2}\tilde{c}_{n-2}, c_n = \frac{1}{2}\tilde{c}_{n-1}$.

Type IV – negative symmetry and even number of taps

$$H(\omega) = \sum_{k=1}^{n} d_k \sin\left(\left(k - \frac{1}{2}\right)\omega\right), \tag{2.14}$$

$n = N/2, d_k = 2h[n-k], k = 1, \ldots, n$. We also have $H(\omega) = \sin\frac{\omega}{2}H'(\omega)$, where

$$H'(\omega) = \sum_{k=0}^{n-1} \tilde{d}_k \cos(k\omega),$$

$d_1 = \tilde{d}_0 - \frac{1}{2}\tilde{d}_1, b_k = \frac{1}{2}\left(\tilde{d}_{k-1} - \tilde{b}_k\right), k = 2, \ldots, n-1, d_n = \frac{1}{2}\tilde{d}_{n-1}$.

The filter design question for type I filters which will be central to the next two chapters can be stated as:

Problem 2.13 (Equiripple (or minimax) FIR filter design)**.** *Let Ω be a compact subset of $[0, \pi]$ and $D(\omega)$ an ideal frequency response, continuous on Ω. For a given* filter degree *$n \in \mathbb{N}$, we want to determine $H(\omega) = \sum_{k=0}^{n} a_k \cos(k\omega)$ such that the weighted error function $E(\omega) = W(\omega)(D(\omega) - H(\omega))$ has* minimum *uniform norm*

$$\|E(\omega)\|_{\infty,\Omega} = \sup_{\omega \in \Omega} |E(\omega)|,$$

where the weight function W is continuous and positive over Ω.

Once a suitable solution is found, the filter taps can be determined directly from the a_i coefficients inside (2.11). Based on the previous formulas, type II to IV problems where $H(\omega)$ has a different expression can also be cast in this way, by optimizing for $H'(\omega)$ instead. The error function becomes

$$E(\omega) = \underbrace{W(\omega)Q(\omega)}_{W'(\omega)} \left(\underbrace{\frac{D(\omega)}{Q(\omega)}}_{D'(\omega)} - H'(\omega) \right),$$

where $Q(\omega)$ is either $\cos\left(\frac{\omega}{2}\right), \sin(\omega)$ or $\sin(\frac{\omega}{2})$, depending on the filter type, with the remark that for type II problems $\Omega \subseteq [0, \pi)$, for type III problems $\Omega \subseteq (0, \pi)$ and for type IV problems $\Omega \subseteq (0, \pi]$ so that $Q(\omega)$ is always nonzero.

2.3 Computer arithmetic

Representing and working with real numbers inside a computer is very much restricted by the finite nature of electronic memory elements. This has led over the years to many different alternatives for representing/approximating a infinite set of real values with a finite set of so-called *machine numbers*. Among the most well known ones we mention: floating-point arithmetic, fixed-point arithmetic, logarithmic number systems, continued fractions and rational numbers. In choosing the most adequate format for a given situation, many constraints, such as accuracy, speed, dynamic range, memory cost, ease of use and implementation, factor in. Floating-point arithmetic is by far the most popular alternative, since it provides a good compromise for all these constraints. Still, in the embedded circuit world, fixed-point arithmetic is still very much the dominant format, due to its simplicity, speed and ease of use (but which usually comes with a diminished range of values, when compared to floating-point arithmetic).

These two formats play an important role in this manuscript, which is why over the next few paragraphs, we will give a brief overview of them, so as to establish some key concepts and notation. For floating-point arithmetic in particular, we refer the reader to [153] for a much more in-depth treatment of the subject. Our exposition is heavily based on [153, Ch. 1–3].

2.3.1 Floating-point arithmetic

A general purpose definition of what a floating-point value is, can be given as:

Definition 2.14 (Floating-point number)**.** A floating-point number belonging to a radix-β ($\beta \in \mathbb{N}, n \geqslant 2$), precision-$p$ ($p \in \mathbb{N}, p \geqslant 2$) format, is a number of the form

$$x = (-1)^s \cdot m \cdot \beta^e,$$

where

- $s \in \{0, 1\}$ is the sign bit of x;

Table 2.1: Parameters of the binary floating-point formats from the 754-2008 standard [107].

Name	binary16	binary32 (single precision)	binary64 (double precision)	binary128 (quad precision)
p	11	24	53	113
$e_{\max}$	+15	+127	+1023	+16383
$e_{\min}$	-14	-126	-1022	-16382

- $m = |M| \cdot \beta^{1-p}$ is called the *normal significand* (or *mantissa*). M is an integer, called the *integral significand*, represented in radix β, $|M| \leqslant \beta^p - 1$. This means that m has one digit before the radix point and at most $p-1$ digits after it; and
- e is called the *exponent*, an integer such that $e_{\min} \leqslant e \leqslant e_{\max}$, where the bounding exponents $e_{\min} < 0 < e_{\max}$ are given.

This way of representing floating-point numbers does not preclude some values from having several representations in the same format. In order to prevent this, we *normalize* finite nonzero floating-point values by choosing the representation for which the exponent is minimum (but still larger than $e_{\min}$), giving rise to two kinds of floating-point numbers:

- *normal* numbers: $1 \leqslant |m| < \beta$, or, equivalently $\beta^{p-1} \leqslant |M| < \beta^p$;
- *subnormal* numbers: $e = e_{\min}$ and $|m| < 1$, or, equivalently, $|M| \leqslant \beta^{p-1} - 1$. Zero is a special case, treated separately [153, Ch. 3].

The presence of subnormal numbers is important in practice, since it allows for *gradual overflow*, which, according to [153, Ch. 2] aids significantly in the writing of stable numerical software. In terms of radix β, based on the fact that most computers use two-state logic, radix 2 systems (and more generally, power of 2 radixes) are most common. However, due to our familiarity with decimal arithmetic, radix 10 systems are often used in financial computations and in pocket calculators.

In order to ensure a setting for performing reliable, accurate and portable floating-point computations, in 1985, the IEEE754-1985 Standard for Binary Floating-Point Arithmetic was introduced [106], which is nowadays implemented by basically all commercially significant computing systems. Its most recent revision, called IEEE754-2008 (which also includes specifications for decimal floating-point arithmetic), was adopted in June 2008 [107]. The IEEE floating-point standard specifies various formats, the behavior of basic arithmetic operations (such as $+, -, \times, \div$ and $\sqrt{}$), conversions and exceptional conditions. The parameters for the binary formats specified in the 2008 version of the standard are given in Table 2.1.

An interesting concept brought about by the standardization effort is the idea of *rounding mode*, which helps define what value an operation (or a function) involving floating-point operands outputs, when the result is not exactly representable in the floating-point system being used. There are four such modes defined by the IEEE754-2008 standard:

- round toward $-\infty$: $\mathrm{RD}(x)$ is the largest floating-point number (or $-\infty$) less than or equal to x;
- round toward $+\infty$: $\mathrm{RU}(x)$ is the smallest floating-point number (or $+\infty$) larger than or equal to x;
- round toward zero: $\mathrm{RZ}(x)$ is the closest floating-point number to x that is no greater in magnitude than x (it is equal to $\mathrm{RD}(x)$ if $x \geqslant 0$, and to $\mathrm{RU}(x)$ if $x \leqslant 0$);
- round to nearest: $\mathrm{RN}(x)$ is the floating-point number closest to x. A tie-breaking rule is applied when x falls exactly halfway between two consecutive floating-point numbers. A frequently chosen rule is *round to nearest even*: x is rounded to the only one of these two competing floating-point values whose integral significand is even. This is the default mode of the IEEE754-2008 standard.

Unless otherwise stated, in the sequel we will consider the round to nearest setting when talking about floating-point computations. To measure rounding errors in floating-point arithmetic, it is often convenient

to express them in what would intuitively be defined as "weight of the last bit of the significand" or *unit in the last place* (ulp) (see [153, Ch. 2.6] for a detailed account). In a loose manner, ulp(x) is the distance between the two consecutive floating-point bounding x. Ambiguity ensues when x is close to a power of the radix 2, and there are several slightly different definitions one can use (due for instance to W. Kahan, J. Harrison and D. Goldberg). For instance, we have

Definition 2.15 (Goldberg's ulp)**.** If $x \in [2^e, 2^{e+1})$, then $\text{ulp}(x) = 2^{e-p+1}$.

Closely related to the notion of ulp is the *unit roundoff*, also called *machine epsilon*:

Definition 2.16 (Unit roundoff)**.** The *unit roundoff* u of a radix-β, precision-p, floating point system is defined as

$$u = \begin{cases} \frac{1}{2}\text{ulp}(1) & = \frac{1}{2}\beta^{1-p} \text{ in round-to-nearest mode,} \\ \text{ulp}(1) & = \beta^{1-p} \text{ in directed rounding modes.} \end{cases}$$

The *unit roundoff* is widely used in the analysis of numerical algorithms. For any basic arithmetic operation $T \in \{+, -, \times, \div\}$, any rounding mode $\circ \in \{\text{RN,RD,RU,RZ}\}$ and for all floating point values x, y such that xTy does not underflow or overflow, we get

$$\circ(xTy) = (xTy)(1 + \varepsilon_1) = (xTy)/(1 + \varepsilon_2),$$

with $|\varepsilon_1|, |\varepsilon_2| \leqslant u$. This formula is very useful in the computation of error bounds [102]. We will sometimes use the unit roundoff as well, when discussing numerical stability issues in Chapter 3.

2.3.2 Fixed-point arithmetic

Fixed-point values play an important role in Chapter 4 and are in general simpler to work with than floating-point values, essentially because they are just scaled integers. Because of this, fixed-point arithmetic does not need specialized arithmetic circuits (as is the case for floating-point formats), apart from a usual integer arithmetic unit. They are mainly used in embedded circuits for signal processing tasks, where power consumption constraints make floating-point formats a less attractive proposition.

There is no standard formalization of fixed-point values (in the same vein of the IEEE754 standard for floating-point arithmetic). Hence, there are numerous definitions and notations for fixed-point values, but they can generally be described by:

Definition 2.17 (Fixed-point number)**.** For a given real scaling factor $s > 0$ and integer mantissa bounds $m_- < m_+$, a fixed-point value is any number

$$x = M \cdot s,$$

where $M \in [m_-, m_+] \cap \mathbb{Z}$ is called the *integer mantissa* of x.

In many cases, $s = 2^{-\ell}, \ell \in \mathbb{Z}$, meaning $-\ell$ designates the position of the *least significant bit* (lsb) in the binary encoding of x. The mantissa M is also representable on a finite number of bits. If, for instance $M \in [-2^b + 1, 2^b - 1] \cap \mathbb{Z}$, then $b + 1$ will be the *width* of the format (*i.e.,* the number of bits necessary to encode a value belonging to this fixed-point format), with $b - \ell - 1$ representing the position of the *most significant bit* in the signed binary encoding of x, meaning

$$x = s \sum_{i=-\ell}^{b-\ell-1} x_i 2^i, \tag{2.15}$$

where $s \in \{-1, 0, 1\}$ ($s = 0$ when $x = 0$) and $x_i \in \{\pm 1\}$.

The name fixed-point comes from the fact that the scaling factor s is a fixed quantity, whereas if we look at floating-point values in Definition 2.14, the equivalent to s is β^{e+1-p}, which varies with e.

Besides (2.15), there are many other encodings one can use to represent x, like one and two's complement, which affect the actual range of M. Another useful rewriting used in practice is the canonic signed digit (CSD) representation, which consists of rewriting (2.15) such that $s = 1$ and $x_i \in \{-1, 0, 1\}$ with $x_i x_{i+1} = 0$, for $i = -\ell, \ldots, b - \ell - 2$. It can be derived recursively from (2.15) by noting the k ones in succession in the binary representation of equals $2^k - 1$, which can be rewritten using two nonzero digits and $k - 1$ zeros in a CSD form. Because they often lead to very compact encodings, CSD representations are mainly used in hardwired signal processing operations which lack specialized circuits for performing integer multiplication. Since the specifics of the fixed-point representations mentioned here are not important in the sequel, we will not go into further details.

In contrast to the floating-point setting, there is no real standard for fixed-point arithmetic. The notion of ulp however, can be carried over from floating-point arithmetic very easily; it is the scaling factor s. In terms of rounding as well, we can similarly define rounding towards $\pm\infty$ and round to nearest operations. For our purposes, we will only be considering rounding to nearest explicitly (which we will sometimes call *naive rounding*). For a given x we adopt the convention that its rounding to nearest in the fixed-point format introduced by Definition 2.17 is

$$\mathrm{RN}(x) = \left\lfloor \frac{x}{s} \right\rceil \cdot s, \tag{2.16}$$

which makes sense only when $\left\lfloor \frac{x}{s} \right\rceil \in [m_-, m_+] \cap \mathbb{Z}$.

3

Efficient tools for weighted minimax approximation

In a subject like approximation theory which is very strongly influenced by our need to solve practical problems of computation it seems appropriate to begin by considering some concrete examples of such problems.

E. W. Cheney, Introduction to Approximation Theory

Chebyshev approximation problems are one of the pillars of Approximation Theory and are routinely at the heart of many engineering applications. As discussed in the previous chapters, one of the most telling such use cases is digital filter design, which serves as the catalyst for the contributions of this thesis.

Algorithmically, it was Remez, in the 1930s [176, 177], that spearheaded the great deal of research on the computational aspects of minimax approximation problems in the decades that followed. A pivotal development in this story occurred in the beginning of the 1970s, when Parks and McClellan [164] adapted one of Remez's algorithms for the design of finite impulse response filters. Today, their approach is a standard routine in many Signal Processing packages. Despite this, it is possible to find practical design specifications where existing codes fail to work or they produce incorrect results.

Our goal in this chapter is twofold. We first examine and present novel solutions for the practical difficulties related to weighted minimax polynomial approximation problems on multi-interval domains (*i.e.,* the general setting under which the Parks and McClellan approach operates). Using these ideas, we then describe a robust implementation of one of Remez's algorithms, which, in particular, routinely outperforms existing minimax filter design routines.

At this point in time, the result is an effective, standalone C++ library, available under an open source license. The contents presented in this chapter are an extended version of our published work [82].

3.1 Introduction

Current implementations of the Parks-McClellan algorithm suffer from robustness issues. A typical instance where codes such as those found in MATLAB's Signal Processing toolbox, GNURadio or Scipy fail to work, is when computing a minimax filter of degree $n = 100$ adhering to the specification from Example 3.19. Such a degree is plausible in practice, yet it proves difficult to design. This unsatisfactory situation was the initial motivation of the present study. Together with the three other examples from Section 3.4.2, it will be used extensively to highlight the various contributions of this chapter.

The starting point of our work is [163], which describes a robust double-precision implementation of the second Remez exchange algorithm for computing best polynomial approximations on a single *compact interval*, as part of Chebfun (the `remez` command). The main elements that allow Chebfun `remez` to perform well in practice are:

(1) the use of Chebyshev nodes as starting reference;

(2) barycentric Lagrange interpolation [24];

(3) a root-finding method [34, 205] based on the recursive subdivision of the approximation domain.

The Parks-McClellan variant computes minimax approximations in a *weighted multi-interval* context and can therefore be viewed as an extension of the case addressed by [163]. Unfortunately, adapting this implementation to our context is not at all straightforward and raises several significant issues. The present work addresses all of them and, to the best of our knowledge, it describes the first fast, robust and scalable implementation of the Parks-McClellan algorithm. Note also that, despite the fact that most of our examples come from filtering applications (the traditional setting for the Parks-McClellan algorithm), our code can compute more general weighted minimax polynomial approximations over a multi-interval compact subset of $\mathbb{R}$.

We begin with an overview of linear Chebyshev approximation problems (Section 3.2), the algorithmic approaches devised to solve them (Section 3.3) and the practical difficulties in using them (Sections 3.4 and 3.5). The main problem we faced is finding a *suitable* initial reference. In particular, we do not know of any *simple*, *closed-form* equivalent for Chebyshev nodes in this setting (in the sense that interpolation at those points is *close* to the optimal approximation). Fast practical convergence of the exchange algorithm is very much dependent on placing the right number of reference points inside each interval of the domain. In Section 3.6, we describe two *complementary* heuristic solutions to this initialization problem. Our experiments show that they are quite satisfactory in practice. Section 3.7 looks at how initialization choices affect execution. We also identify possible sources of numerical instability at runtime, argue how they are related to each other and how to *counter* them. Central to our study is the *Lebesgue constant* (see Section 3.6.1 for a definition) that we use, in an apparently novel way, as a tool for detecting numerical problems. Another important difference with the Chebfun `remez` code is the fact that we eliminate recursion during root-finding by using problem-specific information about the root positions. Major advantages are a smaller number of computations compared to a recursive search and the possibility of introducing parallelism (see Sections 3.8 and 3.9), further speeding up execution on multicore systems.

The resulting code[1] allows one to successfully address problematic instances in a scalable manner (see the examples from Sections 3.4.1 and 3.9.2). We offer three versions of our implementation. The first one uses IEEE-754 double-precision floating-point arithmetic, while the second one requires `long double` operations. On x86 machines and compilers, this format corresponds to 80-bit floating-point numbers. The other version uses `MPFR` [84] multiple precision formats and serves mostly as a verification tool for our `double` and `long double` versions. All of the numerical tests in the sequel (and many others) are included with the source code. The double-precision version is sufficient for *most* practical uses.

3.2 Properties of best approximations

Polynomials are attractive to use in practice because of their simple structure which can be implemented on a computer using only a linear number (in the degree of the polynomial) of additions and multiplications, the two fastest and most optimized arithmetic operations on modern processors. Another major reason for using polynomials to approximate continuous real-valued functions is due to the Stone-Weierstrass theorem:

Theorem 3.1 (Stone–Weierstrass)**.** *A continuous function $f(\mathbf{x})$, defined on a compact subset X of $\mathbb{R}^d$, is uniformly approximable over X by polynomials in $\mathbf{x} \in \mathbb{R}^d$.*

Proof. See [75, Ch. 1.3–1.4] for a proof and further extensions. □

This result basically tells us that we can approach f uniformly using a polynomial $p \in \mathbb{R}[x_1, \ldots, x_d]$ arbitrarily well, provided the degree of p is sufficiently large.

We will mostly focus on univariate problems ($d = 1$) in what follows, where X is compact subset of an interval $[a, b]$. Using weight functions when performing the approximation (as is frequently the case with the

[1] See https://github.com/sfilip/firpm for the accompanying C++ code

problems we are considering) can invalidate the result of Theorem 3.1. This is not generally an issue, since our weight functions $w \in \mathcal{C}(X)$ satisfy $w(x) > 0, \forall x \in X$, for which it is immediate to see the result still holds.

Weights can be taken into account more easily if we replace, for a degree $n \in \mathbb{N}$ problem, the space of polynomials, given for instance in the monomial basis $1, x, \ldots, x^n$, by a set of $n+1$ linearly independent functions $\varphi_0, \varphi_1, \ldots, \varphi_n$ from $\mathcal{C}([a,b])$. Following [56, Ch. 3.4], we will call any linear combination of φ_i (the finite-dimensional linear space $S_n = \operatorname{span}_{\mathbb{R}}(\varphi_i)_{0 \leqslant i \leqslant n}$) a *generalized polynomial* (or even simply *polynomial* when we see no risk of confusion). The existence of an element $p^\star \in S_n$ which best approximates f in a L_∞ sense is guaranteed by:

Theorem 3.2 (Existence of best approximations)**.** *For any $f \in \mathcal{C}(X)$, there exists a generalized polynomial $p^\star \in S_n$ such that*

$$\|f - p^\star\|_{\infty,X} \leqslant \|f - p\|_{\infty,X},$$

for all $p \in S_n$. This is frequently called a best, uniform, Chebyshev *or* minimax *approximation.*

Proof. This is a classic result. For a proof, see for instance [56, Ch. 1.6] or [168, Thm. 1.2]. □

3.2.1 Unique best approximations: the Haar condition

The space of polynomials with real coefficients $\mathbb{R}[x]$ plays a central role in Approximation Theory and many of their properties can be extended to more general settings. We shall thus start our discussion by introducing the concept of *Chebyshev system*, that is, a family of functions which satisfy the following condition:

Definition 3.3 (Haar condition)**.** Consider $n+1$ functions $\varphi_0, \ldots, \varphi_n$ defined over $[a,b]$. We say that $\varphi_0, \ldots, \varphi_n$ satisfy the *Haar condition* iff

a) $\varphi_0, \ldots, \varphi_n$ are continuous on $[a,b]$;

b) the following equivalent statements hold:

(i) any $p = \sum_{k=0}^{n} \alpha_k \varphi_k \neq 0$ has at most n distinct zeros;

(ii) if $p \in \operatorname{span}_{\mathbb{R}}(\varphi_i)_{0 \leqslant i \leqslant n}$ and $p \neq 0$, then if the number of zeros of p inside $[a,b]$ is equal to j and k of these zeros are points inside (a,b) where p does not change sign, then $j + k < n + 1$;

(iii) if $x_0, x_1, \ldots, x_n$ are $n+1$ arbitrary distinct points from $[a,b]$, then the matrix

$$\begin{bmatrix} \varphi_0(x_0) & \cdots & \varphi_n(x_0) \\ \vdots & \vdots & \vdots \\ \varphi_0(x_n) & \cdots & \varphi_n(x_n) \end{bmatrix}$$

is non-singular.

A proof of the equivalence of (i)–(iii) can be found in [168, Appx. A].

As hinted, the prototypical example of a system satisfying the Haar condition is the set of polynomials $\mathbb{P}_n(\mathbb{R})$ of degree at most n. The following basic result allows us to build other examples very quickly:

Lemma 3.4. *If $(\varphi_0, \ldots, \varphi_n)$ is a Chebyshev system on $[a,b]$ and $w \in \mathcal{C}([a,b])$ is positive on $[a,b]$, then $(w\varphi_0, \ldots, w\varphi_n)$ is also a Chebyshev system on $[a,b]$.*

Proof. Consider $q \in \operatorname{span}_{\mathbb{R}}(w\varphi_i)_{0 \leqslant i \leqslant n}, q \neq 0$. We can write $q = wp$, with $p \in \operatorname{span}_{\mathbb{R}}(\varphi_i)_{0 \leqslant i \leqslant n}, p \neq 0$. Since $(\varphi_0, \ldots, \varphi_n)$ satisfies the Haar condition, p has at most n distinct roots on $[a,b]$. Because w is positive on $[a,b]$, this property is true for q as well, meaning the Haar condition also holds for $(w\varphi_0, \ldots, w\varphi_n)$. □

Example 3.5. *Let $w \in \mathcal{C}([a,b])$ be a function which does not have any zeros on $[a,b]$. Then, by the previous lemma, $(w, wx, \ldots, wx^n)$ satisfies the Haar condition. If f has no zeros on $[a,b]$, then the relative error minimax polynomial approximation of f on $X \subseteq [a,b]$ is equivalent to the minimax approximation of $g(x) = 1$ on X using the basis $(1/f(x), x/f(x), \ldots, x^n/f(x))$.*

Example 3.6. *If f has a unique zero $x_0 \in [a,b]$ of order k, then the system $\left(\frac{(x-x_0)^k}{f(x)}, \ldots, \frac{(x-x_0)^n}{f(x)}\right)$ satisfies the Haar condition.*

Example 3.7. *Consider the following:*

(a) the system $(1, \cos(x), \sin(x), \cos(2x), \sin(2x), \ldots, \cos(nx), \sin(nx))$ of trigonometric polynomials verifies the Haar condition on all intervals $[a,b]$ such that $0 \leqslant a < b < 2\pi$;

(b) the Haar condition is valid for $(\sin(x), \sin(2x), \ldots, \sin(nx))$ over $(0,\pi)$;

(c) $(1, \cos(x), \cos(2x), \ldots, \cos(nx))$ satisfies the Haar condition on $[0,\pi]$.

Remark 3.8. The system of Example 3.7 (c) is generally the one which is used for minimax FIR filter design, and as such it will be central to us in the sequel. By the change of variable $y = \cos(x)$, it is equivalent to taking the space generated by the Chebyshev polynomials of the first kind up to degree n, $(T_i(y))_{0 \leqslant i \leqslant n}$, on $[-1,1]$. Other examples can also be found in [75, Ch. 3.3] and [29, Ch. 3].

What is noteworthy about polynomial systems which satisfy the Haar condition is that they ensure *uniqueness* of the solution to the Chebyshev approximation problem. Indeed, the following *qualitative* description holds:

Theorem 3.9 (Equioscillation of best approximations)**.** *Let $(\varphi_0, \ldots, \varphi_n)$ be a Chebyshev system on $[a,b]$ with $X \subseteq [a,b]$ a closed set containing at least $n+2$ points. If $f \in \mathcal{C}(X)$, then $p^\star \in S_n$ is the unique best approximation of f on X if and only if there are $n+2$ values $x_0 < \ldots < x_{n+1}$ inside X and $\lambda \in \{\pm 1\}$ for which*

$$f(x_i) - p^\star(x_i) = \lambda(-1)^i \delta_X, \quad i = 0, \ldots, n+1 \tag{3.1}$$

and $\delta_X = \|f - p^\star\|_{\infty,X}$.

Proof. For this particular formulation, see for instance [56, Ch. 3.4] or [75, Ch. 3.5]. □

Traditionally known as the *Chebyshev theorem*, this result was in fact not proven by Chebyshev himself, although some of the underlying ideas are present in his text [203]. Such work was done in the beginning of the 20th century by Blichfeldt [26], Kirchberger [118] and Borel [28] in the context of best polynomial approximations on a closed interval $[a,b]$. See also [205, Ch. 10] for a very interesting exposition on this result.

The points x_i in (3.1) are known as *alternation points* of the minimax result $p^\star$ and are characterized by the fact that error function $f - p^\star$ attains its global extremum there. In a more general setting where a polynomial $p \in S_n$ has $n+2$ alternating points which do not oscillate with equal amplitude, de la Vallée Poussin [72] showed how p can be used to provide a lower bound on the value of δ_X:

Theorem 3.10 (de La Vallée Poussin)**.** *Under the same conditions as Theorem 3.9, if for a generalized polynomial $p \in S_n$ there is an ordered set of $n+2$ distinct nodes $(x_i)_{0 \leqslant i \leqslant n+1}$ from X such that $f(x_i) - p(x_i) = \lambda(-1)^i \mu_i$, with $\lambda = \pm 1$ and $\mu_i > 0, i = 0, \ldots, n+1$, then $\|f - p^\star\|_{\infty,X} \geqslant \min_i \mu_i$.*

Proof. Assume $\|f - p^\star\|_{\infty,X} < \min_i \mu_i$. Then, $\operatorname{sgn}[p^\star(x_i) - p(x_i)] = \operatorname{sgn}[(f(x_i) - p(x_i)) - (f(x_i) - p^\star(x_i))] = \lambda(-1)^i$, meaning $p - p^\star$ has $n+1$ sign changes in $[a,b]$, which, according to the Haar condition, gives $p = p^\star$, a contradiction. □

As an immediate consequence, we have the following:

Corollary 3.11. *If in Theorem 3.10, all μ_i are equal to each other (call this value μ), then*

$$\mu \leqslant \|f - p^\star\|_{\infty,X} \leqslant \|f - p\|_{\infty,X}.$$

Proof. The first inequality is derived from Theorem 3.10, whereas the optimality of $p^\star$ gives the second. □

These last results are very important in the design of numerical algorithms for minimax approximation (as will be detailed in Section 3.3). Indeed, if p and μ are given, we can generally compute very accurate approximations of $\|f - p\|_{\infty,X}$ (see Section 3.8 and the references therein). If this value is sufficiently close to μ, then, for all practical purposes, we can consider p to be optimal.

3.2.2 When uniqueness is no longer guaranteed

The Haar condition is quite general, in the sense that it holds for many situations arising in practice. For FIR filter design, by looking at the existing Signal Processing literature, this seems to always be the case.

Because of this, its absence has not been a very active concern in the work detailed in this chapter. Nevertheless, as presented in [58, Ch. 2.2], the absence of the Haar condition in the context of function approximation is very much possible. For an in-depth presentation and several relevant practical examples we recommend [58, Ch. 2.2].

Of note is the fact that, even when the Haar condition is not met, a certain generalization of de la Vallée Poussin's theorem which proves useful in a computational setting (an example of its use is in the Sollya [60] software tool for the `remez` command), still holds:

Corollary 3.12. *Let $X \subseteq [a,b]$ be a set containing at least $n+2$ distinct points $x_0, \ldots, x_{n+1}$, S_n the finite-dimensional space of generalized polynomials we have considered up to now and $f \in \mathcal{C}(X)$. Suppose only that the matrix $\mathbf{V}(x_0, \ldots, x_{n+1}) = [v_{ij}] := [\varphi_j(x_i)], i = 0, \ldots, n+1, j = 0, \ldots, n$ has full rank $n+1$. If p is a best approximation from S_n on the given $n+2$ points and $\mu = \max_{0 \leqslant i \leqslant n+1} |f(x_i) - p(x_i)|$, then for any polynomial $p^\star$ of best approximation to f on X, we have*

$$\mu \leqslant \|f - p^\star\|_{\infty,X} \leqslant \|f - p\|_{\infty,X}.$$

Proof. See results and discussion from [58, Ch. 2.2.3–2.2.4]. □

3.3 Minimax exchange algorithms

The two main algorithmic approaches for solving linear Chebyshev problems have their origin in the 1930s with the work of Remez [176, 177] and are traditionally known as the "first" and "second" algorithms of Remez. Out of the two, the second one has garnered more notoriety over the years, in particular due to its wide adoption in Signal Processing, where it is more commonly known as the Parks-McClellan algorithm [164]. For completeness, we will present both methods, although our focus will be on the second Remez algorithm.

3.3.1 The second Remez algorithm

Let us give a broad overview of the inner workings of these iterative methods. The second Remez algorithm applies to Chebyshev families of functions and thus exploits the alternation property of the de La Vallée Poussin theorem. Essentially, it consists of retrieving a sequence of approximations with alternating error functions, that ultimately converge to the target minimax solution.

Consider the listing of Algorithm 2. To initialize the procedure, we can take $n+2$ arbitrary distinct points $\{x_i^1\}_{i=0}^{n+1}$ from X which will be used to determine the starting approximation $p_1 \in S_n$ to f (Step 1). Each collection of $n+2$ such points appearing during the course of the algorithm is called a *reference*. At every iteration k, the current approximant p_k is the solution of the minimax approximation problem on the current reference $\{x_i^k\}_{i=0}^{n+1}$ (Step 2). Because of Theorem 3.9, $f - p_k$ will alternate, with equal modulus $|h_k|$, according to (3.2). Provided we are not already very close to the minimax result, we construct a new reference $\{x_i^{k+1}\}_{i=0}^{n+1}$ from the old one by a so-called *exchange* procedure.

If this exchange satisfies (3.3) (*i.e.*, one of the new reference points x_i^{k+1} has error of amplitude $\|f - p_k\|_{X,\infty}$ and $|h_k| < \|f - p_k\|_{X,\infty}$), then we are certain to have $|h_k| < |h_{k+1}|$. To show this, we need the following:

Lemma 3.13. *Let $e \in \mathcal{C}([a,b])$ and $\{x_i\}_{i=0}^{n+1}$ a reference, such that*

$$(-1)^i e(x_i) \geqslant 0, \qquad i = 0, \ldots, n+1.$$

Then e has at least $n+1$ zeros in $[a,b]$, provided that any double zero is counted twice. A double zero is any zero strictly inside $[a,b]$ where e does not change sign.

Proof. See [168, Thm. 7.5]. □

Algorithm 2: Second Remez algorithm

Input: degree $n \in \mathbb{N}$, compact subset X of $[a,b]$, $f \in \mathcal{C}(X)$, a Chebyshev system $S_n = \operatorname{span}_{\mathbb{R}}(\varphi_i)_{0\leqslant i\leqslant n}$ over $[a,b]$, convergence parameter threshold $\varepsilon_t > 0$

Output: an element $p(\boldsymbol{\alpha}, x) = \sum_{k=0}^{n} \alpha_k \varphi_k(x) \in S_n$, where $\boldsymbol{\alpha} \in \mathbb{R}^{n+1}$, close to optimal

Step 1 (Initialization) Set $k = 1$ and choose the starting *reference* set $\left\{x_i^1\right\}_{i=0}^{n+1}$ from X such that

$$a \leqslant x_0^1 < x_1^1 < \ldots < x_{n+1}^1 \leqslant b.$$

Step 2 Determine $p_k \in S_n$ and $h_k \in \mathbb{R}$ (the *levelled error* at iteration k), satisfying

$$e_k(x_i^k) = f(x_i^k) - p_k(x_i^k) = (-1)^i h_k, i = 0, \ldots, n+1. \tag{3.2}$$

Step 3 Construct the next reference $a \leqslant x_0^{k+1} < x_1^{k+1} < \ldots < x_{n+1}^{k+1} \leqslant b$ from X, such that

$$\lambda(-1)^i (f(x_i^{k+1}) - p_k(x_i^{k+1})) \geqslant |h_k|, i = 0, \ldots, n+1, \tag{3.3}$$

where $\lambda \in \{\pm 1\}$, and for one $i \in \{0, \ldots, n+1\}$, the left hand side of (3.3) equals $\|f - p_k\|_{\infty,X}$.

Step 4 By Theorem 3.10, $|h_k| \leqslant \|f - p^\star\|_{\infty,X} \leqslant \|f - p_k\|_{\infty,X}$. Unless the approximation is good enough (*i.e.*, $\frac{\|f-p_k\|_{\infty,X} - |h_k|}{\|f-p_k\|_{\infty,X}} \leqslant \varepsilon_t$, in which case **return** $p_k(x) = p(\boldsymbol{\alpha}, x) = \sum_{i=0}^{n} \alpha_i \varphi_i(x)$), increment k and go to **Step 2**.

We thus have:

Theorem 3.14. *During the execution of Algorithm 2, if at a certain iteration, $p_k \in S_n$ is not the best approximation to f on X, then there is a strict increase of the levelled reference error at the next iteration:*

$$|h_k| < |h_{k+1}|.$$

Proof. Suppose that $|h_k| \geqslant |h_{k+1}|$. Because of how the new reference is chosen, we have

$$\left|e_{k+1}(x_i^{k+1})\right| \leqslant \left|e_k(x_i^{k+1})\right|, \qquad i = 0, \ldots, n+1, \tag{3.4}$$

with at least one of the inequalities being strict. Looking at the set

$$\left\{e_k(x_i^{k+1}) - e_{k+1}(x_i^{k+1})\right\}_{i=0}^{n+1} = \left\{p_{k+1}(x_i^{k+1}) - p_k(x_i^{k+1})\right\}_{i=0}^{n+1},$$

we see from expression (3.4) that

$$(-1)^{i+\ell}(p_{k+1}(x_i^{k+1}) - p_k(x_i^{k+1})) \geqslant 0, \ell \in \{0,1\}, i = 0, \ldots, n+1,$$

which, based on the previous lemma, tells us that $p_{k+1} - p_k$ has at least $n+1$ zeros in $[a,b]$, provided that each double zero is counted twice. Since S_n satisfies the Haar condition and property (ii) of Definition 3.3 holds for $p_{k+1} - p_k$, this leads to $p_k = p_{k+1}$. In particular, $\left|e_{k+1}(x_i^{k+1})\right| = \left|e_k(x_i^{k+1})\right|$ for $i = 0, \ldots, n+1$, but since one of the inequalities of (3.4) is strict, we have a contradiction.

□

Remark 3.15. The condition for the error of one of the new reference values to have magnitude $\|f - p_k\|_{X,\infty}$ in (3.3) can be relaxed for the previous result. Indeed, it is sufficient that there is at least one new reference element x_i^{k+1} satisfying $\left|f(x_i^{k+1}) - p_k(x_i^{k+1})\right| > |h_k|$ for Theorem 3.14 to hold.

Theorem 3.14 by itself is sufficient to show that Algorithm 2 converges. But to ensure that it converges to the minimax result (*i.e.*, $\lim_{k\to\infty} |h_k| = \|f - p^\star\|_{\infty,X}$), two exchange strategies are widely considered in the literature for Step 3. The first one, a *one-point exchange* approach, changes at each iteration one appropriate reference point by bringing in one where the norm of the error is attained. A nice proof of its effectiveness is given by Powell in [168, Ch. 9.1–9.2] when $X = [a, b]$, and we leave it to the interested reader to check that the reasoning used there works identically when we more generally have a compact $X \subseteq [a, b]$.

Traditionally, most of the computational work done during an iteration of Algorithm 2 goes into finding the global maximum of $|e_k(x)|$ over X. This normally involves computing all the local extrema of e_k in X. Thus, it makes sense to consider a *multi-point exchange* strategy: swap all $n+2$ current reference points by $n+2$ local extrema of e_k with sign alternating error values, where, for one of the new points, the modulus of the error attains its global maximum over X (see for instance [217] or [56, Ch. 3.8] for the precise exchange rules when $X = [a, b]$, but also Section 3.8.3). A proof of convergence to the minimax result, again when $X = [a, b]$, is given by Cheney [56, Ch. 3.8]. In terms of speed of convergence, his proof establishes a linear convergence to the minimax result. Veidinger [217] shows it to be in fact quadratic, provided that f and $\varphi_0, \ldots, \varphi_{n+1}$ are twice differentiable and that the minimax error function $e^\star(x) = f(x) - p^\star(x)$ equioscillates between its extrema values *exactly* $n+2$ times on $[a, b]$. This leads, at successive iterations k and $k+1$, to:

$$\|e_{k+1}\|_{\infty,[a,b]} - \|e^\star\|_{\infty,[a,b]} = \mathcal{O}\left(\left(\|e_k\|_{\infty,[a,b]} - \|e^\star\|_{\infty,[a,b]}\right)^2\right). \tag{3.5}$$

Based on our experiments, this quadratic behavior of a multi-point exchange rule generally holds for multi-interval domains X as well. Also, under the same regularity assumptions from [217] for the one point exchange strategy, Powell [168, Ch. 9.3–9.4] also shows a $(n+2)$-step quadratic rate of convergence to $p^\star$.

Remark 3.16. In terms of numerical implementations, it is perhaps fair to say that the topic has garnered more interest in the Signal Processing literature over the years than from numerical mathematicians. Not long after the publication of the article by Parks and McClellan [164], a first Fortran implementation of their ideas was presented in [148]. It *explicitly* works on a discretized version of the design problem; at each iteration, the new reference set from Step 3 is chosen from a dense grid X_d of points from $X \subseteq [-1, 1]$, still a compact subset of $[-1, 1]$. The size of X_d is taken to be large enough (the initial value advocated in [164] was $20n$), with the hope that the solution will be close to the one over X. Computing the reference set for the next iteration is done using a multi-point exchange strategy and an exhaustive evaluation of $e_k(x)$ over X_d. This early Fortran program is a starting point for current codes, like the `firpm` function from MATLAB's Signal Processing toolbox and the variations from [6], or similar routines from Scilab, SciPy and GNURadio. We will explore more of this literature in Section 3.5.

The main elements which distinguish minimax FIR filter design approaches from most other function approximation-specific codes (some of which we introduce in the next paragraph) are:

- *multi-interval domains* and *weighting functions* versus working on $X = [a, b]$;
- using the basis of Chebyshev polynomials $(T_k(x))_{0\leqslant i\leqslant n}$ over the monomial basis $(x^i)_{0\leqslant i\leqslant n}$.

Besides the minimax routines available in well-known numerical and Computer Algebra software packages like Mathematica and Maple, which have been around for many years now, we would like to signal out two other recent efforts which greatly inspired and motivated our own work. The first one is the Sollya code we mentioned in Section 3.2.2, a fast C/C++ multiple-precision implementation of the second Remez algorithm, highly suitable for polynomial approximation of functions to be used in the development of mathematical libraries [130]. It can work with arbitrary finite-dimensional spaces S_n of linearly independent continuous functions $\varphi_0, \ldots, \varphi_n$, to approximate functions over $[a, b]$. The other is, reiterating Section 3.1, Chebfun `remez`. More recently, with the extension of Chebfun to handle periodic function approximation by using trigonometric polynomials [231] (see the basis from Example 3.7(a)), the `trigremez` command [111] was introduced; it computes best approximations with real coefficients to periodic functions on a closed interval.

3.3.2 The first Remez algorithm

The first Remez algorithm (summarized in the listing for Algorithm 3) is applicable to more general problems and also consists of solving a sequence of discrete problems, but this time of increasing size. It does not assume we are working with a Chebyshev system, only that the system matrix generated by the initial reference $X^1 \subseteq X$ has full rank (Step 1). At each iteration k, the Chebyshev approximation problem on the finite set X^k is solved and a point where the error function modulus reaches its norm is added to the new reference X^{k+1} (Step 2). The process continues until we are close enough to *a* best approximation over X (Step 3). A proof of convergence of h_k to $\|f - p^\star\|_{\infty,X}$ for this procedure can be consulted in [56, Ch. 3.8]. Similarly to the second Remez algorithm, multiple exchange strategies can also be introduced in the hope of accelerating convergence [25, 129].

Algorithm 3: First Remez algorithm

Input: degree $n \in \mathbb{N}$, compact subset X of $[a, b]$, $f \in \mathcal{C}(X)$, a finite-dimensional space $S_n = \operatorname{span}_{\mathbb{R}}(\varphi_i)_{0 \leqslant i \leqslant n}$ of linearly independent functions from $\mathcal{C}(X)$, convergence parameter threshold $\varepsilon_t > 0$

Output: an element $p(\boldsymbol{\alpha}, x) = \sum_{k=0}^{n} \alpha_k \varphi_k(x) \in S_n$, where $\boldsymbol{\alpha} \in \mathbb{R}^{n+1}$, close to optimal

Step 1 (Initialization) Set $k = 1$ and choose a starting *reference* set $X^1 \subseteq X$ containing $n + 2$ points $\{x_i\}_{i=0}^{n+1}$ such that the Vandermonde-like system matrix

$$V = [v_{ij}] := [\varphi_j(x_i)], \qquad i = 0, \ldots, n+1, j = 0, \ldots, n$$

has rank $n + 1$.

Step 2 Select a coefficient vector $\boldsymbol{\alpha}^k \in \mathbb{R}^{n+1}$ for which the error function $e(\boldsymbol{\alpha}^k, x) = f(x) - p(\boldsymbol{\alpha}^k, x)$ is such that $h_k = \|e(\boldsymbol{\alpha}^k, x)\|_{\infty,X^k}$ is *minimal*.

Step 3 Pick $x^k \in X$, a point where $\|e(\boldsymbol{\alpha}^k, x)\|_{\infty,X}$ is attained. If $\dfrac{\|e(\boldsymbol{\alpha}^k, x)\|_{\infty,X} - h_k}{\|e(\boldsymbol{\alpha}^k, x)\|_{\infty,X}} \leqslant \varepsilon_t$, then we are done and **return** $p(\boldsymbol{\alpha}^k, x)$. Otherwise, let $X^{k+1} = X^k \cup \{x^k\}$, increment k and go to **Step 2**.

It is worth discussing in a bit more detail the discrete linear Chebyshev approximation problem appearing in Step 2. A very interesting historical account of it is given by Watson [223, Sec. 2.4]. We will retread some of his notes in the following paragraphs.

Let us thus assume that X is a finite set containing at least $n+2$ points. If the Haar condition is satisfied over X (*i.e.,* for any choice of $n+1$ distinct points from X, the determinant appearing in Definition 3.3 is different from zero), then, an exchange algorithm approach due to Stiefel [198] can be used to determine $p^\star \in S_n$ for which $\|f - p^\star\|_{\infty,X}$ is minimal. This *ascent* procedure is an extension of the one-point exchange version of Algorithm 2 and is characterized by the same behavior: the levelled error modulus is *strictly increasing* at successive iterations. Because there is only a finite number of $(n + 2)$-element references in X, the algorithm is guaranteed to finish in a finite number of steps. Descloux showed that the algorithm can also be made to work in the absence of the Haar condition, provided some appropriate changes [74] are made.

Interestingly, but perhaps not that surprisingly, there is a strong link between the Stiefel exchange method and the simplex algorithm applied to the dual of the linear programming formulation of the discrete Chebyshev approximation problem. This was discussed by Stiefel in [199], but the precise equivalence between a step of the dual simplex method and a step of the Stiefel exchange method was apparently only later given by Osborne and Watson [159]. An argument for preferring the Stiefel exchange algorithm is related to the fact that it seems to be more concise and efficient to implement than applying a general purpose simplex implementation. A series of articles [13, 16–18, 104] discuss different approaches (in terms of efficiency and numerical results) for implementing such exchange/simplex-based methods. There is also

a rich literature on *descent* methods which are related to the primal formulation of the discrete linear Chebyshev approximation problem. We will not go into them here, but refer the reader to [223, Sec. 2.4] for some details and further references.

At the time of writing this thesis, we are not aware of any widely available implementations of the first Remez algorithm for univariate linear Chebyshev approximation. Even the Sollya `remez` routine, when dealing with systems S_n which do not satisfy the Haar condition, uses a heuristic hybrid approach of the multi-point exchange version of Algorithm 2 and the one-point Stiefel exchange algorithm (see [58, Ch. 2.2.7] for more details). Even though it offers no guarantee of theoretical convergence, the Sollya implementation works quite well in many situations when the Haar condition is not satisfied. In particular, if the routine terminates successfully, then Corollary 3.12 can be used to validate the quality of the result.

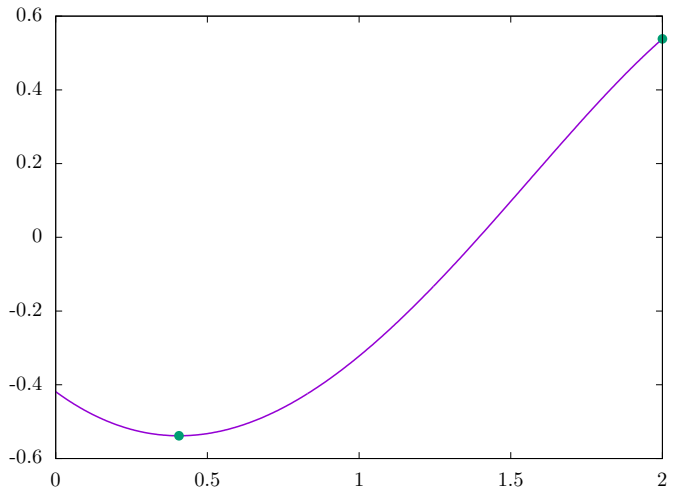

Figure 3.1: Error curve for the minimax approximation of x^2 over $[0, 2]$ using the basis (x, e^x).

Remark 3.17. A general and robust implementation of the first Remez algorithm needs an effective way of addressing *singular problems*, that is, instances where the solution on a continuum X has fewer that $n + 2$ points where the error modulus attains its norm. Note that for finite X, this situation does not occur; it is a *feature* of the fact that the corresponding system matrix has full rank [159]. The issue is explored by Osborne and Watson in [160], where they argue that some of the reference points can coalesce, slowing down convergence and giving rise to ill-conditioned simplex bases. This is most likely to happen when the singular problem solution is *unique*. They exhibit this for the uniform approximation of $f(x) = x^2$ over $[0, 2]$ using the basis (x, e^x), which has the solution

$$p^\star(x) \simeq 0.18423256x + 0.41863122e^x,$$

and only two points of maximum deviation from f exist (as visible in Figure 3.1). A possible way of dealing with such problems might be the use of extended-precision and/or exact arithmetic linear programming solvers, in the vein of [89, 90].

3.4 Our setting

In the rest of this chapter, we will mostly concentrate on describing an efficient approach for implementing the second Remez algorithm, with particular attention to multi-interval weighted polynomial approximation problems. Without any loss of generality, we take the domain X to be a subset of $[-1, 1]$. The bases we will be considering are of the form $(w(x)T_i(x))_{0 \leqslant i \leqslant n}$, where $w \in \mathcal{C}(X)$ is a positive function.

Since a large motivation for this work is applications to digital filter design, we will frequently be switching between the frequency domain formulation we introduced in Section 2.2.3 and the notation we

Table 3.1: Link between the two representations used interchangeably in the rest of the text.

$\Omega \subseteq [0, \pi]$	$X \subseteq [-1, 1]$
ω	$x = \cos(\omega)$
$\cos(k\omega)$	$T_k(x)$
$W(\omega)$	$w(x) = W(\omega)$
$W(\omega)D(\omega)$	$f(x) = w(x)D(\omega)$
$W(\omega)H(\omega)$	$p^\star(x) = w(x)H(\omega)$
$p^\star \in w\mathbb{P}_n(\mathbb{R}) = \mathrm{span}_{\mathbb{R}}(wx^i)_{0 \leqslant i \leqslant n}$	
$\delta_\Omega = \delta_X = \|W(D-H)\|_{\infty,\Omega} = \|f - p^\star\|_{\infty,X}$	

used in Sections 3.2 and 3.3. The two main reasons for this is that the frequency domain setting is much more convenient to use when discussing the examples and references from the Signal Processing literature, whereas working with the change of variable $x = \cos(\omega)$ is much better suited for presenting theoretical and implementation details related to the exchange algorithm. For convenience, the link between the two is summarized in Table 3.1.

3.4.1 The role of multi-interval domains

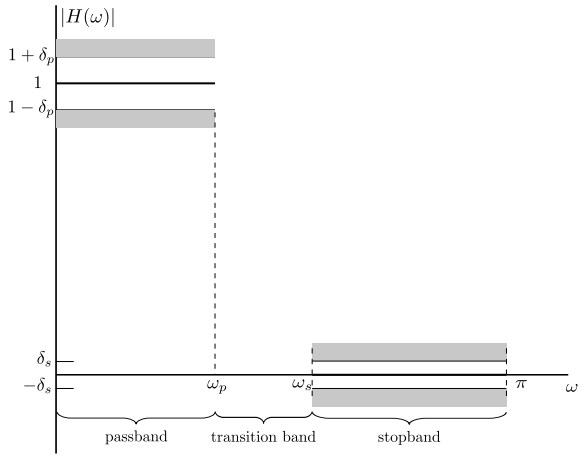

Figure 3.2: Tolerance scheme and ideal frequency response for a lowpass filter.

A typical use case for the exchange method is the design of lowpass filters, like the one in Figure 3.2, with $\Omega = [0, \omega_p] \cup [\omega_s, \pi]$,

$$D(\omega) = \begin{cases} 1, & 0 \leqslant \omega \leqslant \omega_p, \\ 0, & \omega_s \leqslant \omega \leqslant \pi, \end{cases} \quad \text{and} \quad W(\omega) = \begin{cases} \frac{\delta_s}{\delta_p}, & 0 \leqslant \omega \leqslant \omega_p, \\ 1, & \omega_s \leqslant \omega \leqslant \pi, \end{cases}$$

where δ_p and δ_s denote the maximum errors allowed over $[0, \omega_p]$ (the *passband*) and $[\omega_s, \pi]$ (the *stopband*). Between pass and stopbands, we find transition bands. They are crucial here because their absence incurs jump discontinuities in the functions to approximate (at $\omega = \omega_p = \omega_s$), giving rise to the unwanted Gibbs-Wilbraham phenomenon (see [205, Ch. 9] for a detailed description).

Some difficulties

In many situations arising in practice, the frequency response of a minimax filter inside transition regions is required to be *monotonic* or at least not present *overshoots* [158, 172]: that is, on a transition band,

$H(\omega)$ can only take values which are inside the interval determined by its minimum and maximum on the neighboring passband and stopband. Such restrictions are implicitly assured in most practical cases, even though they do not hold in general [172].

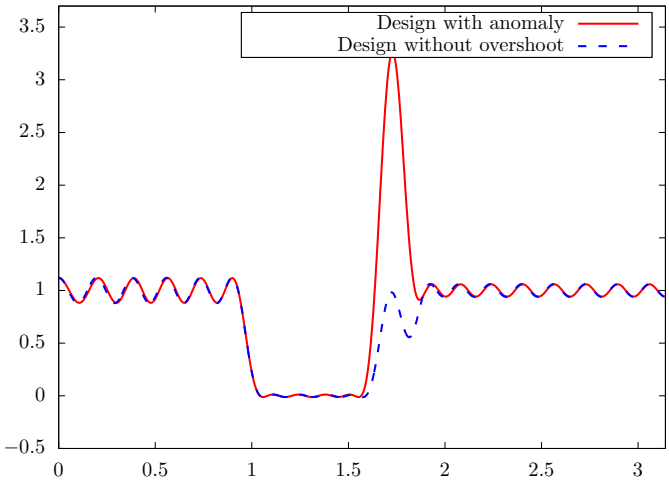

Figure 3.3: Design example showcasing a transition band anomaly and how it can be removed.

Figure 3.3 shows an example of a three band, degree $n = 38$, type I minimax FIR filter with passband $[0, 0.3\pi]$, stopband $[0.33\pi, 0.5\pi]$, passband $[0.6\pi, \pi]$ and corresponding weights 1, 10 and 2. A huge unwanted spike is clearly visible in the transition region between the stopband and the second passband. Such a scenario is possible because, in the case of multi-band filters, the minimax problem results in a so-called *underdetermined* approximation polynomial $H(\omega)$ over Ω, meaning that some of the local extrema of $H(\omega)$ over $[0, \pi]$ can occur inside transition regions (see the discussion from [191]).

Such issues have been approached in different ways over the years. The most satisfactory solution seems to be the one advocated by Shpak and Antoniou [191]. They propose extensions of Algorithm 2 for the design of *maximal-ripple* (MR) and *extra-ripple* (ER) filters. MR filters are those filters that have the maximum possible number of ripples in the magnitude response, while ER filters are those that have fewer ripples than the maximum, but more than the minimum $n + 2$ required by the equioscillation theorem. We will not go into details since such approaches are very similar to the classic multi-point exchange algorithm.

The methods of Shpak and Antoniou completely eliminate or significantly reduce the underdetermined nature of $H(\omega)$ that affects multi-band designs obtained using the exchange algorithm, thus ensuring monotonicity of H inside transition regions. This desirable behavior on the transition bands comes at the cost of not having explicit control on the value of the approximation error inside the pass and stop bands.

If we only need to contain the frequency response, we can adapt the initial specification using mild constraints on the problematic transition region [6, Sec. 3]. For instance, by adding $[0.51\pi, 0.59\pi]$ to Ω and extending D and W to take the values 0.5 and 0.25 on $[0.51\pi, 0.59\pi]$, we obtain the second minimax filter from Figure 3.3, where the transition region response is now between 0 and 1. There is only a minor loss in quality with respect to the initial design (the minimax error δ_Ω grows from 0.1172 to 0.1205). Another more involved approach of a similar nature (*i.e.,* forcing a design problem on $[0, \pi]$ instead of Ω) is described in [66]. Grenez [93] describes a version of the exchange algorithm which can also take into account lower and upper bound functions on the approximant to be computed. This constraint barrier method can limit the size of any ripples that might appear in the frequency response, which means it can be used to eliminate transition band anomalies by forcing suitable bounds. The downside is that in this setting as well, the resulting filter is under-determined. A similar approach is given in [147].

Despite the focus on filter design problems, the routines we provide are more general. They can be used for arbitrary weighted minimax polynomial approximation problems involving multi-interval domains. Such a scenario arises, for instance, when minimizing the *relative error* of approximation for a function with

zero(s).

As an example, consider the function $f(x) = \ln(x)e^{\sin(x)}$. We want to find the polynomial p of degree at most $n = 10$, which *best* approximates f over $[1 - 2^{-3}, 1 + 2^{-3}]$ in terms of relative error, implying the weight function $w(x) = 1/|f(x)|$ (the minimax result is then $p^\star = wp$). Since f has a zero at $x = 1$, in a double-precision context, we can split the domain into two parts:

$$[1 - 2^{-3}, 0.99999999999999988897769753748434595763683319091796875]$$

and

$$[1.00000000000000022204460492503130808472633361816406 25, 1 + 2^{-3}],$$

that is, remove all the values between the largest double-precision value smaller than 1 and the smallest double-precision value greater than 1.

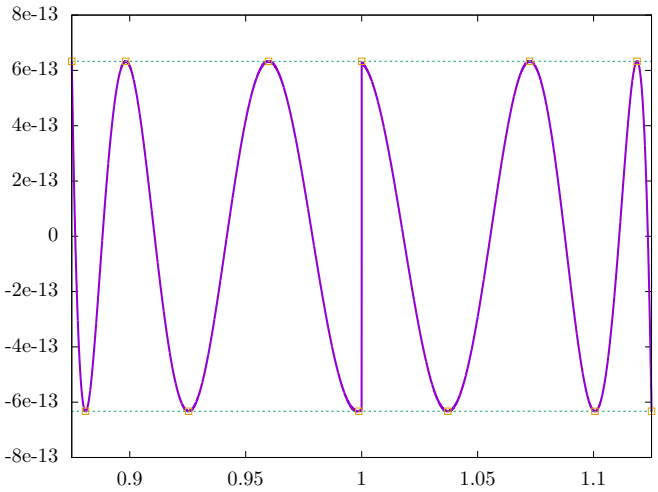

Figure 3.4: Degree $n = 10$ minimax approximation of $f(x) = \ln(x)e^{\sin(x)}$ in terms of relative error.

Figure 3.4 shows the minimax error $(f(x) - p(x))/|f(x)|$ obtained by running our implementation of the exchange algorithm on this new two interval domain, together with the $n + 2 = 12$ equioscillation points around the extremal values $\pm 6.32824 \cdot 10^{-13}$.

3.4.2 Four key examples

We will be frequently using four filter design problems to highlight our approach. They are taken and/or adapted from the literature and prove to be hard or impossible to solve using current *de facto* minimax implementations.

The first two examples come from [170].

Example 3.18. *A unit weight type I lowpass filter with* $[0, 0.4\pi]$ *passband and* $[0.5\pi, \pi]$ *stopband. Multiple degrees* n *are considered.*

Example 3.19. *A type I, uniformly weighted, bandstop filter with passbands* $[0, 0.2\pi], [0.6\pi, \pi]$ *and stopband* $[0.3\pi, 0.5\pi]$. *Several degrees* n *are used.*

The next problem comes from [235] and is related to the design of equiripple comb FIR filters.

Example 3.20. *A unit weight type I linear phase filter with passband* $[0, 0.99\pi]$, *stopband centered at* π *and degree* $n = 520$.

Our last example is derived from [1]. There, the exchange algorithm acts as an intermediary step in designing efficient wideband channelizers. The lengths of the filters they use are proportional to the number of desired channels. In order to design a 8192-subchannel system, we can use:

Example 3.21. *A degree* $n = 13 \cdot 4096 = 53248$ *unit weight type II lowpass filter with passband* $\left[0, \frac{1}{8192}\pi\right]$ *and stopband* $\left[\frac{3}{8192}\pi, \pi\right]$.

3.5 Practical problems

Over the years, certain problems with implementing multi-interval exchange methods have been discussed in the literature. We give short accounts on the most important ones, previous solutions and how we address them in the sequel.

3.5.1 Robustness issues: initialization is a key step

During the 1980s, the need for obtaining high-order filters, with $n = 200 \dots 500$ or even larger became a concern. Examples of such application scenarios (along with references) are given in [80, 166] and include the design of transmultiplexers and filter banks for high resolution spectral analysis.

The default strategy: uniform initialization

A good choice for the starting reference vector $\mathbf{x}^1 = (x_i^1)_{0 \leqslant i \leqslant n+1}$ becomes critical for practical convergence of the exchange algorithm for large degrees, even though theoretically, their placement inside X is not important. There are two main reasons for picking a *good* starting reference:

- reasonable execution time: a small number of iterations required for convergence;
- some starting references are much better conditioned numerically than others.

Historically, initialization for FIR filter design is carried out in the Ω setting. According to [11, p. 679], the basic idea is to take the references in a *uniform* way inside Ω. If this domain is formed from m distinct non-degenerate bands of widths $W_1, \dots, W_m$ and we allocate an extrema at the left and right edge of each band, then the total bandwidth $\sum_{i=1}^{m} W_i$ of Ω should be divided into $n + 2 - m$ intervals. This gives us an average distance between adjacent reference points of

$$d_0 = \frac{1}{n + 2 - m} \sum_{i=1}^{I} W_i.$$

Computing the number of intervals $(n_i)_{1 \leqslant i \leqslant m}$ located in each band can now be accomplished if we round the quantities W_i/d_0 to integers. This also allows us to readjust the distance between consecutive extrema inside each band, which we denote by $(d_i)_{1 \leqslant i \leqslant I}$. We get the following formulas

$$\begin{aligned} n_i &= \left\lfloor \frac{W_i}{d_0} + 1 \right\rfloor, \qquad i = 1, \dots, m-1, \\ n_m &= n + 1 - \sum_{i=1}^{m-1} (n_i + 1), \\ d_i &= \frac{W_i}{n_i}, \qquad i = 1, \dots, n, \end{aligned}$$

which allow us to compute the exact position of each initial ω_i^1. Recall that $x_i^1 = \cos(\omega_i^1), i = 0, \dots n+1$. The approach when Ω is replaced by a discretization Ω_d of equidistant points from Ω is even more simple. If the points of Ω_d are ordered, the reference values are taken uniformly.

An empirical justification comes from a parallel with how one initializes the Remez algorithm on $X = [-1, 1]$. There, good choices are usually Chebyshev nodes of the second kind (see Section 2.1.1), which, in the $\Omega = [0, \pi]$ setting, map to a uniform grid.

Least-squares approximations

The problem with the uniform initialization approach of the previous paragraphs is that, for a large number of high-degree problems, it does not behave very well. An alternative strategy is presented in [170]. It seems to greatly reduce the number of iterations required for convergence compared to the classical initialization scheme. The idea is to first compute a certain optimal L_2 approximation with the same design constraints and take the local extrema of its approximation error as the initial reference set.

Problem 3.22 (Least-squares FIR filter design)**.** *Let $D(\omega)$ and $W(\omega)$ be real valued functions, continuous on Ω and assume W to be positive over Ω and zero everywhere else. We want to determine the filter $H(\omega) = \sum_{k=0}^{n} h_n \cos(\omega k)$, with corresponding weighted approximation error $E(\omega) = W(\omega)\,(D(\omega) - H(\omega))$, which minimizes the following least-squares measure*

$$\Delta(\boldsymbol{h}) = \|E(\omega)\|_{2,W} = \int_0^\pi W(\omega)\,(D(\omega) - H(\omega))^2\,\mathrm{d}\omega,$$

where we denoted with $\boldsymbol{h}$ the coefficient vector $[h_0 \cdots h_n]$ of $H(\omega)$.

We notice that this function is differentiable in each h_k coefficient, thus at any extrema of Δ the conditions $\partial\Delta/\partial h_k = 0, k = 0, \ldots, n$ must hold. Performing the differentiations, we obtain a linear system in $\boldsymbol{h}$, defined by the equations

$$\int_0^\pi W(\omega)\,(D(\omega) - H(\omega))\cos(\omega k)\mathrm{d}\omega = 0, \qquad k = 0, \ldots, n.$$

The corresponding system matrix is invertible (see [56, Ch. 4.1] for a proof), and as such, the problem has a unique solution. Also, solving it can be done efficiently in $\mathcal{O}(n^2)$ operations or less, by taking advantage of the special Toeplitz-plus-Hankel structure of the system (see [53, 149] for more information), compared to the $\mathcal{O}(n^3)$ operation count we would get from using Gaussian elimination.

One of the reasons to use the result of Problem 3.22 for choosing the starting reference values $(\omega_i^1)_{0 \leqslant i \leqslant n+1}$ is due to Theorem 2 from [170], which states that the least-squares solution exhibits at least $n + 2$ local extrema inside Ω. Equally important is the fact that L_2 designs such as these tend to behave well with respect to the minimax error measure (see [44, 45] for corresponding FIR filter design methodologies). Examples 3.18 and 3.19 with degrees ranging from 25 to 100 are used to highlight this approach. The results are compared to those obtained using the MATLAB `firpm` routine. In particular, designs where the `firpm` code requires a lot of iterations or does not even achieve convergence are handled without problem by using this initialization approach. Based on our own tests, this remains true even for more recent versions (R2014b) of MATLAB.

Other choices for the initial reference

Based on experimental observations for filter design (see [80] for example), good initialization choices seem to be those where the final references in Ω are taken to be more densely packed near the interior edges of the pass and stop bands. Articles like [5, 166, 191] try to compute some empiric distributions that fit this behavior. Using Carathéodory-Fejér (CF) near-best approximations [163, Sec. 3.6] is also an option[2].

Potential shortcomings of the solutions we mentioned here is the fact that some are not general enough (only uniform weighted filters seem more appropriate for CF approximation) or that they prove to be too expensive when compared to uniform initialization. For the L_2 approach, its effectiveness for very large degrees rests on the numerical (in)stability of the Toeplitz-plus-Hankel solver used for dealing with Problem 3.22.

Our approach: In Section 3.6 we introduce two new heuristics for initializing the exchange algorithm. They are quite general and effective in practice, often leading to significant speedups when compared to existing implementations. Moreover, Section 3.7.2 introduces useful analysis tools for debugging numerical problems, which are, in particular, common for high-degree designs.

[2] http://www.chebfun.org/examples/approx/FiltersCF.html

3.5.2 Scalability issues: speeding up the extrema search

The most computationally intensive part of Algorithm 2 is traditionally considered to be the search for potential extrema. For filter design, articles that concentrate on this aspect are [9, 10, 27]. The common denominator of all three approaches is that they use information pertaining to the first and/or second derivative of the error function $e_k(x)$.

In [9], the strategy is to start at the elements of the current reference vector $(x_i^k)_{0\leqslant i\leqslant n+1}$ and evaluate the error only at those grid points of X_d that tend to approach a potential extrema. The search direction for a given x_i^k is determined based on the sign of the first derivative of e_k at that particular point. As the algorithm converges and the values of $(x_i^k)_{0\leqslant i\leqslant n+1}$ get closer to the actual extrema of the error function, this allows for a gradual reduction of the number of evaluations of e_k and its derivatives. The author specifies that this leads on average to a reduction of about 50% in the number of error related evaluations compared to the FORTRAN implementation from [148]. Examples for the design of two and three band filters are provided.

A similar strategy is presented inside [27]. The extrema search consists of determining an analytic formula for $e_k'(x)$ and locating its zeros, which will correspond to extrema of $e_k(x)$. It uses Brent's method (see [37, Ch. 4]) for computing the zeros of a nonlinear function. The *average* results provided seem to suggest a decrease in the computation time and in the number of required iterations.

The key remark of [10] is that, during the initial iterations of exchange algorithm, using one of the aforementioned methods might still require a considerable number of computations. Without concentrating here on the details, this is due to the fact that the actual extrema values to be determined are assumed not to be close to the starting points x_i^1. In such cases, the use of piecewise cubic or quadratic polynomial interpolation for approximating the error is proposed.

Again, it is argued that this solution works well for the design of FIR filters of up to three bands, with on average, 50% savings in computation time compared to just using the selective search from [9]. Still, for multi-band designs, such a search strategy can fail to find all potential extrema, especially for large ($n > 200$) filter degrees. One possible reason is that the interpolation intervals might contain more extrema than the cubic or quadratic polynomial can accurately capture.

Our approach: The author of [9, 10] also admits that, although being fast, the approaches described in those papers come at the cost of slightly more convergence failures to the minimax result, than when using the code from [148]. We will come back to the idea of using small degree interpolation polynomials for the error in Section 3.8 and describe a search strategy based on Chebyshev root-finding. Compared to [163], our domain subdivision routine is not recursive, making it very easy to introduce parallelism.

3.6 Initialization: two new heuristic approaches

We already hinted at the importance of starting an exchange method (of both the first and second Remez algorithm type) with an *appropriate* reference vector $\mathbf{x}^1$. Over the course of this section, we are going to consider two different, but closely connected, ways of looking at this problem in the (weighted) polynomial setting $w\mathbb{R}[x]$. Both point to extremely important and classic issues in Approximation Theory.

The first one is *interpolation*. Consider Eq. (3.2) and treat h_k as a constant. Then p_k is a weighted polynomial of degree at most n which takes the values $f(x_i^k) + (-1)^{i+1}h_k$ at the elements of the reference vector $\mathbf{x}^k$. Since it is generally expected for $|h_k|$ to be a small error value, we can see this as a perturbed interpolation of f at $\mathbf{x}^k$. What sets $\mathbf{x}^k$ are good for interpolation?

The second idea is of an *asymptotic* nature. Do the equialternating sets to which the reference vectors $\mathbf{x}^k$ converge follow a certain distribution over X as n tends to infinity? If the answer is yes, can this information be used in a numerical setting, as a robust way of starting the exchange algorithm?

3.6.1 Lagrange interpolation problems

The exposition and ideas of the next pages are based on [30–32, 47, 73, 195, 196]. We place ourselves in the general (multivariate) setting where $X \subset \mathbb{R}^d$ (or $\mathbb{C}^d$) is a compact set, $S_n = \operatorname{span}_{\mathbb{R}}(\varphi_i)_{0\leqslant i\leqslant n}$ is a

finite-dimensional space of continuous functions over X and $\{x_0, \dots, x_n\} \subset X$ is a finite set (we also use the vector $\mathbf{x} = (x_i)_{0 \leqslant i \leqslant n}$). Then, we can construct the Vandermonde-like matrix

$$\mathbf{V}(x_0, \dots, x_n) = [v_{ij}] := [\varphi_j(x_i)].$$

If $\det \mathbf{V}(x_0, \dots, x_n) \neq 0$, then the set $\{x_0, \dots, x_n\}$ is called *unisolvent* for interpolation in S_n, which allows us to introduce the notion of *Lagrange interpolation operator*. It is defined as

$$\mathcal{L}_{S_n,\mathbf{x}} f(x) = \sum_{i=0}^{n} f(x_i)\ell_i(x), \tag{3.6}$$

where

$$\ell_i(x) = \frac{\det \mathbf{V}(x_0, \dots, x_{i-1}, x, x_{i+1}, \dots, x_n)}{\det \mathbf{V}(x_0, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_n)}, \qquad i = 0, \dots, n \tag{3.7}$$

is called a *Lagrange* or *cardinal basis* (*i.e.*, $\ell_i(x_k) = \delta_{ik}$).

$\mathcal{L}_{S_n,\mathbf{x}} f$ interpolates any function f at $\{x_0, \dots, x_n\}$. The norm of the interpolation operator $\mathcal{L}_{S_n,\mathbf{x}} : \mathcal{C}(X) \to S_n$ (also called *Lebesgue constant*) satisfies

$$\Lambda_{S_n,X}(\mathbf{x}) := \|\mathcal{L}_{S_n,\mathbf{x}}\| = \max_{x \in X} \sum_{i=0}^{n} |\ell_i(x)|\,. \tag{3.8}$$

The Lebesgue constant provides us with a *quantitative* way of measuring the quality of the interpolation operator. Indeed, if $\mathcal{L}_{S_n,\mathbf{x}}$ is a *linear projection operator*, meaning

$$\begin{aligned} \mathcal{L}_{S_n,\mathbf{x}}[\mathcal{L}_{S_n,\mathbf{x}} f] &= \mathcal{L}_{S_n,\mathbf{x}} f, \qquad \forall f \in \mathcal{C}(X), \\ \mathcal{L}_{S_n,\mathbf{x}}(\lambda f) &= \lambda \mathcal{L}_{S_n,\mathbf{x}} f, \qquad \forall f \in \mathcal{C}(X), \lambda \in \mathbb{R}, \\ \mathcal{L}_{S_n,\mathbf{x}}(f + g) &= \mathcal{L}_{S_n,\mathbf{x}} f + \mathcal{L}_{S_n,\mathbf{x}} g, \qquad \forall f, g \in \mathcal{C}(X), \end{aligned}$$

then the following holds:

Theorem 3.23. *For all $f \in \mathcal{C}(X)$, we have*

$$\|f - \mathcal{L}_{S_n,\mathbf{x}} f\|_{\infty,X} \leqslant (1 + \Lambda_{S_n,X}(\mathbf{x}))\mathrm{dist}_X(f, S_n), \tag{3.9}$$

where the distance is taken with respect to the uniform norm over X.

Proof. The operator norm of $\mathcal{L}_{S_n}$ is the smallest real number such that $\|\mathcal{L}_{S_n,\mathbf{x}} f\|_{\infty,X} \leqslant \Lambda_{S_n,X}(\mathbf{x})\|f\|_{\infty,X}$. Since S_n is a finite-dimensional linear subspace of $\mathcal{C}(X)$, there exists a $p^\star \in S_n$ such that $\|f - p^\star\|_{\infty,X} = \mathrm{dist}_X(f, S_n)$ (see [168, Thm. 1.2], for instance). We have $\mathcal{L}_{S_n,\mathbf{x}}(f - p^\star) = \mathcal{L}_{S_n,\mathbf{x}} f - p^\star$. From the triangle inequality it then follows that

$$\|f - \mathcal{L}_{S_n,\mathbf{x}} f\|_{\infty,X} - \|f - p^\star\|_{\infty,X} \leqslant \|\mathcal{L}_{S_n,\mathbf{x}} f - p^\star\|_{\infty,X} = \|\mathcal{L}_{S_n,\mathbf{x}}(f - p^\star)\|_{\infty,X} \leqslant \Lambda_{S_n,X}(\mathbf{x})\|f - p^\star\|_{\infty,X}.$$

□

Remark 3.24. For an arbitrary family of nodes $\mathbf{x} \in X^{n+1}$, its associated Lebesgue constant is independent of which basis S_n is represented in. Indeed, all the determinants appearing in the previous computations are scaled by a factor given by the change of basis matrix, independent of the point set under consideration.

Because of Theorem 3.23, a question worth asking at this point is, how small can the Lebesgue constant of a family of points be? In case of polynomial approximation, the general conclusion is nicely summed up by the Hungarian mathematician Pál Turán, who says [208, p. 25]: *"After ... the approximation theorem of Karl Weierstrass, it was hoped that there exists a (non-equidistant) system of nodes for which the Lagrange interpolation polynomials converge uniformly for every continuous function in $[-1, 1]$. The mathematical world was awakened from this dream in 1914 by Georg Faber who showed that there is no such system."*

In practice, the situation is not as dire as Faber's result (see for instance [56, Ch. 6.5] for a precise statement and proof) might lead us to believe. If f is $\nu \geqslant 0$ times differentiable, then it is usually the case

that the degree n Lagrange interpolant at Chebyshev nodes of the second kind (a Chebyshev interpolant, as defined in Section 2.1.2) achieves an accuracy of $\mathcal{O}(n^{-\nu})$. When f is *analytic* on $[-1,1]$ (for every $x \in [-1,1]$ f has a Taylor series expansion at that point and which converges to f in a neighborhood of x), the work of Bernstein [21] helps show that the rate of convergence for Chebyshev interpolants is even better, becoming $\mathcal{O}(C^{-n})$, for some constant $C > 1$. For the precise details of these results (which also hold for the truncated Chebyshev series of f) and further references, we recommend [205, Ch. 7–8].

The usefulness of Chebyshev nodes inside $X = [-1,1]$ is further supported by the fact that their associated Lebesgue constants grow at an extremely low rate as $n \to \infty$:

Theorem 3.25. *The Lebesgue constants $\Lambda_{\mathbb{R}_n[x],[-1,1]}$ associated to the mapping defined by a degree $n \geqslant 0$ polynomial interpolation using any set of $n+1$ distinct points from $[-1,1]$ satisfy*

$$\Lambda_{\mathbb{R}_n[x],[-1,1]} \geqslant \frac{2}{\pi}\left(\log(n+1) + \gamma + \log\frac{4}{\pi}\right),$$

where $\gamma \approx 0.577\ldots$ is Euler's constant. Also,

- *for Chebyshev nodes (of the first and second kind), we have an upper bound*

$$\Lambda_{\mathbb{R}_n[x],[-1,1]} \leqslant \frac{2}{\pi}\log(n+1) + 1 \qquad \text{and} \qquad \Lambda_{\mathbb{R}_n[x],[-1,1]} \sim \frac{2}{\pi}\log(n) \text{ as } n \to \infty$$

- *for equispaced points,*

$$\Lambda_{\mathbb{R}_n[x],[-1,1]} > \frac{2^{n-2}}{n^2} \qquad \text{and} \qquad \Lambda_{\mathbb{R}_n[x],[-1,1]} \sim \frac{2^{n+1}}{en\log(n)} \text{ as } n \to \infty.$$

Proof. See for instance [205, Ch. 15] and the references therein. □

Even in the weaker context where f is only Lipschitz continuous over $[-1,1]$, meaning that there exists a constant $C > 0$ such that $|f(x) - f(y)| < C\,|x - y|$ for all $x, y \in [-1,1]$, one of Jackson's theorems [109] states that the minimax approximation $p^\star \in \mathbb{R}_n[x]$ to f satisfies

$$\|f - p^\star\|_{\infty,[-1,1]} \leqslant \frac{\pi C}{n+2}.$$

Based on Theorems 3.23 and 3.25, it means that even in this setting, Chebyshev interpolants will have at least an $\mathcal{O}\left(\frac{\log n}{n}\right)$ (uniform) rate of convergence.

Thus, for practical purposes, Chebyshev nodes are essentially optimal when performing polynomial interpolation in $[-1,1]$, whereas equidistant nodes are usually a bad idea. But what about more intricate domains X?

More generally, a natural way to bound the value of the Lebesgue constant is by taking a set of points which minimize the absolute value of the denominator of (3.7) over X^{n+1} (so-called *Fekete points*), leading to $\|\ell_i\|_{\infty,X} \leqslant 1$ for every i and consequently $\Lambda_{S_n,X}(\mathbf{x}) \leqslant n+1$ in (3.8). The difficulty with computing Fekete points is that they require the solution of a (complicated) nonlinear optimization problem with $d(n+1)$ real (or complex) unknowns. In our setting, maybe the only truly satisfactory case where their analytical properties are well understood is the interval (*i.e.,* it is known that for a degree n interpolation problem on $[-1,1]$, Fekete points correspond to the extrema of the $n+1$ degree Legendre polynomial [200] and grow like $\mathcal{O}(\log n)$).

A reasonable idea worth considering is to simplify the problem by taking a *suitable discretization* Y of X,

$$Y = \{y_0, \ldots, y_m\} \subseteq X, \qquad m > n,$$

construct the rectangular $(m+1) \times (n+1)$ Vandermonde-like matrix

$$\mathbf{V}(y_0, \ldots, y_m) = [v_{ij}] := [\varphi_j(y_i)]$$

and extract from it a maximum volume $(n+1) \times (n+1)$ submatrix. Even in this discrete Fekete points setting, the problem we are trying to solve is NP-hard [49] in general. Despite this, we briefly present two heuristic greedy algorithms found in the literature, which tend to behave quite well in practice [30].

Approximate Fekete points

Suppose $\mathbf{A} = \mathbf{V}(y_0, \ldots, y_m)^t \in \mathbb{R}^{(n+1)\times(m+1)}$ has full rank (true, if Y contains a subset of $n+1$ elements satisfying the Haar condition for S_n). The idea behind the first alternative (implemented by Sommariva and Vianello [195]) is to select the $n+1$ columns $\mathbf{a}_0, \ldots, \mathbf{a}_n$ from $\mathbf{A}$ as follows: (1) $\mathbf{a}_0$ is the column of maximum ℓ_2 norm and (2) having chosen $\mathbf{a}_0, \ldots, \mathbf{a}_k$, the $(k+1)$ column is picked such that the volume generated by $\mathbf{a}_0, \ldots, \mathbf{a}_k, \mathbf{a}_{k+1}$ is as large as possible.

The nodes from Y associated with the selected columns are called *approximate* Fekete points (AFPs). They can be computed quite efficiently using the well-established QR factorization with column pivoting of Businger and Golub [46] as expressed in the listing for Algorithm 4. We refer the reader to [32, Sec. 2] for the precise link between the two formulations.

Algorithm 4: AFP

Input: $\mathbf{V}(y_0, \ldots, y_m)$ full-rank matrix
Output: a set $X^\star \subset Y$ containing $n+1$ distinct elements that generates a large volume submatrix of $\mathbf{A} = \mathbf{V}(y_0, \ldots, y_m)^t$

(a) $\mathbf{V}_0 \leftarrow \mathbf{V}(y_0, \ldots, y_m), \mathbf{P}_0 \leftarrow \mathbf{I}_{n+1}$ // initialization
// successive orthogonalization
for $k \leftarrow 0$ ***to*** $s-1$ **do**
  $\mathbf{V}_k \leftarrow \mathbf{Q}_k\mathbf{R}_k, \mathbf{U}_k \leftarrow \mathbf{R}_k^{-1}$
  $\mathbf{V}_{k+1} \leftarrow \mathbf{V}_k\mathbf{U}_k, \mathbf{P}_{k+1} \leftarrow \mathbf{P}_k\mathbf{U}_k$
// greedy algorithm for a maximum volume submatrix
(b) $\mathbf{A} \leftarrow \mathbf{V}_s^t; \mathbf{b} \leftarrow (1, \ldots, 1)^t$ // any nonzero column vector $\mathbf{b}$ should work
$\mathbf{w} \leftarrow \mathbf{A}^{-1}\mathbf{b}$ // solve using column pivoting QR
(c) $ind \leftarrow \mathrm{find}(\mathbf{w} \neq 0), X^\star \leftarrow Y(ind)$ // extraction of the AFPs
return $X^\star$

Remark 3.26. The greedy process for choosing the appropriate columns $\mathbf{a}_k$ is in fact implemented by part (b) of Algorithm 4. Part (a) serves the purpose of managing rank-deficiency and ill-conditioning that might arise when working with nonorthogonal bases of S_n, like the standard monomial basis for polynomial approximation. It performs a change of basis from $(\varphi_0, \ldots, \varphi_n)$ to an (almost) discrete orthonormal basis

$$(\varphi_0, \ldots, \varphi_n) = (\varphi_0, \ldots, \varphi_n)\mathbf{P}_s,$$

with respect to the inner product

$$\langle f, g\rangle = \sum_{i=0}^{m} f(y_i)\overline{g(y_i)}$$

using a sequence of QR decompositions [195, Sec. 3]. About $s = 1$ or 2 iterations are generally sufficient in practice, provided the initial matrix $\mathbf{V}(y_0, \ldots, y_m)$ is not too ill-conditioned with respect to the numerical precision used for the computations. At the end of execution, part (c) outputs $X^\star$ as the set of elements from Y whose corresponding terms inside $\mathbf{w}$ are different from zero, that is, if $y_i \in Y$ and $w_i \neq 0$, then $y_i \in X^\star$. Also, the computed AFPs will depend on the choice of basis.

Leja points

Let us consider $(\varphi_0, \ldots, \varphi_n)$ to be an *ordered* basis of S_n. We define a *sequence* of points $\xi_0, \ldots, \xi_n \in X$ to be *Leja* points for X, up to dimension $n+1 = \dim S_n$, if they are constructed as follows.

In the beginning, ξ_0 is taken to be a max argument of $|\varphi_0|$ on X (each such point gives rise to a *different* Leja sequence). At a certain moment when the points $\xi_0, \ldots, \xi_{k-1}$ have been chosen, $\xi_k \in X$ is taken as a max point of the function

$$x \to |\det \mathbf{V}_k(\xi_0, \ldots, \xi_{k-1}, x)|,$$

where

$$\mathbf{V}_k(x_0, \ldots, x_k) = [v_{ij}] := [\varphi_j(x_i)], \qquad i = 0, \ldots, k, j = 0, \ldots, k.$$

A first variant of this procedure was introduced by Leja [131] (he called $\xi_0, \ldots, \xi_n$ *extremal sequence*), in the context of polynomial approximation over a compact $X \subset \mathbb{C}$. The greedy aspect of the approach to obtain a set of points that generate a large volume matrix is easily noticeable. By definition, at each iteration, the running sequence $\xi_0, \ldots, \xi_k$, are Leja points for the basis $(\varphi_0, \ldots, \varphi_k)$.

Algorithm 5: DLP

Input: $\mathbf{V}(y_0, \ldots, y_m)$ full-rank matrix
Output: $X^\star \subset Y$ with $n+1$ distinct elements, generating a large volume submatrix of $\mathbf{V}(y_0, \ldots, y_m)$

1 $\mathbf{A} \leftarrow \mathbf{V}(y_0, \ldots, y_m)$
2 $[\mathbf{L}, \mathbf{U}, \boldsymbol{\sigma}] \leftarrow \mathrm{LU}(\mathbf{A}, \text{'vector'})$ `// perform a partial (row) pivoting` LU `factorization of` A
`// σ is the corresponding permutation vector of the rows of` A
3 $ind \leftarrow (\sigma_0, \ldots, \sigma_n), X^\star \leftarrow Y(ind)$
4 **return** $X^\star$

When X is infinite, determining a Leja sequence using the ideas of the preceding paragraphs can be impractical. That is why, just like for AFPs, it is generally preferable to work on a finite subset $Y \subset X$. Such discrete Leja points (DLPs) can also be computed using standard tools from numerical linear algebra, namely the LU decomposition with partial pivoting, as expressed in Algorithm 5. An explanation as to why this is possible can be consulted in [30, Sec. 6.1–6.2].

There is a large literature on Leja points, both on its theoretical and computational aspects (see for instance [30, Sec. 6] and the references therein). We remark, in particular, the use of the ordering they give to improve the conditioning in evaluating the Newton form of polynomial interpolation [175].

3.6.2 Polynomial meshes and an AFP-based approach

We want to use either Algorithm 4 or 5 to construct a suitable starting reference for the second Remez algorithm. To do so, we still need a *constructive* way of discretizing $X \subset \mathbb{R}^d$. Consider the polynomial space $\mathbb{R}_n[x_1, \ldots, x_d]$.

In a 2008 paper, Calvi and Levenberg [47] propose attacking the problem of finding particular families of points which prove to be efficient for doing Lagrange interpolation by means of discretization processes used in Numerical Analysis. Their approach is to focus beforehand on the following L_2 problem:

Problem 3.27 (Least squares approximation)**.** *Given the values $f(y)$, where $y \in Y (Y \subset X)$, we want to find a polynomial p of given degree that minimizes $\sum_{y \in Y} |f(y) - p(y)|^2$.*

Theorems 1 and 2 from [47] give a way to relate the least squares solution over Y with the uniform approximation error of p over X ([47, Thm. 2] essentially gives a result for p which is similar to Theorem 3.23 for Lagrange interpolation). They show that the geometry of X plays an important role in picking a set Y of reasonable size for which p is a good approximation of f over X.

If X is a *polynomially determining* compact set (*i.e.,* polynomials vanishing on X are identically zero), a *weakly admissible mesh* (WAM) is defined to be a sequence of discrete subsets $Y_n \subset X$ such that:

$$\|p\|_{\infty,X} \leqslant C(Y_n) \, \|p\|_{\infty,Y_n}, \qquad \forall p \in \mathbb{R}_n[x_1, \ldots, x_d], \tag{3.10}$$

where both $\mathrm{card}(Y_n) \geqslant \dim(\mathbb{R}_n[x_1, \ldots, x_d]) = \binom{n+d}{d}$ and $C(Y_n)$ grow at most *polynomially* with n. If $C(Y_n)$ is bounded, then we speak of an *admissible mesh* (AM).

In terms of their properties, we mention the following (adapted from [47]):

P1 $C(Y_n)$ is invariant under affine mapping;

P2 any sequence of unisolvent interpolation sets whose Lebesgue constant grows at most polynomially with n is a WAM, with $C(Y_n)$ taking the value of the Lebesgue constant itself;

P3 any sequence of supersets of a WAM whose cardinalities grow polynomially with n is a WAM with the same constant $C(Y_n)$;

P4 a finite union of WAMs is a WAM for the corresponding union of compacts, $C(Y_n)$ being the maximum of the corresponding constants;

P5 any X satisfying a Markov polynomial inequality like $\|\nabla p\|_{\infty,X} \leqslant M n^r \|p\|_{\infty,X}$ has an AM with $O(n^{rd})$ points;

P6 the least squares solution p_n of Problem 3.27 for $f \in \mathcal{C}(X)$ over Y_n satisfies

$$\|f - p_n\|_{\infty,X} \leqslant \left(1 + C(Y_n)\left(1 + \sqrt{\operatorname{card}(Y_n)}\right)\right) \operatorname{dist}_X(f, \mathbb{R}_n[x_1, \ldots, x_d])$$

P7 Fekete points: the Lebesgue constant of Fekete points extracted from a WAM can be bounded like $\Lambda_{\mathbb{R}_n[x_1,\ldots,x_d],X} \leqslant \dim(\mathbb{R}_n[x_1, \ldots, x_d])C(Y_n)$.

Property **P5** is a consequence of [47, Thm. 5]. It gives a constructive way of finding admissible meshes when Markov-like inequalities are available. For example, a classic instance is the following

$$\|p'\|_{\infty,[-1,1]} \leqslant (\deg p)^2 \|p\|_{\infty,[-1,1]}, \qquad \forall p \in \mathbb{R}[x]. \tag{3.11}$$

In such cases, the conditions from the definition of an admissible mesh are guaranteed by virtue of metric properties. If Y_n is such a set that for each $x \in X$ there exists $y \in Y_n$ with $\|x - y\|_\infty \leqslant \varepsilon_n$ where $\varepsilon_n = \mathcal{O}(1/n^r)$ as $n \to \infty$, then the sequence Y_n forms an AM (see proof of [47, Thm. 5]).

Property **P7** is a consequence of (3.8) and (3.10). It tells us that taking as interpolation nodes Fekete points of WAMs, we generally get good quality Lagrange interpolants.

Even though the main interest in WAMs seems to be multivariate ($d > 1$) polynomial approximation, [196] shows their usefulness in the weighted univariate setting as well, which prompted us to consider them here. There are two families of meshes which are computationally easy to construct:

Example 3.28. ***Admissible meshes on*** $X \subset \mathbb{R}$ ***for*** $S_n = \mathbb{R}_n[x]$

*From (3.11), we have $r = 2$ in **P5**. Based on the previous paragraphs, if we take Y_n composed of equispaced points inside $[-1, 1]$, such that the distance between two consecutive ones is smaller than $2/n^2$ (meaning $\operatorname{card}(Y_n) = \mathcal{O}(n^2) > \dim \mathbb{R}_n[x] = n + 1$), then (Y_n) becomes an AM for $[-1, 1]$.*

*When $X = [a_1, b_1] \cup \ldots \cup [a_k, b_k]$, by the previous construction, **P1** and **P4**, we can obtain an AM for X containing $\mathcal{O}(kn^2)$ points. If $\mathbf{y}_n$ is a vector of Fekete points for such a mesh, **P7** tells us that*

$$\Lambda_{\mathbb{R}_n[x],X}(\mathbf{y}_n) \leqslant \mathcal{O}(n).$$

Example 3.29. ***Weakly admissible meshes on*** $X \subset \mathbb{R}$ ***for*** $S_n = \mathbb{R}_n[x]$

The quadratic number of points for AMs obtained from Markov-like inequalities may be unappealing when dealing with high degree problems. In that case, it can be more interesting to consider WAMs which have a much lower cardinality.

*From Theorem 3.25 and **P2**, if (Y_n) is an* appropriate *sequence of Chebyshev nodes of increasing degree, it will be a WAM on $[-1, 1]$ with $C(Y_n) = \mathcal{O}(\log n)$. Analogous to the previous example, by using **P1** and **P4**, we can construct WAMs for finite unions of closed intervals in $\mathbb{R}$. If X is a union of k such intervals, then Y_n will have size $\mathcal{O}(kn)$. The Lebesgue constant of any of its $\mathbf{y}_n$ Fekete point vectors will satisfy*

$$\Lambda_{\mathbb{R}_n[x],X}(\mathbf{y}_n) \leqslant \mathcal{O}(n \log n).$$

The appeal of using Algorithms 4 and 5 on any weakly admissible mesh $Y_n \subset X$ rests on the fact that the resulting AFPs and DLPs will have the same asymptotic distribution as the true Fekete points over X (in the sense that their corresponding discrete probability measures converge weak-$\star$ to the (pluri)potential equilibrium measure of K). These results are made explicit in [31, Thm. 1] and [30, Thm. 6.2].

It seems that approximate Fekete points are usually of slightly better quality (in terms of Lebesgue constant values and interpolation errors) than discrete Leja points, which, on the other hand, require a bit less time to compute. This is exemplified in [30,31] for several two dimensional problems. Because of this, we have preferred AFP-based starting references to DLP-based ones in the sequel.

Incorporating weight functions into the problem

We can extend the previous paragraphs to the $w\mathbb{R}_n[x_1,\ldots,x_d]$ setting as well. The weighted interpolation operator is strongly linked to the non-weighted one by uniqueness:

$$\mathcal{L}_{w\mathbb{R}_n[x_1,\ldots,x_d],\mathbf{x}} f(x) = w(x)\mathcal{L}_{\mathbb{R}_n[x_1,\ldots,x_d],\mathbf{x}} g(x), f = wg$$

and consequently

$$\left\|f - \mathcal{L}_{w\mathbb{R}_n[x_1,\ldots,x_d],\mathbf{x}} f\right\|_{\infty,X} = \left\|w(g - \mathcal{L}_{\mathbb{R}_n[x_1,\ldots,x_d],\mathbf{x}} g)\right\|_{\infty,X}.$$

The principal idea to notice is that, *in many cases*, **WAMs for non-weighted polynomial interpolation give rise to very good interpolation nodes in the weighted setting**, assuming $w \neq 0$ over X. Indeed, if Y_n is a weakly admissible mesh in the non-weighted setting (*i.e.*, $\|p\|_{\infty,X} \leqslant C(Y_n)\|p\|_{\infty,Y_n}, \forall p \in \mathbb{R}_n[x_1,\ldots,x_d]$), it follows that

$$\|wp\|_{\infty,X} = |w(x^*)p(x^*)| \leqslant |w(x^*)|\,\|p\|_{\infty,X} \leqslant |w(x^*)|\,C(Y_n)\,\|p\|_{\infty,Y_n} = |w(x^*)|\,C(Y_n)\,|p(\mu)|,$$

for certain $x^* \in X, \mu \in Y_n$. Thus, by the positivity of w over X, we get

$$\begin{aligned}\|wp\|_{\infty,X} &\leqslant \frac{|w(x^*)|}{|w(\mu)|}C(Y_n)\,|w(\mu)p(\mu)| \\ &\leqslant \frac{\max_{x\in X}|w(x)|}{\min_{x\in Y_n}|w(x)|}C(Y_n)\,\|wp\|_{\infty,Y_n} \\ &= \|w\|_{\infty,X}\,\|1/w\|_{\infty,Y_n}\,C(Y_n)\,\|wp\|_{\infty,Y_n}.\end{aligned}$$

If we take $C_w(Y_n) = \|w\|_{\infty,X}\,\|1/w\|_{\infty,Y_n}\,C(Y_n)$, we can rewrite the previous inequality as

$$\|wp\|_{\infty,X} \leqslant C_w(Y_n)\,\|wp\|_{\infty,Y_n}.$$

It then follows from (3.8) that the weighted Fekete points $\mathbf{y}_n$ of the space $w\mathbb{R}_n[x_1,\ldots,x_d]$ over Y_n have a Lebesgue constant over X which is bounded like

$$\Lambda_{w\mathbb{R}_n[x_1,\ldots,x_d],X}(\mathbf{y}_n) \leqslant C_w(Y_n)\binom{n+d}{d}.$$

Such bounds are usually overestimates of the actual ones. We can try improving them by using a combination of **P1**–**P7**. Possible approaches include taking advantage of X's structure (if it is a finite union of compact sets) and trying to come up with Markov-like inequalities for weighted polynomial approximation.

The actual rate of increase of Lebesgue constants associated to AFPs extracted from AMs and WAMs seems to be much smaller in practice, with [196, Sec. 3] showing several weighted multi-interval examples where it is $\mathcal{O}(\log n)$, something we also experienced in our tests.

FIR filter design weighting functions

A typical instance of minimax FIR filter design involves a domain $X \subseteq [-1,1], X = \bigcup_{i=1}^k X_i$, where the X_k's are disjoint closed intervals. The weighting function is usually of Jacobi-type, in the sense that

$$w(x) = w_k(1-x)^\alpha(1+x)^\beta, x \in X_k, w_k \in \mathbb{R}, w_k > 0,$$

where $(\alpha,\beta) \in \{(0,0),(1/2,0),(0,1/2),(1/2,1/2)\}$ and each possible value of (α,β) corresponds to one of the four basic filter types introduced in Section 2.2.3, as summarized in Table 3.2.

Consider now that we construct a WAM for X using Chebyshev nodes of the second kind, via Example 3.29. On each $X_i = [a_i,b_i]$ we have a grid $Y_{n,i}$ of size $n+1$, which contains a_i and b_i. Then w will achieve its maximum and minimum value over X_i (and $Y_{n,i}$) at one of these nodes or possibly at 0 if $0 \in X_i$ and we are dealing with a type III filter. In all cases, this implies

$$\|w\|_{\infty,X_i}\|1/w\|_{\infty,Y_{n,i}} = \mathcal{O}(1),$$

meaning that over $Y_n = \bigcup_{i=1}^k Y_{n,i}$, $C_w(Y_n) = \mathcal{O}(C(Y_n)) = \mathcal{O}(\log n)$. Examples are discussed in Section 3.6.4.

Table 3.2: Default weighting values for linear-phase FIR filters ($x = \cos(\omega)$).

Filter type	$W(\omega)$	$w(x)$
I	1	1
II	$\cos\left(\frac{\omega}{2}\right)$	$[(1+x)/2]^{1/2}$
III	$\sin(\omega)$	$(1-x^2)^{1/2}$
IV	$\sin\left(\frac{\omega}{2}\right)$	$[(1-x)/2]^{1/2}$

3.6.3 A reference scaling idea

The limit behavior of (weighted) minimax polynomial approximation problems is an important subject of Potential Theory [133]. Of particular interest to us is the fact that, *asymptotically*, the final reference sets of minimax approximations and the zeros of the minimax error follow a certain distribution over X, called the *equilibrium distribution* of X (see, for instance, [189, Sec. 5] for the necessary theoretical results). Computing it, in all but the simplest setting (the interval), is an involved procedure [81, 189], requiring the use of elliptic integrals and numerical conformal mapping.

Table 3.3: Number of reference values in each band for the bandstop specification of Example 3.19 before the first iteration and at the end of execution.

	$n = 50$	$n = 100$	$n = 200$
$[0, 0.2\pi]$	14/13	26/26	51/51
$[0.3\pi, 0.5\pi]$	14/15	26/31	51/59
$[0.6\pi, \pi]$	24/24	50/45	100/92

An important element developed in the aforementioned articles, confirmed experimentally in our tests, is that the number of reference values for minimax polynomial approximants inside each interval of X is directly proportional with the degree. As an example, we consider the design specification from Example 3.19 and look at what happens when uniform initialization is used within the exchange algorithm. The a/b entries in Table 3.3 show the number of reference points inside each band before the first iteration (the a value) and upon convergence (the b value), for three different degrees. As expected, the b values tend to be proportional with n, while for the second and third band, the corresponding a and b become further apart as $n \to \infty$.

This behavior allowed us to develop a *relatively fast* and *very simple* to implement heuristic for choosing an initial reference. The core of the approach is that, if, for a degree n approximation, we first compute the minimax result of degree $\lfloor n/2 \rfloor$, then we have a very good idea on the number of reference values to put inside each band for the degree n problem. The values of the new reference set $\mathbf{x}$ are established by taking the $\lfloor n/2 \rfloor + 2$ final references of the smaller result and adding the remaining $n - \lfloor n/2 \rfloor$ points uniformly between them, thus ensuring a *similar* distribution of the references inside X for the two degrees. If needed, this strategy can be applied recursively. Since it is based on asymptotic results, this idea works best for high degree designs, where the AFP and DLP strategies become too expensive.

3.6.4 Numerical examples

Performance-wise, Table 3.4 shows some results when considering Examples 3.18 to 3.21. For each entry of the Iterations columns, the first value corresponds to the uniform initialization technique, while the second and third ones represent our scaling and AFP approaches. We can observe that, in many cases, there is a considerable reduction in the number of iterations required for convergence. This frequently translates to smaller execution times as well, despite a more involved setup of the starting references. Still, the biggest advantage of these heuristics is that often, they allow us to address, in a simple manner, filter design problems where uniform initialization gives rise to numerical instability issues and the implementation does not converge (NC). We also tried the strategies from [170, 191], but encountered numerical problems on

Table 3.4: Iteration count comparison for uniform/reference scaling/ AFP-based initialization

Example 3.18		Example 3.19		Example 3.20		Example 3.21	
Degree	Iterations	Degree	Iterations	Degree	Iterations	Degree	Iterations
$n = 50$	11/4/6	$n = 50$	14/14/4				
$n = 80$	8/3/4	$n = 80$	13/3/12	$n = 520$	$12^*/3/1$	$n = 53248$	NC/3/NC
$n = 100$	9/8/3	$n = 100$	$23^*/18/16$				

the larger designs of Examples 3.20 and 3.21. For [170], the quadratic linear solver used to determine the least-squares optimal filter did not work accurately, while for the heuristic of [191], the starting reference did not contain a good proportion of values inside each band for the algorithm to converge numerically.

In dealing with Example 3.20 we applied reference scaling two times, starting from a degree $n = 130$ filter and then moving up to $n = 260$ and finally $n = 520$. We only needed $4, 3$ and 3 iterations for convergence. AFP initialization performed even better, only requiring one iteration. For Example 3.21, we again applied reference scaling two times, for a number of $6, 3$ and 3 iterations, whereas AFP proved to be too costly. Note that, actually, convergence with uniform initialization for Examples 3.19 and 3.20 (the two $*$-ed cases) occurs in spite of numerical problems: this behavior is fortunate and does not generally happen in such a bad numerical context. See also Remark 3.30 for more insight.

Table 3.5: Statistics for the exchange algorithm execution on Example 3.18.

	Minimax	Uniform initialization	Chebyshev WAM + AFP	Reference scaling
Degree	$\|f - p^\star\|_{\infty,X}$ Distribution	$\|f - p_1\|_{\infty,X}$ $\|h_1\|$ Initial distribution	$\|f - p_1\|_{\infty,X}$ $\|h_1\|$ Initial distribution	$\|f - p_1\|_{\infty,X}$ $\|h_1\|$ Initial distribution
$n = 50$	$5.113 \cdot 10^{-5}$ 29 23	$1.272 \cdot 10^{-3}$ $8.188 \cdot 10^{-8}$ 28 24	$1.937 \cdot 10^{-4}$ $1.579 \cdot 10^{-5}$ 29 23	$9.098 \cdot 10^{-5}$ $4.292 \cdot 10^{-5}$ 29 23
$n = 80$	$4.22 \cdot 10^{-7}$ 45 37	$6.686 \cdot 10^{-5}$ $1.01 \cdot 10^{-11}$ 45 37	$2.331 \cdot 10^{-6}$ $1.709 \cdot 10^{-7}$ 45 37	$6.685 \cdot 10^{-7}$ $3.491 \cdot 10^{-7}$ 45 37
$n = 100$	$1.616 \cdot 10^{-8}$ 56 46	$1.167 \cdot 10^{-5}$ $1.796 \cdot 10^{-14}$ 56 46	$5.45 \cdot 10^{-8}$ $1.053 \cdot 10^{-8}$ 56 46	$1.587 \cdot 10^{-4}$ $9.032 \cdot 10^{-9}$ 57 45

Table 3.6: Statistics for the exchange algorithm execution on Example 3.19.

	Minimax	Uniform initialization	Chebyshev WAM + AFP	Reference scaling
Degree	$\|f - p^\star\|_{\infty,X}$ Distribution	$\|f - p_1\|_{\infty,X}$ $\|h_1\|$ Initial distribution	$\|f - p_1\|_{\infty,X}$ $\|h_1\|$ Initial distribution	$\|f - p_1\|_{\infty,X}$ $\|h_1\|$ Initial distribution
$n = 50$	$5.51 \cdot 10^{-5}$ 24 15 13	$9.626 \cdot 10^{-4}$ $1.649 \cdot 10^{-7}$ 24 14 14	$1.877 \cdot 10^{-4}$ $1.669 \cdot 10^{-5}$ 23 16 13	$2.994 \cdot 10^{-3}$ $2.653 \cdot 10^{-5}$ 22 17 13
$n = 80$	$3.472 \cdot 10^{-7}$ 36 25 21	$8.042 \cdot 10^{-5}$ $5.453 \cdot 10^{-14}$ 38 23 21	$1.927 \cdot 10^{-6}$ $1.093 \cdot 10^{-8}$ 37 24 21	$1.096 \cdot 10^{-6}$ $3.071 \cdot 10^{-7}$ 36 25 21
$n = 100$	$1.177 \cdot 10^{-8}$ 45 31 26	$2.315 \cdot 10^{-5}$ $9.546 \cdot 10^{-18}$ 50 26 26	$4.272 \cdot 10^{-8}$ $5.419 \cdot 10^{-9}$ 46 30 26	$5.386 \cdot 10^{-6}$ $1.641 \cdot 10^{-10}$ 48 29 25

Tables 3.5 and 3.6 give more details on the execution of the exchange algorithm. We see that in both the case of AFP-based references and scaled ones, the starting levelled errors h_1 are quite close to δ_X (which is not generally true when uniform initialization is used). Since the levelled error h_k strictly increases at each step and converges to δ_X, we expect that the number of iterations to be generally small. Equally important is the fact that the number of reference values inside each band at the start of the exchange algorithm are very close (if not equal) to their values upon convergence. Intuitively, this means that not a lot of iterations will be spent on moving references between intervals.

Inside the test files, we have also provided code for the design of over 70 different filters that are considered high degree (with n in the order of hundreds and thousands). While the reduction in number of iterations, when computable, was very variable (savings from 0% to over 70% compared to uniform initialization were observable on most of the examples), reference scaling *always* ensured convergence, whereas for more than 30% of the considered filters, uniform initialization failed to converge. The AFP-based idea also performed very well, with only a small percentage (around 5%) of the tests failing. The examples where this happened had degree $n \geqslant 1000$ and were much better suited for the scaling approach.

3.7 Computing the current approximation

Making abstraction of the iteration number k, Step 2 of Algorithm 2 asks one to, for a given reference vector $\mathbf{x} \in \mathbb{R}^{n+2}$, determine $p \in w\mathbb{R}_n[x]$ and $h \in \mathbb{R}$ such that

$$f(x_i) - p(x_i) = (-1)^i h, \qquad i = 0, \ldots, n+1. \tag{3.12}$$

By taking $p(x) = \sum_{i=0}^{n} \alpha_i w(x) T_i(x)$ and $g(x) = \frac{f(x)}{w(x)}$, we can express (3.12) in linear system form

$$\begin{bmatrix} 1 & T_1(x_0) & T_2(x_0) & \cdots & T_n(x_0) & \dfrac{1}{w(x_0)} \\ 1 & T_1(x_1) & T_2(x_1) & \cdots & T_n(x_1) & \dfrac{-1}{w(x_1)} \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 1 & T_1(x_{n+1}) & T_2(x_{n+1}) & \cdots & T_n(x_{n+1}) & \dfrac{(-1)^{n+1}}{w(x_{n+1})} \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_n \\ h \end{bmatrix} = \begin{bmatrix} g(x_0) \\ g(x_1) \\ \vdots \\ g(x_n) \\ g(x_{n+1}) \end{bmatrix}$$

This system has a unique solution, since the $(n+2) \times (n+2)$ matrix is non-singular (see [217] for a proof idea). It is also a consequence of Theorem 3.9. According to [11, Ch. 15.3.3] and [158, Ch. 7.7.3], solving it directly is not encouraged, since it is computationally inefficient and susceptible to numerical ill-conditioning.

In practice, one deduces h analytically and uses barycentric Lagrange interpolation to determine p (*i.e.,* the original choice of [164]). The entirety of this section focuses on the numerical stability of this approach and that of computing h during the execution of the exchange method.

3.7.1 Barycentric interpolation formulas

In recent years, barycentric interpolation has become an active research topic in Numerical Analysis. This is mostly due to the good numerical properties of this interpolation scheme [24, 83].

The basic setting is the following: if $f : [a, b] \to \mathbb{R}$ and $n \in \mathbb{N}$, let $\mathbf{x} \in \mathbb{R}^{n+2}$ be a vector of distinct interpolation points $x_k \in [a, b], k = 0, \ldots, n+1$, given in increasing order, together with $\mathbf{y} \in \mathbb{R}^{n+2}$, where $y_k = f(x_k), k = 0, \ldots, n+1$. We want to find a polynomial $p_{\mathbf{x}}$ with real coefficients of degree at most $n+1$ which interpolates f at $\mathbf{x}$ (*i.e.,* $p_{\mathbf{x}}(x_k) = y_k, k = 0, \ldots, n+1$).

According to Rutishauser [179], the barycentric forms of $p_{\mathbf{x}}$ are given by

$$p_{\mathbf{x}}(x) = \ell(x) \sum_{k=0}^{n+1} \frac{w_k}{x - x_k} y_k, \tag{3.13}$$

known as the *first* barycentric interpolation formula, and

$$p_{\mathbf{x}}(x) = \frac{\displaystyle\sum_{k=0}^{n+1} \frac{w_k}{x - x_k} y_k}{\displaystyle\sum_{k=0}^{n+1} \frac{w_k}{x - x_k}}, \tag{3.14}$$

the *second* (or *proper*) barycentric formula, where $\ell(x) = \prod_{i=0}^{n+1}(x - x_i)$ and

$$w_k = \frac{1}{\ell'(x_k)} = \frac{1}{\prod_{i \neq k}(x_k - x_i)}, \qquad k = 0, \ldots, n+1. \tag{3.15}$$

Formula (3.14) is obtained from (3.13) since by looking at the representation of the constant function $f(x) = 1$ we have

$$\ell(x) \sum_{k=0}^{n+1} \frac{w_k}{x - x_k} = 1. \tag{3.16}$$

If the barycentric weights w_k have been precomputed, both (3.13) and (3.14) can be evaluated with only $\mathcal{O}(n)$ arithmetic operations at an arbitrary point x, the same complexity as Newton's interpolation formula once the corresponding divided differences have been determined. An essential difference between the two is the fact that divided differences have to be recomputed for each new f, whereas the w_k's only depend on the interpolation grid (see [24, Sec. 3] for a discussion on Newton versus barycentric Lagrange interpolation).

In most practical situations, equation (3.14) is usually preferred. One of the reasons for this, as pointed out by Berrut and Trefethen [24], is that the symmetry afforded to it by the presence of weighting functions in both numerator and denominator helps it avoid some underflow and/or overflow issues when n is very large (*i.e.,* in the order of thousands or more).

By changing the w_k's to take other non-zero values, then (3.16) will no longer be valid and (3.14) will become a rational function interpolating f at the x_i's. This more general setting was first explored by Schneider and Werner [183]. Moreover, Berrut and Mittlemann [22] show that any rational interpolant to f at the points $x_0, \ldots, x_{n+1}$, having numerator and denominator degrees not exceeding $n+1$ can be written in barycentric form (3.14) for some choice of weights. One reason for considering barycentric rational interpolants is that they can be good choices when doing fitting on equispaced data (we refer in particular to the family of Floater-Hormann interpolants [83]) where polynomials can behave very badly due to the Runge phenomenon [178]. A classical example is to take the function $f(x) = \frac{1}{1+25x^2}, x \in [-1, 1]$, where polynomial interpolants at the nodes $x_i = -1 + \frac{2i}{n}, i = 0, \ldots, n$ actually diverge when $n \to \infty$, a result which should not be that surprising if we consider the last part of Theorem 3.25.

To use barycentric interpolation for our evaluation of $p(x)$, we first compute h using the formula

$$h = \frac{\displaystyle\sum_{k=0}^{n+1} w_k \frac{f(x_k)}{w(x_k)}}{\displaystyle\sum_{k=0}^{n+1} \frac{(-1)^k w_k}{w(x_k)}} \tag{3.17}$$

(see [163, Sec. 3.3] for a proof idea).

We then have

$$p(x) = w(x) p_{\mathbf{x}}(x) = w(x) \frac{\displaystyle\sum_{k=0}^{n+1} \frac{w_k}{x - x_k} f_k}{\displaystyle\sum_{k=0}^{n+1} \frac{w_k}{x - x_k}} \tag{3.18}$$

where $f_k = \dfrac{f(x_k)}{w(x_k)} - (-1)^k \dfrac{h}{w(x_k)}$ and $\mathbf{w} = (w_i)_{0 \leqslant i \leqslant n+1} \in \mathbb{R}^{n+2}$ is the weight vector whose elements are determined according to (3.15).

3.7.2 Numerical stability issues

The evaluation of (3.17) and (3.18) is split into the computation of the barycentric weights (done once for each new reference vector $\mathbf{x}$) and of the sums in the numerators and the denominators. For a numerically robust evaluation of the w_k's ($\mathcal{O}(n^2)$ operations), we use the formulas and ideas of [24, Sec. 7] and [163, Sec. 3.4]. Notice that inside (3.18) we are using $n+2$ points to interpolate a polynomial of degree at most n. This is intentional. We could have easily picked only a subset of $n+1$ elements of $\mathbf{x}$, but leaving out one element can pose numerical problems, especially if that point is the smallest or largest one from $\mathbf{x}$ [224].

The current levelled error

Let $\alpha_{\mathbf{x}}$ and $\beta_{\mathbf{x}}$ be the numerator and denominator values of h in formula (3.17). For the remainder of this section, unless otherwise stated, all elementary operations are performed in round-to-nearest double-precision arithmetic with unit roundoff $u = 2^{-53}$.

Central to our analysis is the summation condition number [153, Ch. 6.1]: given n real values $a_0, \ldots, a_{n-1}$ whose sum $s_n = \sum_{k=0}^{n-1} a_k$ we want to compute, its corresponding condition number is

$$C_n = \left(\sum_{k=0}^{n-1} |a_k| \right) \Big/ \left| \sum_{k=0}^{n-1} a_k \right|.$$

If we only have access to perturbed versions $\hat{a}_k = a_k(1+\varepsilon_k), k = 0, \ldots, n-1$ of the input terms, then the relative error E_n of computing $\hat{s}_n = \sum_{k=0}^{n-1} \hat{a}_k$ is upper bounded by $\max_k |\varepsilon_k| \, C_n$. Indeed,

$$E_n = \frac{|\hat{s}_n - s_n|}{|s_n|} = \frac{\left| \sum_{k=0}^{n-1} \varepsilon_k a_k \right|}{\left| \sum_{k=0}^{n-1} a_k \right|} \leqslant \frac{\sum_{k=0}^{n-1} |\varepsilon_k| \, |a_k|}{\left| \sum_{k=0}^{n-1} a_k \right|} \leqslant \max_k |\varepsilon_k| \, \frac{\sum_{k=0}^{n-1} |a_k|}{\left| \sum_{k=0}^{n-1} a_k \right|}.$$

Even if the perturbations are as small as possible with the current arithmetic, meaning $\varepsilon_k = \mathcal{O}(u), k = 0, \ldots, n-1$, if there is severe cancellation in computing s_n and $C_n \gg u^{-1}$, then E_n will most likely be greater than 1 and give totally inaccurate results. Since $\alpha_{\mathbf{x}}$ and $\beta_{\mathbf{x}}$ are obtained using finite precision arithmetic, in certain situations, there is pronounced numerical cancellation when computing $\alpha_{\mathbf{x}}$. This can happen when:

- the minimax error at the end of executing the exchange algorithm is very small (in the sense of being relatively close to or smaller than u);
- the final minimax error is much larger than u, but the starting leveled error h is in the order of u or much smaller.

In practice, the second bullet point is more likely to occur than the first one, since polynomial filters with minimax error $\delta_X < 10^{-10} (\gg u \sim 10^{-16})$ seldom seem to be required.

When using uniform initialization, such issues are not hard to find, as they sometimes occur in the starting iteration(s) of the exchange method. Consider, for instance, the specification from Example 3.20. Although the final alternating error is around $1.6067 \cdot 10^{-7}$, the first iteration should give $h \simeq 1.521 \cdot 10^{-21} \ll u$ and a condition number for $\alpha_{\mathbf{x}}$ of about $6.572 \cdot 10^{20} \simeq 72967 \cdot u^{-1}$. Because of this extreme ill-conditioning, a double-precision computation of h gives a wrong value of $5.753 \cdot 10^{-19}$.

Based on our discussion up to this point, we propose two possible solutions:

1. use a higher precision arithmetic for doing *all* computations in the current iteration (barycentric weights, h and local extrema of $e(x) = f(x) - p(x)$);

2. pick a *better* starting reference (see initialization strategies from Section 3.5.1).

The first alternative is generally the safest, but costlier choice. From our experience, around 150-200 bits of precision are more than enough for the problems discussed here. Hence, the `MPFR`-based version of our code uses 165 bits of precision by default. Once enough iterations have passed and the current condition number for $\alpha_{\mathbf{x}}$ is sufficiently small (*i.e.,* smaller than say 10^8), there is a good chance that using double-precision arithmetic for the remaining iterations will still lead to an acceptable result.

For the second strategy, the basic intuition is that the starting h should be much larger and closer to the minimax one (see also Tables 3.5 and 3.6). This consequently implies that the condition number for $\alpha_{\mathbf{x}}$ is much smaller compared to u^{-1} and the summations are more stable. To illustrate this, we again consider Example 3.20, but this time, use the reference scaling strategy from Section 3.6.3. The initial leveled error is $h \simeq 1.456 \cdot 10^{-7} \gg u^{-1}$ and the condition number for $\alpha_{\mathbf{x}}$ is around $6.864 \cdot 10^6$. Double-precision computations are again quite viable. Since it usually results in much smaller execution times, we recommend starting with this second approach.

Numerical problems are much less likely to occur when computing $\beta_{\mathbf{x}}$. Since we assume the interpolation nodes $\mathbf{x}$ are ordered by value, the barycentric weights w_i alternate in sign. This means that all the terms of $\beta_{\mathbf{x}}$ have the same sign and hence, its condition number has value 1.

Lebesgue constants as tools for detecting numerical problems

The numerical stability of (3.13) and (3.14) over $[-1, 1]$ was first looked at in a rigorous manner by Rack and Reimer [173] and later refined by Higham [103]. This second reference shows how the modified Lagrange formula (3.13) is backward stable in general, whereas (3.14) is shown to be forward stable for vectors of points $\mathbf{x} \in \mathbb{R}^{n+2}$ having a small Lebesgue constant $\Lambda_{\mathbb{R}_n[x],[-1,1]}(\mathbf{x})$. Higham goes on to reinforce this experimentally, by comparing the two barycentric expressions with the Newton divided difference interpolation formula for different vectors $\mathbf{x}$ inside $[-1, 1]$.

Although the aforementioned paper points out that the error analysis for formula (3.14) is not as favorable as the one for (3.13), we should not infer from this a *general* conclusion that the second barycentric formula is less stable than the first one. This seems to be especially true when Chebyshev nodes of the second kind are concerned. In a recent account, Mascarenhas [143] shows that if we use the closed form of the barycentric weights for second kind Chebyshev nodes as established by Salzer [180], meaning

$$w_k = (-1)^k \delta_k, \qquad \delta_k = \begin{cases} 1/2, & k = 0 \text{ or } k = n+1, \\ 1, & \text{otherwise;} \end{cases}$$

then the second barycentric formula can be shown to be backward stable with respect to perturbations in the values of f. If f turns out to also have a derivative of moderate size, this will also lead to small forward errors.

On the other hand, for this particular setting, using the first barycentric formula for large n can be problematic. That is to say, if rounded (or perturbed) versions $\hat{x}_k$ of the nodes x_k are used during the evaluation of (3.13), then the relative error of computing $p_{\mathbf{x}}(x)$ can actually be of order $n^2 u$. This leads to approximations which are less stable than if using the second barycentric formula. Examples with $f(x) = \sin(x)$ and $n \geqslant 10^3$ are considered. Possible ways of making the first barycentric formula more stable in this case are discussed in [145].

The main idea we are trying to convey is that, when doing polynomial interpolation, one should first consider using (3.14). To support this claim, we also mention [144], where Mascarenhas and Camargo argue that (3.14) is backward stable if the relevant Lebesgue constant corresponding to $\mathbf{x}$ is small.

An example of a reference set with small Lebesgue constant is showcased in Figures 3.5 and 3.6 for the specification given in Example 3.20. Both plots show the relative errors in computed $p(x)$ during the first iteration of the exchange algorithm. The 'exact' computations were performed with `MPFR` using 600 bits of precision (roughly 180 digits). In Figure 3.5, $\Lambda_{\mathbb{R}_n[x],[-1,1]}(\mathbf{x})$ is quite large and we see that the loss in accuracy is clearly visible around $x = -1$. The results are much better in Figure 3.6 for a smaller Lebesgue constant, reinforcing our earlier statements.

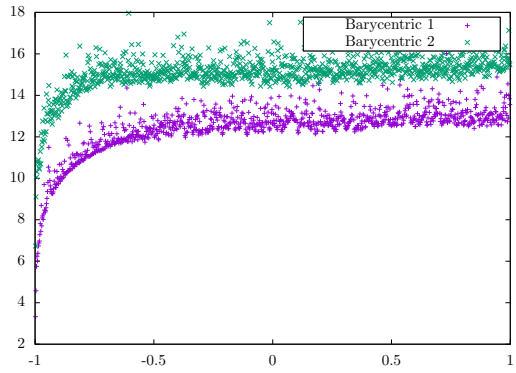

Figure 3.5: Relative errors (correct significant digits) when computing the starting p for Example 3.20 with both types of barycentric formulas. Uniform initialization ($\Lambda_{\mathbb{R}_n[x],[-1,1]}(\mathbf{x}) \simeq 4.24973 \cdot 10^{14}$) is used.

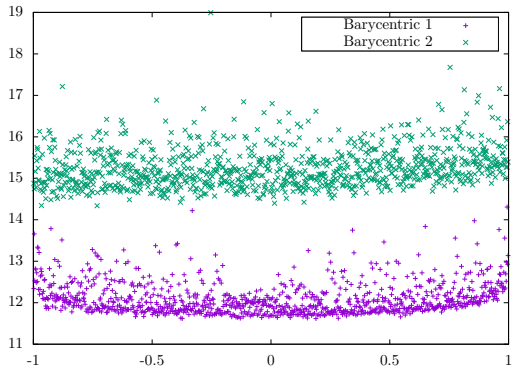

Figure 3.6: Relative errors (correct significant digits) when computing the starting p for Example 3.20 using both types of barycentric formulas. Reference scaling ($\Lambda_{\mathbb{R}_n[x],[-1,1]}(\mathbf{x}) \simeq 1.29773 \cdot 10^{5}$) is used.

Table 3.7: Lebesgue constant evolution during the execution of the exchange algorithm (*i.e.*, first, middle and last iteration). The starting reference set is computed using uniform intialization.

Example 3.18 $n = 100$		Example 3.19 $n = 100$		Example 3.20 $n = 520$	
Iteration	$\Lambda_{\mathbb{R}_n[x],[-1,1]}(\mathbf{x})$	Iteration	$\Lambda_{\mathbb{R}_n[x],[-1,1]}(\mathbf{x})$	Iteration	$\Lambda_{\mathbb{R}_n[x],[-1,1]}(\mathbf{x})$
1	$3.72896 \cdot 10^{10}$	1	$3.00467 \cdot 10^{15}$	1	$4.24973 \cdot 10^{14}$
5	$8.31678 \cdot 10^{6}$	12	$7.45988 \cdot 10^{7}$	6	$6.11242 \cdot 10^{5}$
9	$4.00568 \cdot 10^{6}$	23	$1.35889 \cdot 10^{7}$	12	$1.38665 \cdot 10^{5}$

Table 3.8: Lebesgue constant evolution when the reference scaling approach described in Section 3.6 is used.

Example 3.18 $n = 100$		Example 3.19 $n = 100$		Example 3.20 $n = 520$	
Iteration	$\Lambda_{\mathbb{R}_n[x],[-1,1]}(\mathbf{x})$	Iteration	$\Lambda_{\mathbb{R}_n[x],[-1,1]}(\mathbf{x})$	Iteration	$\Lambda_{\mathbb{R}_n[x],[-1,1]}(\mathbf{x})$
1	$5.98098 \cdot 10^{6}$	1	$4.12072 \cdot 10^{8}$	1	$1.29773 \cdot 10^{5}$
4	$4.68392 \cdot 10^{6}$	9	$1.04020 \cdot 10^{7}$	2	$1.38250 \cdot 10^{5}$
8	$4.00763 \cdot 10^{6}$	18	$1.35728 \cdot 10^{7}$	3	$1.38015 \cdot 10^{5}$

Table 3.9: Lebesgue constant evolution when the AFP-based approach described in Section 3.6 is used.

Example 3.18 $n = 100$		Example 3.19 $n = 100$		Example 3.20 $n = 520$	
Iteration	$\Lambda_{\mathbb{R}_n[x],[-1,1]}(\mathbf{x})$	Iteration	$\Lambda_{\mathbb{R}_n[x],[-1,1]}(\mathbf{x})$	Iteration	$\Lambda_{\mathbb{R}_n[x],[-1,1]}(\mathbf{x})$
1	$4.06753 \cdot 10^{6}$	1	$3.59135 \cdot 10^{8}$		
2	$4.0023 \cdot 10^{6}$	8	$9.90082 \cdot 10^{7}$	1	$1.38015 \cdot 10^{5}$
3	$4.00476 \cdot 10^{6}$	16	$1.35837 \cdot 10^{7}$		

Typical scenarios of how $\Lambda_{\mathbb{R}_n[x],[-1,1]}(\mathbf{x})$ changes are given in Tables 3.7 to 3.9 for Examples 3.18 to 3.20. The numerical estimates were computed using the Chebfun routine `lebesgue` mentioned in [205, Ch. 15]. They tell us that execution will tend to be more stable during latter iterations, with the risk that the starting iterations for uniform initialization are prone to extreme ill-conditioning. The effect of using the

strategies from Section 3.6 is also clearly visible in Tables 3.8 and 3.9, leading to better numerically behaved computations from the beginning. At this point, one can conjecture that the Lebesgue constants on $[-1,1]$ associated with the references that appear during execution of the exchange algorithm generally tend to decrease in value when starting far from the optimal reference.

Why is this? An intuitive, but still theoretically grounded, reason is related to the equilibrium distribution of equialternating sets for best approximation mentioned in Section 3.6.3. By taking an $\mathbf{x}$ that follows such a distribution, its Lebesgue constant $\Lambda_{w\mathbb{R}_n[x],X}(\mathbf{x})$ will generally be small (take for instance the Fekete points of X). This effect seems to carry over to $\Lambda_{\mathbb{R}_n[x],[-1,1]}(\mathbf{x})$, provided:

(1) the intervals of $[-1,1]-X$ are small compared to those of X;

(2) if (1) does not hold, then n is not too large.

Both AFPs and scaled nodes follow such distributions and it is reasonable to assume that subsequent references during execution will do so too. Our statements are reinforced by the fact that Lebesgue constant values are stable under small perturbations of the sampling nodes [167].

Since a large Lebesgue constant $\Lambda_{\mathbb{R}_n[x],[-1,1]}(\mathbf{x})$ is symptomatic for a very small levelled error h, the solutions for countering ill-conditioning of the previous subsection are very much true for evaluating (3.18).

Remark 3.30. When numerical problems of the types discussed in this section occur (typically because of uniform initialization), our exchange algorithm implementation using double-precision arithmetic can sometimes still find the correct result. What happens in such cases is that, even though our computations of the current interpolant and its approximation error are inaccurate, the next reference we retrieve from them, despite being wrong, can have a small Lebesgue constant. Having such a reference, as we saw, usually improves chances for convergence. It can be viewed as a kind of reinitialization midway through execution. Nevertheless, as our test cases show, convergence in the presence of numerical issues is a completely random behavior. It is not to be expected and, most of all, trusted. One should *definitely* consider using a higher working precision and/or one of the initialization techniques from Section 3.6.

3.8 Extrema search

Let's denote by $\mathbf{x}' \in \mathbb{R}^{n+2}$ the reference vector at iteration $k+1$ in Step 3 of Algorithm 2. To compute its elements, we want to know the local extrema of the current error function $e(x)$. For this purpose, Chebyshev root-finding is reviewed in the next paragraphs, while Section 3.8.2 introduces our non-recursive subdivision strategy. For completeness, Sections 3.8.3 and 3.8.4 talk about updating the reference set at each iteration and how to retrieve the minimax result coefficients upon convergence.

3.8.1 Chebyshev-proxy root-finding

We first consider the slightly different problem of determining the zeros of a function $f \in \mathcal{C}([-1,1])$, located inside $[-1,1]$. We note that by suitable changes of variable, the following results hold for any closed interval $[a,b]$. The idea of the Chebyshev-proxy root-finder (CPR) method [34, 205] is to replace $f(x)$ by a degree m Chebyshev interpolant *proxy* $p_m(x)$. If the chosen polynomial is an accurate enough approximation of f, then its zeros will very closely match those of f.

Recall from Section 2.1.2 that p_m interpolates f at the Chebyshev nodes of the second kind $\nu_k = \cos((m-k)\pi/m), 0 \leqslant k \leqslant m$, and is expressed using the basis of Chebyshev polynomials of the first kind. We have

$$p_m(x) = \sum_{k=0}^{m} a_k T_k(x), \qquad x \in [-1,1], a_k \in \mathbb{R}, k = 0, \ldots, m,$$

with $p_m(\nu_k) = f(\nu_k), k = 0, \ldots, m$ and

$$a_k = \frac{1}{m} f(\nu_0) T_k(\nu_0) + \frac{2}{m} \sum_{i=1}^{m-1} f(\nu_i) T_k(\nu_i) + \frac{1}{m} f(\nu_m) T_k(\nu_m), \qquad k = 0, \ldots, m. \tag{3.19}$$

The roots of p_m are then computed as the eigenvalues of a generalized companion matrix [33, 70], called *colleague matrix*, which behaves well in practice [35]:

Theorem 3.31 (Colleague matrix)**.** *The roots of the polynomials*

$$p_m(x) = \sum_{i=0}^{m} a_i T_i(x), \qquad a_m \neq 0$$

correspond to the eigenvalues of the matrix

$$\mathbf{C}(p_m) = \begin{bmatrix} 0 & 1 & & & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & & & \\ & \frac{1}{2} & 0 & \frac{1}{2} & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & & & & \frac{1}{2} \\ -\frac{a_0}{2a_m} & -\frac{a_1}{2a_m} & -\frac{a_2}{2a_m} & & & \frac{1}{2} - \frac{a_{m-2}}{2a_m} & -\frac{a_{m-1}}{2a_m} \end{bmatrix}, \tag{3.20}$$

where the entries not displayed are zero. If there are multiple roots, these correspond to eigenvalues with the same algebraic multiplicities.

Proof. See for example [205, Ch. 18]. □

To obtain good approximations of the local extrema of $e(x)$, suppose we already have an accurate degree m proxy

$$e_m(x) = \sum_{k=0}^{m} a_k T_k(x).$$

The value of m should be thought of independently from the degree n of the target minimax response $p^\star(x)$. We look for the roots of $e'_m(x) = \dfrac{\mathrm{d}e_m}{\mathrm{d}x}$, whose Chebyshev expansion is

$$e'_m(x) = \sum_{k=0}^{m-1} b_k T_k(x),$$

where: $b_{k-1} = 2ka_k + b_{k+1}, k = 1, \ldots, m$, and $b_m = b_{m+1} = 0$. Similarly, if we take into account that $\dfrac{\mathrm{d}T_m}{\mathrm{d}x} = mU_{m-1}$, for $m \geqslant 1$, we have

$$e'_m(x) = \sum_{k=0}^{m-1} c_k U_k(x),$$

where $c_{k-1} = ka_k, k = 1, \ldots, m$. Although the CPR method is introduced in the context of T_m, we found the second formula for e'_m more natural to use in our setting, since it is simpler to compute; each c_{k-1} coefficient requires only one multiplication. The colleague matrix corresponding to an expansion using the basis of Chebyshev polynomials of the second kind U_i has basically the same form as (3.20), with the sole difference that the 1 in the first row of $\mathbf{C}(p_m)$ is replaced by a $\frac{1}{2}$.

To find the eigenvalues of a $m \times m$ colleague matrix, for both types of Chebyshev polynomials, we can use a QR/QZ algorithm ($\mathcal{O}(m^3)$ operation count). For numerical reasons, we use the colleague matrix form with non-zero coefficients on the first column, as suggested in [205, Ex. 18.3]:

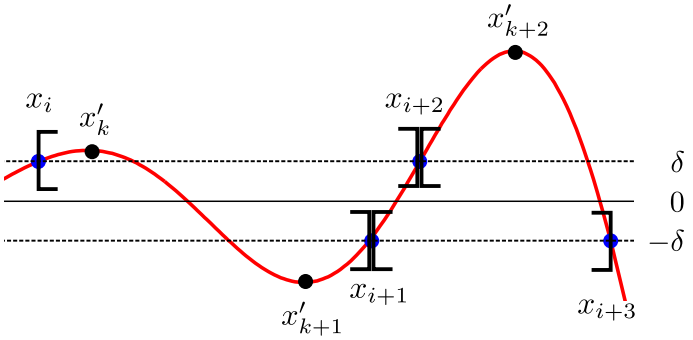

Figure 3.7: Our interval subdivision strategy for computing the extrema of $e(x)$.

$$\begin{bmatrix} -\frac{a_{m-1}}{2a_m} & \frac{1}{2} & & & & & \\ \frac{1}{2} - \frac{a_{m-2}}{2a_m} & 0 & \frac{1}{2} & & & & \\ -\frac{a_{m-3}}{2a_m} & \frac{1}{2} & 0 & \frac{1}{2} & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & & & 1 & \\ -\frac{a_0}{2a_m} & & & & \frac{1}{2} & 0 & \end{bmatrix},$$

preceded by a preconditioning phase, known as *balancing* [165]. It modifies the given matrix in such a way that the eigenvalues are invariant, whereas the norm of the new matrix is smaller.

The cost can be reduced to $\mathcal{O}(m^2)$ computations by dividing the initial domain $[-1, 1]$ into several subintervals and taking Chebyshev interpolation polynomials of smaller degree on each of them. The roots can then be determined as the collection of zeros inside all subintervals. More precisely: if $N_{\max} < m$ is the maximum degree on which one is willing to use an eigenvalue algorithm, *recursively* split $[-1, 1]$ until there are Chebyshev interpolants of degree at most $N_{\max}$ on each subinterval, which approximate f accurately.

There are several general criteria that quantify when we have a good enough approximation of $f(x)$ [33,34]. The Chebfun Remez algorithm implementations [111, 163] use such a recursive approach [205, Ch. 18] by means of the Chebfun `roots` command. Note that the intermediary interpolants computed before the final stage subdivisions are not used for the actual root finding.

3.8.2 A new subdivision strategy

We use a different, *non recursive* approach. In general, during each iteration of the exchange algorithm, we have estimates about the location and number of extrema of e; if we take a subinterval defined by two consecutive points of the current reference vector $\mathbf{x}$ located inside the same band, say $[x_i, x_{i+1}]$, then we *usually* expect to have between zero and two potential extrema of e inside it. Figure 3.7 summarizes this scenario. We then use a degree $N_{\max}$ Chebyshev proxy on each subinterval. Values of $N_{\max}$ we frequently used in our experiments are 4 and 8. Dynamic range issues [34, Sec. 7.4] with the values of $e(x)$ are generally absent because the initial approximations from Section 3.6 are of good quality. Since there is only one subdivision level (no intermediary interpolants), there will be savings in computation time.

At each iteration, we use $\mathcal{O}(n)$ subintervals of the form $[x_i, x_{i+i}]$. To cover all of X, we also consider all intervals of the form $[a, x_i] \subseteq X$ and $[x_j, b] \subseteq X$, where a and b are band extremities of X and x_i and x_j are elements of $\mathbf{x}$ which are closest to a and b, respectively. Inside each subinterval, we evaluate the error $N_{\max} + 1$ times in order to construct a suitable proxy for e and its derivative. Because of barycentric interpolation, each evaluation of e will usually require only $\mathcal{O}(n)$ operations, while solving each small

eigenvalue problem amounts to a constant number of computations. The total operation count will be $\mathcal{O}(n^2)$.

In addition to requiring the evaluation of e a reasonable number of times (in the range of $4n$ to $8n$), this approach can also be very easily parallelized (the computations on each subinterval are *data parallel*). For large degree approximations on multicore systems, the speedup is considerable (see Section 3.9.2).

Even faster iterations

This quadratic extrema search scheme gives very reasonable execution times in practice, even for degrees in the order of thousands. Nevertheless, potential speedups are still possible; because of *apriori* knowledge on the *exact* number of evaluations of e involving formula (3.18), it is feasible to use fast multipole methods for evaluating polynomial interpolants [79] at n arbitrary points. If ε is the numerical precision for our computations, such methods need $\mathcal{O}(n \log(1/\varepsilon))$ operations at each iteration. Parallelization is possible here as well; the $\mathcal{O}(N_{\max} n)$ polynomial evaluations can be split in $\mathcal{O}(N_{\max})$ *independent* applications of the fast multipole method on batches of n points. The only issue is that, from a numerical point of view, such computations can be less stable in practice than if barycentric interpolation is used [79, Example 3].

3.8.3 Updating the current reference set

In the *classic* version of the multi-point exchange rule on $X = [a, b]$, the new reference $\mathbf{x}'$ is determined by Algorithm 6. An explicit computation of the z_i is avoidable in practice. A proper $\mathbf{y}$ can be obtained if we know the sign of e at its local extrema and their position with respect to the nodes of $\mathbf{x}$. Pachón and Trefethen describe a slightly different strategy in [163, Sec. 2.2], but which usually behaves equally well.

Algorithm 6: MULTI POINT EXCHANGE [217]

Input: $X = [a, b], \mathbf{x} \in \mathbb{R}^{n+2}$, the current error function $e(x)$ and levelled error h
Output: new reference vector $\mathbf{x}' \in \mathbb{R}^{n+2}$

1 Determine a vector $\mathbf{z} = (z_i)_{0 \leqslant i \leqslant n}$ such that

$$e(z_i) = 0, \qquad i = 0, \ldots, n,$$

and

$$x_0 \in [a, z_0], x_i \in [z_{i-1}, z_i], \text{ for } i = 1, \ldots, n, x_{n+1} \in [z_n, b].$$

2 Construct $\mathbf{y} \in \mathbb{R}^{n+2}$ such that

$$y_0 \in [a, z_0], y_i \in [z_{i-1}, z_i], \text{ for } i = 1, \ldots, n, y_{n+1} \in [z_n, b]$$

and each y_i is a local extrema of e where $\operatorname{sgn}(e(y_i)) = \operatorname{sgn}(e(x_i))$ and $|e(y_i)| \geqslant |h|$.

3 **if** $\nexists y_i$ s.t. $|e(y_i)| = \|e\|_{\infty, X}$ **then**

    // one point exchange rule

4 Replace one of the y_i's with a point $y \in X$ where $|e(y)| = \|e\|_{\infty, X}$ s.t. e over the new $\mathbf{y}$ still alternates in sign.

5 $\mathbf{x}' \leftarrow \mathbf{y}$

6 **return** $\mathbf{x}'$

For a multi-interval X, one possible approach is to apply Algorithm 6 on each interval of X and then take the union of the obtained partial references. The major issues are:

- the number of reference values inside each interval stays the same from iteration to iteration, impeding convergence in many cases;

- the alternation property on $\mathbf{x}'$ might not be true (*i.e.,* the execution of lines 3–4 on certain intervals can cause sign swaps for the error values at the new reference nodes and no swap on others).

An alternative is to apply lines 1 and 2 on each interval, take $\mathbf{y}$ as the union of the resulting nodes and *only then* potentially perform lines 3–4. This should allow for the migration of reference nodes from one interval to another in certain cases. Other possibilities are described in [11, Ch. 15.3.4]; although they allow the movement of reference points between intervals, they do not seem to enforce the error alternation property on $\mathbf{x}'$, hence incurring the risk of non-convergence in some cases.

Algorithm 7: MULTI INTERVAL EXCHANGE RULE

Input: the current error function $e(x)$ and levelled error h
Output: new reference vector $\mathbf{x}' \in \mathbb{R}^{n+2}$

1 Take $\tilde{\mathbf{x}}$ as the set of all ordered potential extrema $y \in X$ of e such that $|e(y)| \geqslant |h|$.
2 Inside $\tilde{\mathbf{x}}$, for each maximal subset of consecutive elements where the error has the same sign, keep only one with largest absolute error.
3 $m \leftarrow \text{size}(\tilde{\mathbf{x}})$
// if no numerical problems occur, it should be that $m \geqslant n+2$
// if the extra number of values in $\tilde{\mathbf{x}}$ is odd, remove the first or last element
4 **if** $m - n - 2 \equiv 1 \pmod 2$ **then**
// take the one with largest absolute error among them
5 **if** $|e(\tilde{x}_0)| < |e(\tilde{x}_{m-1})|$ **then**
6 remove $\tilde{x}_0$ from $\tilde{\mathbf{x}}$
7 **else**
8 remove $\tilde{x}_{m-1}$ from $\tilde{\mathbf{x}}$
9 $m \leftarrow m - 1$
// if the number of extra elements is even, iteratively remove pairs of successive values, so that the alternation property still holds
10 **while** $m > n + 2$ **do**
11 construct $\mathbf{e} \in \mathbb{R}^{m-1}$ s.t.

$$e_i \leftarrow \max\left(|e(\tilde{x}_i)|, |e(\tilde{x}_{i+1})|\right), \qquad \text{for } i = 0, \ldots, m-2.$$

// remove a pair of values where the max of the absolute errors is smallest
12 **if** $\max\left(|e(\tilde{x}_0)|, |e(\tilde{x}_{m-1})|\right) < \min(\mathbf{e})$ **then**
13 remove $\tilde{x}_0$ and $\tilde{x}_{m-1}$ from $\tilde{\mathbf{x}}$
14 **else**
15 for an element $e_i = \min(\mathbf{e})$, remove $\tilde{x}_i$ and $\tilde{x}_{i+1}$ from $\tilde{\mathbf{x}}$
16 $m \leftarrow m - 2$
17 $\mathbf{x}' \leftarrow \tilde{\mathbf{x}}$
18 **return** $\mathbf{x}'$

For our implementation, we have decided to use another multi-point strategy, inspired by [6, Sec. 4.1] and detailed in Algorithm 7. It is the most robust one we tested. It usually exhibits the same quadratic convergence behavior typical to the multi-point second Remez algorithm over an interval.

Take for instance Tables 3.10 and 3.11. They show how the levelled error h_k evolves with respect to the minimax deviation δ_X. We used uniform initialization (so as to hopefully force a larger number of iterations), `MPFR` multiple precision arithmetic and a tolerance $\varepsilon_t = 2^{-60}$ for Step 4 of Algorithm 2 to highlight that, towards the end of execution, the number of correct digits h_k has with respect to δ_X essentially doubles at each iteration, consistent with (3.5).

Table 3.10: Relative error of h_k with respect to δ_X for Example 3.18, $n = 100$.

Iteration k	$\left\lvert\frac{\delta_X - h_k}{\delta_X}\right\rvert$
1	0.999999
2	0.999969
3	0.999376
4	0.990968
5	0.908509
6	0.595295
7	0.127555
8	$7.53014 \cdot 10^{-3}$
9	$5.20205 \cdot 10^{-5}$
10	$1.85449 \cdot 10^{-9}$
11	$3.26333 \cdot 10^{-18}$

Table 3.11: Relative error of h_k with respect to δ_X for Example 3.19, $n = 100$.

Iteration k	$\left\lvert\frac{\delta_X - h_k}{\delta_X}\right\rvert$
1	1
3	1.00045
5	1.0609
7	1.56985
9	1.74337
10	$1.35944 \cdot 10^{-1}$
11	$3.75978 \cdot 10^{-2}$
12	$2.34496 \cdot 10^{-3}$
13	$7.34568 \cdot 10^{-6}$
14	$7.8493 \cdot 10^{-11}$
15	$5.45774 \cdot 10^{-20}$

3.8.4 Retrieving the minimax coefficients

Upon convergence, we compute the coefficients $(\alpha_i)_{0 \leqslant k \leqslant n}$ of the final approximant p_k obtained from Step 4 in Algorithm 2. As we saw in Section 2.1.2, we can do this in a numerically stable way with a quadratic number of computations by using formula (3.19) and Clenshaw's algorithm or in $\mathcal{O}(n \log n)$ operations with a DCT-based interpolation scheme. Just like in Section 3.8.2, the total cost is dictated by how we evaluate the barycentric form of $p_k(x)$ at the n-th order Chebyshev nodes: $\mathcal{O}(n^2)$ operations, since we use formula (3.18) at each node. If we instead used again the multipole evaluation scheme of [79], we would require $\mathcal{O}(n \log(n/\varepsilon))$ operations (fast polynomial evaluation + DCT-based Chebyshev interpolation).

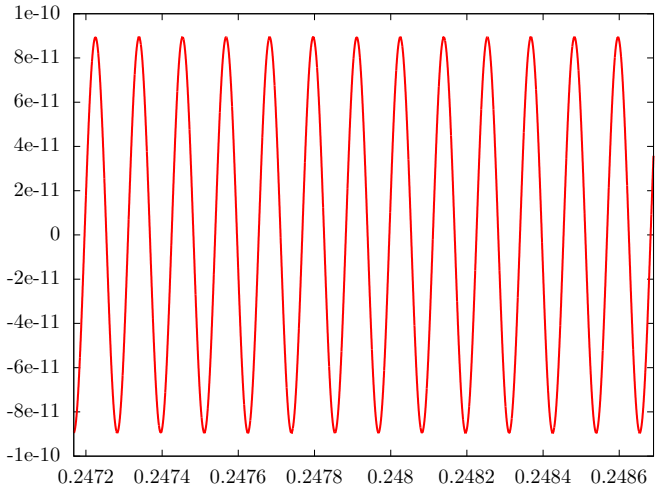

Figure 3.8: The value of the approximation error $e^\star$ on a small interval $[0.247168, 0.24869]$ of X, for the minimax result satisfying the specification for Example 3.21.

Figure 3.8 shows a small sample of the minimax approximation error e_k, for Example 3.21, *after* the filter coefficients have been computed using Clenshaw's recurrence relations. Although the degree $n = 53248$ is quite large, the computations are *numerically accurate*, with the equioscillations of Theorem 3.9 clearly visible.

3.9 Implementation details

In this section we give practical information on how someone can use our routines and compare them to equivalent ones available in widely used Signal Processing packages.

To summarize the previous three sections, each iteration of the exchange algorithm takes $\mathcal{O}(n^2)$ operations with our code (see https://github.com/sfilip/firpm). Choosing the starting reference takes $\mathcal{O}(n)$ operations if uniform initialization is used, but this cost can jump to $\mathcal{O}(n^2)$ with the approach from Section 3.6.3 or $\mathcal{O}(n^3)$ operations for Section 3.6.2. By limiting the use of AFP-based initialization to moderate degrees ($n < 1000$), the cubic cost is not problematic in practice. As discussed in Section 3.6.4, these more expensive strategies are most of the time worthwhile and sometimes even necessary. Also, the computational bottleneck (the extrema search of Section 3.8.2) is *embarrassingly parallel*. Exploiting it is very effective in practice (see Table 3.13).

3.9.1 User interface

Our code requires a small number of external libraries in order to work. Besides calling `MPFR` inside the multiple precision version, we also use the `Eigen` library [94] to perform all the eigenvalue computations when searching for a new reference set. Because it is designed with template metaprogramming techniques, the code calling `Eigen` requires little to no changes between the different versions. To parallelize the extrema search in Section 3.8.2, we use OpenMP [69].

All three versions of our code have an almost identical interface. As such, we will focus on describing only the double-precision version. For filter-centric applications, we have coded functions specifically designed for this task, inspired by the style used for the MATLAB implementation of the Parks-McClellan algorithm.

The function for designing type I and II filters has the following prototype:

```
PMOutput firpm(std::size_t N, std::vector<double>const& f,
               std::vector<double>const& a, std::vector<double>const& w,
               double epsT = 0.01, int Nmax = 4);
```

where

- `N`+1 is the number of coefficients of the designed filter;
- `f` is the vector of the frequency band edges of $\Omega \subseteq [0, \pi]$, normalized to $[0, 1]$, and given in increasing order;
- `a` contains the desired amplitudes at each of the points from `f`;
- `w` is the vector of weights on each band specified by `f` (its size is half that of `f` and `a`);
- `epsT` convergence parameter threshold from Step 4 in Algorithm 2 from Section 3.3.1. The default value is taken from [11, Ch. 15.3].
- `Nmax` designates the degree of the Chebyshev interpolants used for the extrema search from Section 3.8.2, with a default value of 4.

For instance, designing a degree $n = 100$, type I filter adhering to the specification given in Example 3.18, results in the following, valid C++11, function call:

```
PMOutput res = firpm(200, {0, 0.4, 0.5, 1}, {1, 1, 0, 0}, {1, 1});
```

The returned `PMOutput` object `res` has several member variables that can be useful to a filter designer. The vector `res.h` corresponds to the final coefficients of the filter transfer function from equation (2.10), while `res.x` is the final reference vector, belonging to X. The number of iterations required for convergence are stored in the variable `res.iter`, while the value of the final reference error δ_X is denoted with `res.delta`.

The `firpm` functions use uniform initialization by default. To use reference scaling or approximate Fekete points, we supply the functions `firpmRS` and `firpmAFP`. The `firpmRS` versions allow the user to choose between uniform and AFP-based initialization at the lowest level.

More general weighted polynomial approximations can be computed with the function `exchange`, which is used internally by the `firpm` code:

```
PMOutput exchange(std::vector<double>& x, std::vector<Band>& chebyBands,
                  double epsT, int Nmax);
```

It gives the user the freedom to specify the starting reference through the variable `x`, while the (multi)interval domain, together with the values of $f(x)$ and $w(x)$ on each subinterval are specified through the `chebyBands` parameter.

3.9.2 Timings

Table 3.12: Runtime (real time) comparisons of our IEEE-754 double-precision implementation of the second Remez algorithm with those available in GNURadio 3.7.8.1 (also written in C++), MATLAB R2014b, the one from SciPy 0.16.1 (python code) and two other MATLAB implementations. The same machine, a quad core 3.6 GHz 64-bit Intel Xeon(R) E5-1620 running Linux 4.1.12 (15.9), was used for all the tests. The average execution times (in seconds) of running each piece of code 50 times, are given. When a routine did not converge to the minimax result, NC was used in place of the execution time. The a/b entries in the last column correspond to the two implementations described in [6].

Example (degree)	Uniform (sequential)	GNURadio	MATLAB	SciPy	[6]
3.18 ($n = 50$)	0.0034	**0.0029**	0.0532	0.0711	0.0384/0.0098
3.18 ($n = 80$)	**0.0045**	0.0068	0.1174	0.2587	0.0416/0.0316
3.18 ($n = 100$)	**0.0073**	NC	0.1491	0.3511	0.0451/0.0396
3.19 ($n = 50$)	**0.0022**	0.0023	0.0681	0.0971	0.0395/0.0174
3.19 ($n = 80$)	**0.0069**	NC	0.2002	0.3492	0.0761/0.0293
3.19 ($n = 100$)	**0.0301**	NC	NC	NC	0.2599/0.0521
3.20 ($n = 520$)	**0.2375**	NC	NC	NC	1.1691/2.7011
3.21 ($n = 53248$)	NC	NC	NC	NC	NC/NC

There are multiple implementations of the Parks-McClellan algorithm available, so we compared our approach to those we believe are the most widely used and/or robust in practice. Some results are given in Table 3.12. Because they are written in different languages, there are bound to be some differences in terms of execution times. To make matters as *fair* as possible, we *disabled parallelization* of the extrema search in our code and went for *uniform initialization* in all the test cases. Default grid size parameters (see the discussion from Remark 3.16) were used for the routines in the last four columns and a default value of $N_{\max} = 4$ for our implementation.

The two optimized routines from [6] are at heart efficient rewritings of the original code of [148], the only difference between the two being how the reference set gets updated at each iteration. Together with our implementation, they were the only ones that were able to converge on 7 out of the 8 test cases. Even so, our code is the most robust one in terms of execution time. We also tested the corresponding routine from Scilab 5.5.1 (written in Fortran), but did not achieve convergence for any of the test cases.

The effects of starting with a better reference (via *scaling* or *approximate Fekete points*) and *parallelization* of the extrema search are showcased in Table 3.13. As we already emphasized, the greatest gain is for the last two examples, where the filter degrees are larger. In the case of Example 3.21, for which *only* our routine converged, enabling parallelization on the quad core machine we used for testing, resulted in our code running 3.3 times faster than the purely sequential version.

3.10 Conclusion

In this chapter we have presented several ideas that aid in the development of weighted minimax polynomial approximations, with a large focus on optimal linear-phase FIR filters. These contributions amount to the

Table 3.13: Timings (real time) showing the effect of running our code with different options. As for Table 3.12, the numerical values represent the averages in seconds over 50 executions. For the first seven lines, the double-precision version of our routine was used, while for the last one `long double` 80-bit operations were carried out. In all cases, our code was compiled using `g++5.2.0` with `-O3 -DNDEBUG` level optimizations.

Example (degree)	Uniform (sequential)	Uniform (parallel)	Scaling (sequential)	Scaling (parallel)	AFP (sequential)	AFP (parallel)
3.18 ($n = 50$)	0.0034	0.0031	0.0021	0.0018	0.0021	0.0011
3.18 ($n = 80$)	0.0045	0.0035	0.0029	0.0012	0.0029	0.0015
3.18 ($n = 100$)	0.0073	0.0059	0.0099	0.0041	0.0035	0.0023
3.19 ($n = 50$)	0.0022	0.0019	0.0024	0.0015	0.0026	0.0014
3.19 ($n = 80$)	0.0069	0.0064	0.0058	0.0041	0.0029	0.0028
3.19 ($n = 100$)	0.0301	0.0151	0.0123	0.0066	0.0113	0.0071
3.20 ($n = 520$)	0.2375	0.1613	0.1632	0.0725	0.0731	0.0693
3.21 ($n = 53248$)	NC	NC	537.8	162.6	NC	NC

following:

- introduce two new initialization strategies (Sections 3.6.2 and 3.6.3); they ensure the convergence of our routine in cases where all other implementations fail to work, while frequently incurring significant speedups as well (consequence of a small number of iterations required for convergence);
- present pertinent analysis tools which help diagnose when the exchange algorithm is prone to numerical problems in practice (Section 3.7.2);
- introduce a variation of a well-established root-finding approach [34, 205] which allows one to obtain weighted minimax polynomial approximations in an efficient way (Section 3.8.2).
- equally noteworthy is the fact that this study has helped us develop an efficient and highly parallel software library. As we saw throughout all of the examples we considered up to this point, our routine outperforms other existing codes in terms of scalability and numerical accuracy. Another major element differentiating our approach from standard routines like MATLAB's `firpm` is the fact that we do not discretize the approximation domain, meaning in the end we generally get more accurate results, all the while not incurring prohibitive computational costs.

Future work

In terms of continuing such an implementation-centric work, there are several possible avenues of exploration that are worth considering. Here are some of them.

- for the moment, our code can only handle functions using bases of the form $(w(x)T_0(x), \ldots, w(x)T_n(x))$, because of their usefulness to filter design applications. More general bases satisfying the Haar condition should be allowed as well. Monomial-like bases are important in the design of mathematical function implementations for `libms`;
- provide an implementation of the first Remez algorithm for systems not satisfying the Haar condition. Starting with a good initial reference in such a setting is maybe more important than for the second Remez algorithm, since at each iteration, the discrete problem to solve grows in size. Also, degeneracy problems of the type discussed in Remark 3.17, need to be properly taken into account;
- univariate minimax approximations can be used to construct so-called best product Chebyshev approximations to continuous bivariate functions on a rectangle [226] by a tensor product-like approach. Since they can have very good uniform norm properties, it may be interesting to extend our code base to compute such results;

- see if similar ideas to the ones presented here can be applied to other digital signal processing tasks, like those requiring FIR filters with complex coefficients [115, 116].

4

Machine-efficient approximations and filter design

Imaginaţia e şi ea o sursă de informare.
Grigore C. Moisil

According to [100], embedded processors are one of the main economical juggernauts in todays computing industry, mainly because they offer the widest range of alternatives in terms of processing power and cost. Since price is the principal constraint when using such platforms, the goal is to choose the device that meets some given performance requirements at the smallest possible cost. In particular, most digital filtering operations are done in this embedded setting, so employing efficient designs is very important in practice.

This means that we are generally forced to work with strict format constraints on the coefficients of the polynomials or rational functions we are using. Such requirements increase the complexity of determining optimal designs for the problem at hand. In this chapter, we introduce a fast and efficient heuristic for suboptimal finite wordlength coefficient quantization, based on the computation of good nodes for polynomial interpolation, as discussed in Section 3.6, and Euclidean lattice basis reduction. Most of the time, it returns finite precision linear phase FIR filters which behave very well with respect to the weighted minimax norm.

The exposition is an extension of [40, 41] and represents a joint work with N. Brisebarre and G. Hanrot.

4.1 Introduction

The efficient design of FIR filters has been an active research topic in digital signal processing for several decades now. A lot of the focus has been directed to how these objects are implemented in hardware, leading to the consideration of different structures and computational algorithms for performing the dot product filtering operation (2.8) in the time domain. Some notable examples (see [158, Ch. 6.5–6.6] for details) are:

- **direct form**: this is the simplest alternative. It consists of using the filter coefficients directly inside the difference equation (2.8), which for an N-tap FIR filter becomes

$$y[n] = \sum_{k=0}^{N-1} h[k]x[n-k] \tag{4.1}$$

  and corresponds to the transfer function $H(z) = \sum_{k=0}^{N-1} h[k]z^{-k}$.

- **cascade form**: it is obtained by factoring $H(z)$ as

$$H(z) = \sum_{k=0}^{N-1} h[k]z^{-k} = \prod_{k=1}^{\lfloor \frac{N+1}{2} \rfloor} \left(b_{0k} + b_{1k}z^{-1} + b_{2k}z^{-2}\right)$$

  and equivalently evaluating $y[n]$ inside (4.1) using the b_{0k}, b_{1k} and b_{2k} coefficients.

- **lattice filters**: also a type of cascade-like structure that uses a different set of coefficients than the filter taps to evaluate $y[n]$ (see [158, Ch. 6.6]). Lattice filters have no direct relation to Euclidean lattices.

Having access to different realizations is important since, when implemented, some of them might be less sensitive than others to finite precision numerical effects. The use of direct forms is nonetheless preferred in many cases, which is why we will focus on them here. The major reason for working with direct forms is due to the fact that most DSP chips come equipped with efficient structures to perform long sequences of multiply-accumulate operations, just like in (4.1).

In general, an initial design procedure like the Remez exchange algorithm is applied in order to retrieve a set of "infinite precision" coefficients and filter taps, which are frequently computed as 64-bit floating-point values. When, for example, FIR filters are used inside specialized DSP processors or field-programmable gate arrays (FPGAs), the coefficient constraints are very strict, usually requiring the use of fixed-point values of much smaller bit size. Even very recent FPGAs that come with floating-point DSP blocks [128] only offer support for 32-bit floating-point arithmetic. Simply rounding the initial 64-bit precision coefficients to their nearest representations in these imposed formats can give rise to suboptimal filters. In certain cases, the quality of these naive results can be significantly improved, especially when fixed-point values are used.

Two types of quantizations are commonly discussed in the literature:

(i) the filter taps are fixed-point values of the form

$$\frac{m_i}{s}, m_i \in \mathbb{Z} \cap [m_-, m_+], m_-, m_+ \in \mathbb{Z}, \tag{4.2}$$

where the scaling factor s is is usually a power of 2. This is also known as *uniform* coefficient quantization. Such representations are useful when working with dedicated DSP processors or equivalent blocks inside FPGAs.

(ii) sum of power of two terms in binary or CSD formats (see the end of Section 2.3.2 for a definition of CSD encodings). They are useful for multiplierless filtering circuits, where the interest is for the overall number of power of two terms in the coefficients to be as small as possible (*i.e., non-uniform* coefficient quantization).

4.1.1 State of the art

The engineering literature on the coefficient quantization problem in digital filter design is much too large to do it justice in just a few paragraphs. We will nonetheless try to give a comprehensive picture of how it evolved over the past decades.

It could be said that everything started in the 1960s and was due to what is now seen as an unavoidable switch from analog to digital circuits. Kaiser [112], in 1965, is the first to study the effect of coefficient errors on filter performance. Several years later, Chan and Rabiner [51] looked at the statistical properties of the error in the filter's frequency response when rounding of the coefficients is used. A more recent reference on the statistical effects of finite precision computation, including digital filter design, is [229].

For both uniform and non-uniform coefficient quantization problems, finding a set of *optimal* quantized coefficients is a computationally hard problem, with exact solvers making heavy use of (mixed) integer linear programming (MILP) techniques. A summary of early references (circa 1980) on the topic can be consulted for instance in [119, 120, 135]. In the case of optimal type **(i)** solutions, it is probably Kodek [120–123] that, over the years, has worked the most on studying and improving the behavior of specialized branch-and-bound MILP algorithms. These efforts, coupled with the computational capabilities of current processors, make it that for problems where $N \leqslant 60$ and coefficients stored using up to 12 bits, optimal solutions can usually be obtained within seconds [123, Table I], while for larger parameter values, the runtime quickly escalates.

The sum of power of two setting has received even more attention from the engineering community. A probable reason is that, early on, the accuracy gains from doing optimal uniform coefficient quantization as opposed to using more naive methods seemed experimentally not to over-weigh the complexity costs [119], whereas in the non-uniform case, significant improvements were clearly achievable [136]. Optimal approaches

based on MILP have been discussed for instance in [8,96,134,136–138,190], with a name appearing in many of the references being that of Lim. Again, their use is limited to relatively small design parameters.

To successfully address larger word-length (and degree) problems, fast heuristics have also been proposed. For example, in the case of uniform fixed-point coefficients, an early interesting reference is [135], which describes a branch-and-bound tree search strategy combined with a L_2-type coefficient quantization criterion. The approach described in [157], which employs error spectrum shaping techniques, is routinely used inside MATLAB for FIR coefficient quantization. More recently, the telescoping rounding approach described in [124] also seems to produce good results. In the context of non-uniform coefficient problems, there are again a lot of heuristics to choose from. For recent accounts and methods, we refer the reader to [8,68,192,225,234] and the references therein.

The rest of this chapter will be focused on introducing a novel heuristic for designing suboptimal FIR filters with uniform quantized coefficients. It is based on previous work for doing machine-efficient polynomial approximation of functions [38] combined with techniques that yield families of very good interpolation nodes [31,32,47,73,195,196]. We also make available an open source C++ implementation of our approach[1]. Since it produces very good quantizations with respect to the L_∞ error measure, our code should also be able to help accelerate exact solvers based on MILP in certain cases.

4.2 The finite wordlength design problem

A general formalization of the question we are asking in this chapter is the following:

Problem 4.1 (Machine-coefficient optimal approximation)**.** *Given an approximation domain $X \subset \mathbb{R}$, a target real-valued function f to approximate over X, a set of basis functions $\varphi_0, \ldots, \varphi_n$ and a set of finite-precision formats $(\mathcal{F}_i)_{0 \leqslant i \leqslant n}$, find $a_i \in \mathcal{F}_i, 0 \leqslant i \leqslant n$ such that*

$$\sup_{x \in X} \left| \sum_{i=0}^{n} a_i \varphi_i(x) - f(x) \right|$$

is minimal*. Let us call this value δ_{opt}.*

It follows from the fact that the number of candidate approximations is always finite that an instance of Problem 4.1 always has a solution. On the other hand, we cannot guarantee anything regarding uniqueness.

In function approximation problems for `libm` design, X is generally a closed interval, $f \in \mathcal{C}(X)$, the basis functions correspond to (appropriately scaled) monomials and the $\mathcal{F}_i$ formats are usually the single or double-precision floating-point formats or floating-point expansion (that is, sums of two or more floating-point values, like for instance so-called double-`double` and triple-`double` values). For a detailed discussion on the these types of problems and possible solutions, see for instance [58, Ch. 2.3].

In the filter design applications we are focusing on here, X is generally a finite union of closed intervals and the basis functions are scaled versions of the ones introduced by equations (2.11)–(2.14), depending on the target filter type. Indeed, if the filter tap format is the one given in (4.2), then we can force the a_i's to be in $[m_-, m_+] \cap \mathbb{Z}$, whereas the basis functions take the values from Table 4.1. *We recall that our usual $x = \cos(\omega)$ change of variable is in effect.*

While there are a lot of similarities between these two settings, the target applications in the filter use case can sometimes introduce issues which were not present in the function approximation space.

Table 4.2 gives 5 design specifications used throughout the chapter as working examples for our method.

4.2.1 Exact approaches

There are certain advantages and limitations of current methods for finding optimal finite wordlength solutions which are worth discussing here. Looking at Problem 4.1 geometrically is especially interesting.

[1] see https://github.com/sfilip/fquantizer

Table 4.1: Scaled basis functions for finite word length FIR design.

FIR Type	$\varphi_0(x), \varphi_1(x), \varphi_2(x), \ldots, \varphi_n(x)$
I	$\frac{W(\omega)}{s}, \frac{2W(\omega)\cos(\omega)}{s}, \frac{2W(\omega)\cos(2\omega)}{s}, \ldots, \frac{2W(\omega)\cos(n\omega)}{s}$
II	$\frac{2W(\omega)\cos\left(\frac{\omega}{2}\right)}{s}, \frac{2W(\omega)\cos\left(\frac{3\omega}{2}\right)}{s}, \frac{2W(\omega)\cos\left(\frac{5\omega}{2}\right)}{s}, \ldots, \frac{2W(\omega)\cos\left(\frac{(2n+1)\omega}{2}\right)}{s}$
III	$\frac{2W(\omega)\sin(\omega)}{s}, \frac{2W(\omega)\sin(2\omega)}{s}, \frac{2W(\omega)\sin(3\omega)}{s}, \ldots, \frac{2W(\omega)\sin((n+1)\omega)}{s}$
IV	$\frac{2W(\omega)\sin\left(\frac{\omega}{2}\right)}{s}, \frac{2W(\omega)\sin\left(\frac{3\omega}{2}\right)}{s}, \frac{2W(\omega)\sin\left(\frac{5\omega}{2}\right)}{s}, \ldots, \frac{2W(\omega)\sin\left(\frac{(2n+1)\omega}{2}\right)}{s}$

Table 4.2: Type I filter specifications considered in [124].

Filter	Bands	$D(\omega)$	$W(\omega)$
A	$[0, 0.4\pi]$	1	1
	$[0.5\pi, \pi]$	0	1
B	$[0, 0.4\pi]$	1	1
	$[0.5\pi, \pi]$	0	10
C	$[0, 0.24\pi]$	1	1
	$[0.4\pi, 0.68\pi]$	0	1
	$[0.84\pi, \pi]$	1	1
D	$[0, 0.24\pi]$	1	1
	$[0.4\pi, 0.68\pi]$	0	10
	$[0.84\pi, \pi]$	1	1
E	$[0.02\pi, 0.42\pi]$	1	1
	$[0.52\pi, 0.98\pi]$	0	1

Polytope projections

Consider a set of $m+1 \geqslant n+1$ distinct points $x_0, \ldots, x_m \in X$, arbitrarily chosen for the time being, and an error term $\delta \geqslant \delta_X$, where δ_X corresponds to the minimax solution of Problem 4.1 when the a_i are only required to be real values. Then, the following problem, with unknowns $a_0, \ldots, a_n$, has at least one solution in $\mathbb{R}^{n+1}$:

$$\begin{array}{rl} \sum_{i=0}^{n} a_i\varphi_i(x_0) - f(x_0) & \leqslant \delta, \\ f(x_0) - \sum_{i=0}^{n} a_i\varphi_i(x_0) & \leqslant \delta, \\ \vdots & \\ \sum_{i=0}^{n} a_i\varphi_i(x_m) - f(x_m) & \leqslant \delta, \\ f(x_m) - \sum_{i=0}^{n} a_i\varphi_i(x_m) & \leqslant \delta. \end{array} \tag{4.3}$$

As discussed in the previous chapter, for the problems at hand, the basis functions we are using satisfy the Haar condition, thus the system formed by the $\varphi_i(x_j)$'s has rank $n+1$, meaning (4.3) defines a *bounded* polytope $\mathcal{P}$ in $\mathbb{R}^{n+1}$ (see for instance [42, Appx. 2] for a proof idea). We can thus orthogonally project $\mathcal{P}$ on each of the axes spanned by the a_i's using the simplex algorithm (*i.e.,* minimize and maximize each a_i under the linear constraints of (4.3)). If moreover, $\delta \geqslant \delta_{\text{opt}}$, then we have the guarantee that each bounding interval $[a_i^-, a_i^+]$ defined by the projections of $\mathcal{P}$ contains at least one element from $\mathcal{F}_i$ and that by testing all possible combinations of valid coefficients, at least one of them will generate the solution to Problem 4.1.

Example 4.2. *Consider a degree* $n = 22$ *type I polynomial filter adhering to specification A from Table 4.2. The coefficients of its associated transfer function are* 8*-bit fixed-point values with* 7 *bits used for the fractional part. Using the language of Problem 4.1, we are searching for integer values* $a_i \in [-2^7, 2^7]$ *such that*

$$\sup_{x\in X}\left|\sum_{i=0}^{22} a_i\varphi_i(x) - f(x)\right|$$

is minimal, where, based on Tables 3.1 and 4.1, we have $x = \cos(\omega), X = [-1, 0] \cup [\cos(0.4\pi), 1], \varphi_0(x) = \frac{1}{2^7}, \varphi_i(x) = \frac{T_i(x)}{2^6}, 1 \leqslant i \leqslant 22, f(x) = W(\omega)D(\omega) = D(\omega)$.

Let us start by taking a pessimistic $\delta \simeq 0.03701$*, the error of the naive rounding of the minimax solution with real valued coefficients (*i.e., *each coefficient of the minimax solution is rounded to the imposed format using the rule in expression (2.16)). We use as discretization a* $2 \cdot (22 + 2) = 48$*-point weakly admissible mesh consisting of* 24 *appropriately scaled Chebyshev nodes on each of the two intervals making up* X*. We obtain the projection bounds*

$$\begin{array}{llll}
[a_0^-, a_0^+] = [52, 63], & [a_1^-, a_1^+] = [38, 43], & [a_2^-, a_2^+] = [2, 11], & [a_3^-, a_3^+] = [-15, -9],\\
[a_4^-, a_4^+] = [-10, -2], & [a_5^-, a_5^+] = [2, 9], & [a_6^-, a_6^+] = [2, 8], & [a_7^-, a_7^+] = [-6, 2],\\
[a_8^-, a_8^+] = [-6, -2], & [a_9^-, a_9^+] = [-4, 5], & [a_{10}^-, a_{10}^+] = [1, 5], & [a_{11}^-, a_{11}^+] = [-3, 4],\\
[a_{12}^-, a_{12}^+] = [-4, 0], & [a_{13}^-, a_{13}^+] = [-4, 3], & [a_{14}^-, a_{14}^+] = [-1, 4], & [a_{15}^-, a_{15}^+] = [-2, 4],\\
[a_{16}^-, a_{16}^+] = [-4, 2], & [a_{17}^-, a_{17}^+] = [-3, 1], & [a_{18}^-, a_{18}^+] = [-3, 3], & [a_{19}^-, a_{19}^+] = [-2, 3],\\
[a_{20}^-, a_{20}^+] = [-3, 3], & [a_{21}^-, a_{21}^+] = [-3, 2], & [a_{22}^-, a_{22}^+] = [-3, 3], &
\end{array} \tag{4.4}$$

which require testing $1.600601657 \cdot 10^{19}$ *combinations to retrieve a best set of coefficients.*

If we take $\delta \simeq 0.02963$*, a value which is very close to* δ_{opt}*, and the same* 48*-point WAM discretization, we get the bounds*

$$\begin{array}{llll}
[a_0^-, a_0^+] = [53, 62], & [a_1^-, a_1^+] = [38, 42], & [a_2^-, a_2^+] = [3, 10], & [a_3^-, a_3^+] = [-14, -10],\\
[a_4^-, a_4^+] = [-9, -3], & [a_5^-, a_5^+] = [3, 8], & [a_6^-, a_6^+] = [3, 7], & [a_7^-, a_7^+] = [-5, 1],\\
[a_8^-, a_8^+] = [-6, -2], & [a_9^-, a_9^+] = [-3, 4], & [a_{10}^-, a_{10}^+] = [2, 5], & [a_{11}^-, a_{11}^+] = [-2, 3],\\
[a_{12}^-, a_{12}^+] = [-4, 0], & [a_{13}^-, a_{13}^+] = [-4, 2], & [a_{14}^-, a_{14}^+] = [-1, 3], & [a_{15}^-, a_{15}^+] = [-1, 3],\\
[a_{16}^-, a_{16}^+] = [-3, 1], & [a_{17}^-, a_{17}^+] = [-3, 1], & [a_{18}^-, a_{18}^+] = [-2, 3], & [a_{19}^-, a_{19}^+] = [-1, 2],\\
[a_{20}^-, a_{20}^+] = [-2, 2], & [a_{21}^-, a_{21}^+] = [-2, 1], & [a_{22}^-, a_{22}^+] = [-2, 2], &
\end{array} \tag{4.5}$$

decreasing the number of test combinations to $1.48176 \cdot 10^{17}$*.*

Now, if we double the discretization size $m + 1$ *to* 96 *(*i.e., *a* 48*-point Chebyshev grid on each subinterval of* X*), but still take* $\delta \simeq 0.02963$*, the projection intervals will be*

$$\begin{array}{llll}
[a_0^-, a_0^+] = [53, 62], & [a_1^-, a_1^+] = [38, 42], & [a_2^-, a_2^+] = [3, 10], & [a_3^-, a_3^+] = [-14, -10],\\
[a_4^-, a_4^+] = [-9, -3], & [a_5^-, a_5^+] = [3, 8], & [a_6^-, a_6^+] = [3, 7], & [a_7^-, a_7^+] = [-5, 1],\\
[a_8^-, a_8^+] = [-5, -3], & [a_9^-, a_9^+] = [-2, 3], & [a_{10}^-, a_{10}^+] = [2, 4], & [a_{11}^-, a_{11}^+] = [-2, 3],\\
[a_{12}^-, a_{12}^+] = [-4, -1], & [a_{13}^-, a_{13}^+] = [-3, 1], & [a_{14}^-, a_{14}^+] = [0, 3], & [a_{15}^-, a_{15}^+] = [-1, 3],\\
[a_{16}^-, a_{16}^+] = [-3, 1], & [a_{17}^-, a_{17}^+] = [-3, 1], & [a_{18}^-, a_{18}^+] = [-2, 2], & [a_{19}^-, a_{19}^+] = [-1, 2],\\
[a_{20}^-, a_{20}^+] = [-2, 2], & [a_{21}^-, a_{21}^+] = [-2, 1], & [a_{22}^-, a_{22}^+] = [-2, 2], &
\end{array} \tag{4.6}$$

which would still require testing $1.90512 \cdot 10^{16}$ *combinations to retrieve a best set of coeffcient values.*

For reference, an optimal quantization in this case has coefficients

$$\begin{array}{llll}
a_0 = 58, & a_1 = 40, & a_2 = 7, & a_3 = -12,\\
a_4 = -6, & a_5 = 5, & a_6 = 5, & a_7 = -2,\\
a_8 = -4, & a_9 = 0, & a_{10} = 3, & a_{11} = 1,\\
a_{12} = -2, & a_{13} = -1, & a_{14} = 1, & a_{15} = 1,\\
a_{16} = 0, & a_{17} = -1, & a_{18} = 0, & a_{19} = 1,\\
a_{20} = 0, & a_{21} = 0, & a_{22} = 0. &
\end{array} \tag{4.7}$$

In all of (4.4), (4.5) and (4.6), the bounds do not vary very much. The position of the optimal a_i's inside these intervals is interesting though; they appear to be approximately *centered* in all cases. We noticed this behavior for some other examples as well, although we cannot confirm this to be generally the case.

The fact that the bounds we obtain are rather large for all coefficients contrasts somewhat the situation in Chevillard's thesis [58, Ch. 2.3.2]. In that context, projection produces tighter bounds for many examples of floating-point quantization on an interval. A reason might be the much smaller errors $\delta_{\text{opt}} < 10^{-9}$ they target, which are normally much more sensitive to changes in the coefficient values. Another reason might be the working basis; we consider Chebyshev polynomials on essentially a large subset of $[-1, 1]$, meaning all the basis polynomials will have more or less the same range. The function approximation context of [58, Ch. 2.3.2] however, works on very small intervals using the basis of monomials; the range of each basis element will thus be very small and the ranges of any two distinct monomials might differ quite a lot from one another, influencing the tightness of the projection bounds. For the moment these are just speculations.

Using projections in such a way, although capable of providing exact solutions, seems like a viable option only for relatively small degrees (say $n < 15$), considering the $2n + 2$ linear programs that are to be solved each time a new δ and/or discretization is chosen. The denser the discretization, the more accurate the projections will be (a smaller number of candidate solutions to verify), but it will also impact the execution time for the linear programming solver.

Our computations were performed by coding an experimental C/C++ implementation for polytope projection operations. It relies on the `GLPK`[2] C library interface for the exact solution of linear programming problems using `GMP`[3] multiple precision rationals. We are extremely grateful for all the help that Serge Torres has provided, by sharing his experience in coding the similar projection routine of [58, Ch. 2.3.2].

Enumeration

Continuing with the geometric tone of our presentation, another idea that will theoretically lead to all the optimal solutions of Problem 4.1 is that of *enumerating* all the quantization points $(a_0, \ldots, a_n)$ located inside the polytope defined by (4.3). We will not go into the details of the approach; this is explored in [42] and [58, Ch. 2.3.3] for function approximation. By carefully constructing $\mathcal{P}$, the goal is to only deal with a small number of candidate solutions (less than if we use projection). Again, this seems to be feasible only for very small degrees ($n < 15$). Example 4.2 for instance, was much too time consuming. For testing the approach, we wrote a prototype C++ implementation with the help of the ISL library [218], which features state of the art algorithms for the enumeration of integer points inside polyhedra. On the specifications from Table 4.2, we only managed to go up to degree $n = 10$ filters. The running time generally ranged from several minutes for $n = 5$ up to a dozen hours for some $n = 10$ instances.

Mixed Integer Linear Programming

A more fruitful approach is to cast the design problem as an MILP and apply branch-and-bound strategies. Instead of attacking Problem 4.1 directly, the goal is to solve

$$\begin{aligned} \text{minimize} \quad & \underline{\delta} \\ \text{subject to} \quad & \sum_{k=0}^{n} a_k \varphi_k(x) - f(x) \leqslant \delta, x \in X_d, \\ & f(x) - \sum_{k=0}^{n} a_k \varphi_k(x) \leqslant \delta, x \in X_d, \\ & \underline{\delta} > 0, \quad a_k \in \mathcal{F}_k, k = 0, \ldots, n. \end{aligned} \tag{4.8}$$

[2] https://www.gnu.org/software/glpk/
[3] https://gmplib.org/

The set $X_d \subset X$ should be dense enough such that the solution of (4.8) is not far from the optimal one over X. For instance, a number of $16n$ points (the value used by default for most discretized versions of the Remez exchange algorithm found in many Signal Processing packages) is in general sufficient for practical purposes and will actually produce a solution over X as well. We found that in some cases, a smaller number of points inside X_d continued to produce an optimal result over X, while in others it did not.

MILPs of the form (4.8) have been studied quite a lot in the literature. Implementation-wise, it is easy to code up such an MILP instance using current mathematical optimization suites. In our tests we use two particular solvers. One relies on the commercial Gurobi library [95], which performs computations in double precision arithmetic. It is capable of addressing instances with $n < 30$ and coefficients with word-lengths of up to 10 bits in under a minute in many cases. For slightly larger degrees and/or wordlengths on the other hand, it quickly becomes much slower.

The other one is an exact solver based on a multiple precision implementation of the simplex algorithm [65], a fork of the SCIP optimization suite [4]. It is therefore also very slow. An exact solver is nevertheless useful if one wants to *validate* the quality of the result. Although most practitioners would probably see this as overkill, it's impressive in itself that with current tools, we are able to offer such guarantees.

By a validated result, we mean that the exact solution for an instance of (4.8), provides a *true* lower bound for δ_{opt}. In the case of Example 4.2, using an X_d with $400 > 22 \cdot 16$ points gives us

$$\underline{\delta} = \frac{5675024184985759775800304352648757143647490133510571}{191561942608236107294793378393788647952342390272950272} \simeq \mathbf{2.962}50085362298 \cdot 10^{-2},$$

taking around 2891 seconds on a machine equipped with an Intel i7-3687U CPU, as opposed to the 5.83 seconds required by the double precision solver.

For a rigorous upper bound, we can for instance obtain one by taking the quantized polynomial given by the MILP solver (whose coefficients are given by (4.7)) and applying the `supnorm` command from Sollya, the algorithm of which is described in [59]. We then obtain the validated upper bound

$$\overline{\delta} = \mathbf{2.962}75069697236281770562382461047435407539070118 93 \cdot 10^{-2},$$

meaning the exact error δ_{opt} of the minimax solution is *provably* inside the interval $[\underline{\delta}, \overline{\delta}]$.

4.2.2 A heuristic approach

In the context of finite precision function approximation problems for `libm` design, Brisebarre and Chevillard [38] advocate the idea of using a well-chosen discretized version of Problem 4.1 (the `fpminimax` algorithm).

In our case, the basis functions $(\varphi_i)_{0 \leqslant i \leqslant n}$ are appropriately scaled as discussed in the beginning of this section. We then search for $a_i \in [m_-, m_+] \cap \mathbb{Z}$ such that δ_{opt} is minimal. Ideally, if the format constraints given by m_-, m_+ and s are not too strict, then we expect the integral solution $(a_0, \ldots, a_n)$ to match as close as possible the infinite precision solution $p^\star$, obtained for instance using the Remez exchange algorithm. If we take $m + 1 \geqslant n + 1$ distinct points $(z_i)_{0 \leqslant i \leqslant m}$ from X, then it should hopefully follow that

$$\sum_{i=0}^{n} a_i \underbrace{\begin{bmatrix} \varphi_i(z_0) \\ \varphi_i(z_1) \\ \vdots \\ \varphi_i(z_m) \end{bmatrix}}_{\mathbf{b}_i} \simeq \underbrace{\begin{bmatrix} p^\star(z_0) \\ p^\star(z_1) \\ \vdots \\ p^\star(z_m) \end{bmatrix}}_{\mathbf{t}} \tag{4.9}$$

For the vector $\mathbf{t} \in \mathbb{R}^{m+1}$, we want to find the integer combination of $m+1$-dimensional basis vectors $\mathbf{b}_0, \ldots, \mathbf{b}_n$ *closest* to it with respect to a certain norm (as will be explained, we use the Euclidean norm). If we do that, the result will probably give us a good estimation of the actual solution of Problem 4.1.

The integer combinations of vectors on the left hand side of (4.9) are called *lattice vectors*, whereas the problem of determining a lattice vector which is closest to a targer vector $\mathbf{t}$ is called the *closest vector problem* or CVP. Over the course of the next section we will review some basic facts regarding lattices. We

will also look at some important problems and algorithmic approaches used to solve them. They will be essential in describing our method in Section 4.4.

Remark 4.3. It is not mandatory to compute $p^\star$ if we replace the CVP instance in (4.9) with

$$\sum_{i=0}^{n} a_i \underbrace{\begin{bmatrix} \varphi_i(z_0) \\ \varphi_i(z_1) \\ \vdots \\ \varphi_i(z_m) \end{bmatrix}}_{\mathbf{b}_i} \simeq \underbrace{\begin{bmatrix} f(z_0) \\ f(z_1) \\ \vdots \\ f(z_m) \end{bmatrix}}_{\mathbf{t}}, \tag{4.10}$$

after having chosen the nodes $z_0, \ldots, z_m$. It makes sense to do so, since the computed approximant will in fact need to be close to f and not necessarily $p^\star$. Both (4.9) and (4.10) tend to produce very good results, but neither seems to be globally best in practice. This is why we will continue to use (4.9) in the sequel, although everything we say applies to (4.10) also.

4.3 Euclidean lattices

The study of lattices has sparked the interest of some highly influential mathematicians such as Lagrange [127], Gauss [86] (in the study of binary quadratic forms) and Hermite [101]. They are the core of an important branch of Number Theory called *Geometry of Numbers*, which was introduced as a mathematical discipline by Minkowski [152] in the second half of the 19th century.

Today, Euclidean lattices are also seen as powerful geometric objects with many applications in Mathematics and Computer Science. Some notable examples include diophantine approximation, error correcting codes in Information Theory, solutions of integer programming problems and cryptographic security.

4.3.1 Some basic results

We will be working in the $\mathbb{R}^n$ vector space and, for the most part, consider the ℓ_2 Euclidean norm

$$\|\mathbf{x}\|_2 = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}, \mathbf{x} \in \mathbb{R}^n.$$

At the bare minimum, a lattice is just a *discrete subgroup of* $(\mathbb{R}^n, +)$.

Given a set of vectors $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\} \subset \mathbb{R}^n, d \leqslant n$, it will be useful to consider the set

$$L = \mathcal{L}(\{\mathbf{b}_i : 1 \leqslant i \leqslant d\}) = \left\{ \sum_{i=1}^{d} x_i \mathbf{b}_i : x_i \in \mathbb{Z} \right\},$$

which is a subgroup of $\mathbb{R}^n$, but not necessarily a lattice. For L to be a lattice it will suffice that either the $\mathbf{b}_i$ vectors belong to $\mathbb{Q}^n$ or that they are linearly independent $\mathbb{R}^n$ vectors (see [156, Ch. 2, Thm. 1]).

Definition 4.4 (Lattice basis)**.** If $L = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ is a $\mathbb{R}^n$ lattice, then $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ is called a *generating set* of L. If all the $\mathbf{b}_i$'s are linearly independent, then they are called a *basis* of L, with $d \leqslant n$ being the *dimension* (or *rank*) of L. If $d = n$, L is called *full-rank*.

Equivalently, if we take $\mathbf{B}$ as the $n \times d$ matrix whose columns are the basis vectors $\mathbf{b}_1, \ldots, \mathbf{b}_d$, then the lattice they generate can also be defined as

$$L = \mathcal{L}(\mathbf{B}) = \left\{ \mathbf{B}\mathbf{x} : \mathbf{x} \in \mathbb{Z}^d \right\}.$$

A very natural example is the integer lattice $\mathbb{Z}^n$, which has as possible basis, the set of n-dimensional unit vectors $\{\mathbf{e}_1, \ldots, \mathbf{e}_n\}$. Figure 4.1, shows this for $\mathbb{Z}^2$. Figure 4.2 also considers $\mathbb{Z}^2$, but uses a different basis.

It is not hard to check that, when dealing with any rank $d \geqslant 2$ lattice, it will have infinitely many bases.

If we already know a basis $\mathbf{B}$ of L, we can algebraically tell if another matrix $\mathbf{B}'$ is also a basis. It is the analogue of the change of basis operation for finite dimensional vector spaces and relies on the use of *unimodular* matrices, that is elements $\mathbf{U} \in \mathbb{Z}^{d \times d}$ with $\det \mathbf{U} = \pm 1$.

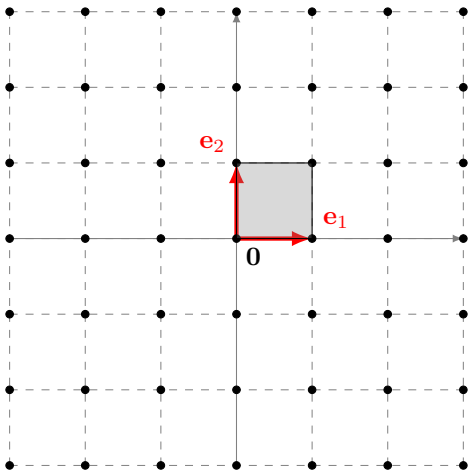

Figure 4.1: The $\mathbb{Z}^2$ lattice with unit basis vectors $\mathbf{e}_1 = [1\ 0]^T$ and $\mathbf{e}_2 = [0\ 1]^T$ and its associated fundamental parallelepiped.

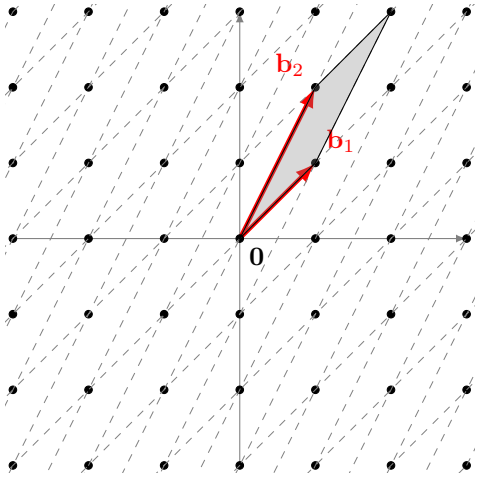

Figure 4.2: The $\mathbb{Z}^2$ lattice, now with basis vectors $\mathbf{b}_1 = [1\ 1]^T$ and $\mathbf{b}_2 = [1\ 2]^T$ and its fundamental parallelepiped.

Lemma 4.5. *Two bases $\mathbf{B}, \mathbf{B}' \in \mathbb{R}^{n\times d}$ are equivalent if and only if $\mathbf{B}' = \mathbf{BU}$, for some unimodular matrix $\mathbf{U} \in \mathbb{Z}^{d\times d}$.*

Proof. Assume that $\mathcal{L}(\mathbf{B}) = \mathcal{L}(\mathbf{B}')$. This implies that each column of $\mathbf{B}'$ can be written as an integer combination of columns of $\mathbf{B}$, meaning there exists $\mathbf{U} \in \mathbb{Z}^{d\times d}$ such that $\mathbf{B}' = \mathbf{BU}$. Analogously, there exists $\mathbf{V} \in \mathbb{Z}^{d\times d}$ for which $\mathbf{B} = \mathbf{B}'\mathbf{V}$. We have $\mathbf{B}' = \mathbf{BU} = \mathbf{B}'\mathbf{VU}$ and consequently $\mathbf{B}'^T\mathbf{B}' = (\mathbf{VU})^T\mathbf{B}'^T\mathbf{B}'(\mathbf{VU})$. By taking determinants, we get $\det(\mathbf{B}'^T\mathbf{B}') = (\det(\mathbf{VU}))^2 \det(\mathbf{B}'^T\mathbf{B}')$. We also have $\det(\mathbf{B}'^T\mathbf{B}') \neq 0$. If this were not true, then there exists $\mathbf{x} \in \mathbb{R}^d \setminus \{\mathbf{0}\}$ such that $\mathbf{B}'^T\mathbf{B}'\mathbf{x} = \mathbf{0}$, meaning $\|\mathbf{B}'\mathbf{x}\|_2 = 0$ and $\mathbf{B}'\mathbf{x} = \mathbf{0}$. Since the columns of $\mathbf{B}'$ are linearly independent, $\mathbf{x} = \mathbf{0}$, resulting in a contradiction. It then follows that $\det(\mathbf{V}) \det(\mathbf{U}) = \pm 1$. Since both $\mathbf{U}, \mathbf{V}$ are integer matrices, $\det(\mathbf{U}) = \pm 1$.

In the other direction, we have $\mathbf{B}' = \mathbf{BU}$ for some unimodular matrix $\mathbf{U}$. This means that each column of $\mathbf{B}'$ is an integer combination of columns of $\mathbf{B}$ (*i.e.,* $\mathcal{L}(\mathbf{B}') \subseteq \mathcal{L}(\mathbf{B})$). Also, $\mathbf{B} = \mathbf{B}'\mathbf{U}^{-1}$. The reader can easily check that $\mathbf{U}^{-1}$ is also unimodular (the set of unimodular matrices forms a group under matrix multiplication), leading to $\mathcal{L}(\mathbf{B}) \subseteq \mathcal{L}(\mathbf{B}')$. The conclusion $\mathcal{L}(\mathbf{B}) = \mathcal{L}(\mathbf{B}')$ then follows. □

The analogy with vector spaces should be done with care. For instance, a rank d lattice L can have a proper sublattice $L' \subset L$ of the same rank. An example is $\mathbb{Z}^2$, which has the lattice $2\mathbb{Z} \times \mathbb{Z}$ as a proper subset.

Definition 4.6 (Fundamental parallelepiped)**.** For any given lattice basis $\mathbf{B}$, its *fundamental parallelepiped* is

$$\mathcal{P}(\mathbf{B}) = \left\{ \sum_{i=1}^{d} x_i \mathbf{b}_i : x_i \in \mathbb{R}, 0 \leqslant x_i < 1 \right\}.$$

Examples can be seen in the gray areas of Figures 4.1 and 4.2, where it is visible that a fundamental parallelepiped depends on the basis. A final basic notion we will be requiring is the following

Definition 4.7 (Lattice determinant)**.** Let $L = \mathcal{L}(\mathbf{B})$ be a rank d lattice. The *determinant* (or *volume*) of L, denoted with $\det(L)$, is the d-dimensional volume of $\mathcal{P}(\mathbf{B})$, meaning $\det(L) = \mathrm{vol}(L) = \sqrt{\det(\mathbf{B}^T\mathbf{B})}$. If L is a full rank lattice and $\mathbf{B}$ is a square matrix, then $\det(L) = |\det(\mathbf{B})|$.

This notion is well-defined; if $\mathbf{B}$ and $\mathbf{B}'$ are two bases of the same lattice L, which, by Lemma 4.5, are related by a unimodular matrix $\mathbf{U}$, $\mathbf{B}' = \mathbf{BU}$, then

$$\sqrt{\det(\mathbf{B}'^T\mathbf{B}')} = \sqrt{\det(\mathbf{U}^T\mathbf{B}^T\mathbf{BU})} = \sqrt{\det(\mathbf{B}^T\mathbf{B})}.$$

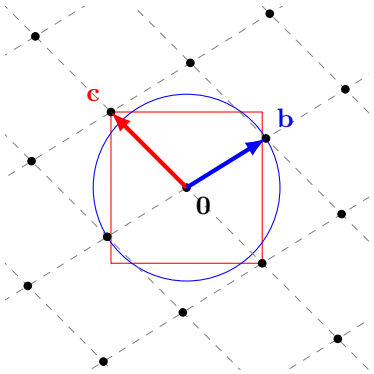

Figure 4.3: A lattice for which the ℓ_2 and ℓ_∞ SVP solutions ($\mathbf{b}$ and $\mathbf{c}$ respectively) are different.

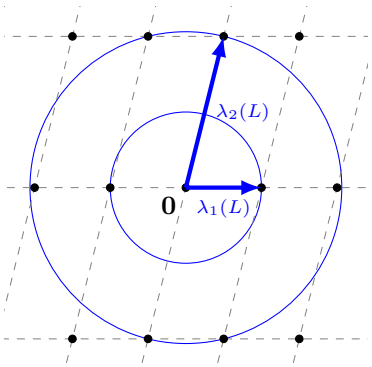

Figure 4.4: Successive minima of a two-dimensional lattice (in the ℓ_2 norm).

The gray areas of Figures 4.1 and 4.2 provide again examples. The value of a lattice determinant can be used to measure the density of a lattice (the so-called *Gaussian heuristic*): the smaller the determinant, the denser the lattice. More precisely, if $\mathcal{B}$ is a ball in the span of L, then the number of lattice points inside $\mathcal{B}$ converges to $\mathrm{vol}(\mathcal{B})/\det(L)$ as the size of $\mathcal{B}$ goes to infinity. $\mathbf{B}^T\mathbf{B}$ is also called the *Gram matrix* of $\mathbf{B}$.

4.3.2 The shortest vector problem

In practice, there are certain bases which are considered to be better than others. For instance, it seems reasonable to prefer a basis which allows us to move easily in the vicinity of a given point (*i.e.,* it favors local searches inside the lattice). In the case of $\mathbb{Z}^2$, the canonical basis from Figure 4.1 is preferable to the one from Figure 4.2.

Continuing with this train of thought regarding locality, a natural question presents itself: what is the shortest nonzero vector inside a lattice? This is known as the *shortest vector problem* (SVP). Since in $\mathbb{R}^n$ all norms are equivalent, it makes sense to ask this question with respect to any norm. Our interest here will be more focused towards the ℓ_2 and ℓ_∞ norms. Note in particular that the solution depends on the norm (Figure 4.3). When not referring to a particular norm in the sequel, we will use the notation $\|\cdot\|$.

Definition 4.8 (Shortest Vector Problem (SVP))**.** Given a basis $\mathbf{B}$ of a rank d lattice $L \subset \mathbb{R}^n$, find a nonzero vector $\mathbf{u} \in L$ such that $\|\mathbf{u}\| = \min_{\mathbf{v}\in L\setminus\{\mathbf{0}\}} \|\mathbf{v}\|$.

Related to the concept of shortest nonzero vector (which we will simply call *shortest vector* in the sequel), Minkowski defined its length as the *first minimum* of the lattice L, denoted with $\lambda_1(L)$. This leads to the notion of successive minima inside a lattice:

Definition 4.9 (Successive minima)**.** If $L \subset \mathbb{R}^n$ is a rank d lattice, for $1 \leqslant i \leqslant d$, the i^{th} minimum of L is

$$\lambda_i(L) = \min_{\substack{\mathbf{u}_1,\ldots,\mathbf{u}_i \in L \\ \text{independent}}} \max_{1\leqslant j\leqslant i} \|\mathbf{u}_j\|.$$

From this definition, it follows that $\lambda_1(L) \leqslant \ldots \leqslant \lambda_d(L)$. Figure 4.4 gives an example. Ideally, *reducing* to a basis whose vectors have lengths corresponding to the successive minima is desirable, but as soon as $n \geqslant 5$ such a basis need not exist (for an example, see for instance [156, Ch. 2]). It is also immediate to note that the shortest vector in a lattice is not unique (*e.g.,* if $\mathbf{u}$ is a SVP solution, then so is $-\mathbf{u}$). Related to the length of a shortest lattice vector, Minkowski proved the following result:

Theorem 4.10 (Minkowski)**.** *Let $L \subset \mathbb{R}^n$ be a n-dimensional lattice and $\mathcal{B} = \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}\| \leqslant 1\}$ the unit ball with respect to the norm $\|\cdot\|$; then there is a vector $\mathbf{v} \in L \setminus \{\mathbf{0}\}$ such that $\|\mathbf{v}\| \leqslant 2\left(\frac{\mathrm{vol}L}{\mathrm{vol}\mathcal{B}}\right)^{1/n}$.*

Unfortunately, the proof of this theorem (see for instance [19, Ch. 7.3]) relies on a kind of pigeonhole principle, making it non-constructive. Computationally, Van Emde Boas showed that the SVP problem

is NP-hard in the ℓ_∞ setting [214]. For the ℓ_2 norm, Ajtai [7] proved the problem to be NP-hard under randomized reductions (see for instance [151, Ch. 4] for further explanations). As a consequence, in applications we might be content with vectors that prove to be *short enough* (see the next section):

Definition 4.11 (Approximate Shortest Vector Problem (aSVP))**.** For a basis $\mathbf{B}$ of a rank d lattice $L \subset \mathbb{R}^n$ and an approximation factor $\gamma \geqslant 1$, find a nonzero vector $\mathbf{u}$ of L such that $\|\mathbf{u}\| \leqslant \gamma \min_{\mathbf{v} \in L \setminus \{\mathbf{0}\}} \|\mathbf{v}\|$.

4.3.3 Lattice basis reduction

Geometrically, the process of determining a lattice basis with reasonably short vectors (*lattice basis reduction*) implies that such vectors are not far from being orthogonal to one other. In finite-dimensional vector spaces, this could correspond to finding an orthonormal basis.

Definition 4.12 (Gram-Schmidt orthogonalization)**.** Let $\mathbf{b}_1, \ldots, \mathbf{b}_d$ be linearly independent vectors in $\mathbb{R}^n$; their *Gram-Schmidt orthogonalization* (GSO) is the orthogonal family $\mathbf{b}_1^\star, \ldots, \mathbf{b}_d^\star$ defined as follows: $\mathbf{b}_1^\star = \mathbf{b}_1$ and more generally $\mathbf{b}_i^\star$ is the orthogonal projection of $\mathbf{b}_i$ on the space orthogonal to the linear span of $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}$. Iteratively, we have

$$\mathbf{b}_i^\star = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j} \mathbf{b}_j^\star, \qquad \text{where } \mu_{i,j} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j^\star \rangle}{\langle \mathbf{b}_j^\star, \mathbf{b}_j^\star \rangle}. \tag{4.11}$$

It is easy to verify that, for all $1 \leqslant i \leqslant d$, $\operatorname{span}(\mathbf{b}_1, \ldots, \mathbf{b}_i) = \operatorname{span}(\mathbf{b}_1^\star, \ldots, \mathbf{b}_i^\star)$ and that the GSO family depends on the order of the basis vectors. Even though the $\mathbf{b}_i^\star$'s do not usually belong to L, they are strongly connected to the characteristics of L.

The main reason for this is that Gram-Schmidt orthogonalization triangulates a given basis. Indeed, if we express $\mathbf{b}_1, \ldots, \mathbf{b}_d$ using the orthonormal basis $(\mathbf{b}_1^\star / \|\mathbf{b}_1^\star\|_2, \ldots, \mathbf{b}_d^\star / \|\mathbf{b}_d^\star\|_2)$, then the coordinates of each $\mathbf{b}_i$ in this basis are given by the lines in the triangular matrix

$$\begin{bmatrix} \|\mathbf{b}_1^\star\|_2 & 0 & \cdots & 0 \\ \mu_{2,1} \|\mathbf{b}_1^\star\|_2 & \|\mathbf{b}_2^\star\|_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ \mu_{d,1} \|\mathbf{b}_1^\star\|_2 & \cdots & \mu_{d,d-1} \left\|\mathbf{b}_{d-1}^\star\right\|_2 & \|\mathbf{b}_d^\star\|_2 \end{bmatrix}.$$

The reader can check that the lattice L spanned by $\mathbf{b}_1, \ldots, \mathbf{b}_d$ satisfies

$$\det(L) = \prod_{i=1}^{d} \|\mathbf{b}_i^\star\|_2 .$$

Because of (4.11), $\|\mathbf{b}_i\|_2 \geqslant \|\mathbf{b}_i^\star\|_2$ and hence $\det(L) \leqslant \prod_{i=1}^d \|\mathbf{b}_i\|_2$. This leads to a possible way of quantifying how orthogonal a given lattice basis is:

Definition 4.13. If $\mathbf{B}$ is a basis for a rank d lattice $L \subset \mathbb{R}^n$, then the *orthogonality defect* of $\mathbf{B}$ is the constant

$$\delta(\mathbf{B}) = \frac{\prod_{i=1}^d \|\mathbf{b}_i\|_2}{\det(L)}.$$

It follows that $\delta(\mathbf{B}) \geqslant 1$, with equality if and only if $\mathbf{B}$ is composed of orthogonal vectors.

Another consequence of the GSO process is the following:

Lemma 4.14. *If $\mathbf{B}$ is a basis of a rank d lattice $L \subset \mathbb{R}^n$ in the ℓ_2 setting, then its GSO satisfies*

$$\lambda_1(L) \geqslant \min_{1 \leqslant i \leqslant d} \|\mathbf{b}_i^\star\|_2 .$$

Proof. Take $\mathbf{v} = \sum_{i=1}^{n} a_i \mathbf{b}_i \in L$ and assume its length equals the first successive minimum of L. At least one of the a_j coefficients is nonzero (take j to be the smallest such index). Representing $\mathbf{v}$ in the orthonormal basis $(\mathbf{b}_1^\star/\|\mathbf{b}_1^\star\|_2, \ldots, \mathbf{b}_d^\star/\|\mathbf{b}_d^\star\|_2)$ gives $\lambda_1(L) = \|\mathbf{v}\|_2 \geqslant |a_j| \left\|\mathbf{b}_j^\star\right\|_2 \geqslant \min_{1 \leqslant i \leqslant d} \|\mathbf{b}_i^\star\|_2$. □

Of particular interest to us should be the fact that $\mathbf{b}_1 = \mathbf{b}_1^\star$. If the Gram-Schmidt vectors have increasing lengths, then $\mathbf{b}_1$ will be a SVP solution in ℓ_2. Such a basis seems to be inexistent in most cases, and even if one did exist, NP-hardness would make it difficult to compute. Despite this, we will see that there are polynomial time algorithms that allow us to construct a basis for which its Gram-Schmidt vectors do not decrease in length too fast.

Size-reduction

A first step for introducing such algorithms uses an idea due to Hermite [101]:

Definition 4.15 (Size-reduction)**.** A basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ of a lattice $L \subset \mathbb{R}^n$ is called *size-reduced* if its Gram-Schmidt coefficients $\mu_{i,j}$ satisfy

$$|\mu_{i,j}| \leqslant \frac{1}{2},$$

for all $1 \leqslant j < i \leqslant d$.

Geometrically, this notion states that each $\mathbf{b}_i$ is already as close as possible to the orthogonal complement of $\mathbf{b}_j^\star$, for all $j < i$ (*i.e.,* by projecting $\mathbf{b}_i$ on the orthogonal complement of $\mathbf{b}_j^\star$, less than half the vector $\mathbf{b}_j^\star$ needs to be added or subtracted). In other words, it is a basis which is closest to its GSO. From the Pythagorean theorem and $\mathbf{b}_i = \mathbf{b}_i^\star + \sum_{j<i} \mu_{i,j} \mathbf{b}_j^\star$, it follows that

$$\|\mathbf{b}_i^\star\|_2^2 \leqslant \|\mathbf{b}_i\|_2^2 \leqslant \|\mathbf{b}_i^\star\|_2^2 + \frac{1}{4} \sum_{j=1}^{i-1} \left\|\mathbf{b}_j^\star\right\|_2^2,$$

meaning the size of the basis vectors are not much different from those of its GSO.

Computing a size-reduced basis is fairly straightforward if we tweak (4.11) (the reader should check that it is indeed the case):

Algorithm 8: SIZE-REDUCE

Input: a basis $\mathbf{B}$ of a rank d lattice $L \subset \mathbb{R}^n$
Output: a size-reduced basis $\mathbf{B}'$ of L

$\mathbf{B}^\star \leftarrow \mathrm{GSO}(\mathbf{B})$
$\mathbf{b}'_1 \leftarrow \mathbf{b}_1$
for $k \leftarrow 2$ ***to*** d **do**
  $\mathbf{b}'_k \leftarrow \mathbf{b}_k$
  for $j \leftarrow k-1$ ***downto*** 1 **do**
    $\mathbf{b}'_k \leftarrow \mathbf{b}'_k - \left\lfloor \frac{\langle \mathbf{b}'_k, \mathbf{b}_j^\star \rangle}{\langle \mathbf{b}_j^\star, \mathbf{b}_j^\star \rangle} \right\rceil \mathbf{b}_j$
return $(\mathbf{b}'_1, \ldots, \mathbf{b}'_d)$

Size-reduction is a rather weak property by itself. This is due, most of all, to the fact that it does not change the $\|\mathbf{b}_i^\star\|_2$'s, not directly serving our goal of having a basis with slowly decreasing GSO vectors. Also, it does not necessarily bound the length of the $\mathbf{b}_i$'s in Definition 4.15. A nice example, taken from [209, Ch. 2.2.1], consists of generating $\mathbb{Z}^2$ by using the basis

$$\mathbf{B}_n = \begin{bmatrix} F_{n+1} & F_{n-1} \\ F_{n+2} & F_n \end{bmatrix},$$

where F_n is the n^{th} Fibonacci number (*i.e.*, $F_0 = F_1 = 1, F_n = F_{n-1} + F_{n-2}, n \geqslant 2$). B_n is size-reduced because $F_k = (F_{k+2} - F_{k-1})/2 \leqslant F_{k+2}/2$ and hence

$$|\mu_{2,1}| = \frac{F_{n+1}F_{n-1} + F_{n+2}F_n}{F_{n+1}^2 + F_{n+2}^2} \leqslant \frac{F_{n+1}^2/2 + F_{n+2}^2/2}{F_{n+1}^2 + F_{n+2}^2} = \frac{1}{2}.$$

As we increase n, the columns of $\mathbf{B}_n$ become arbitrarily large.

Nevertheless, size-reduction does serve as an important ingredient for stronger types of basis reductions.

LLL-reduction

Up to this point, our angle of attack was to *only* modify the basis vectors. Since we expect a size-reduced basis to be close to its Gram-Schmidt vectors, it is natural to try and update the basis vectors in such a way that the quality of its GSO *also* improves (in the sense of successive GSO vectors with slowly decreasing lengths). A notion of reduction achieving this is the following:

Definition 4.16 (LLL-reduction)**.** A basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ of a lattice $L \subset \mathbb{R}^n$ is said to be *LLL-reduced* with factor $\delta \in \left(\frac{1}{4}, 1\right)$ if it is size-reduced and

$$\delta \left\|\mathbf{b}_i^\star\right\|_2^2 \leqslant \left\|\mathbf{b}_{i+1}^\star + \mu_{i+1,i}\mathbf{b}_i^\star\right\|_2^2, \tag{4.12}$$

for all $1 < i \leqslant d$.

The inequality (4.12) is known as the *Lovász condition* and is equivalent to

$$(\delta - \mu_{i+1,i}^2) \left\|\mathbf{b}_i^\star\right\|_2^2 \leqslant \left\|\mathbf{b}_{i+1}^\star\right\|_2^2,$$

basically saying that $\mathbf{b}_{i+1}^\star$ is not much shorter than $\mathbf{b}_i^\star$. The decrease in the size of the $\mathbf{b}_i^\star$'s is at most geometric. Indeed, because of size-reduction ($\mu_{i+1,i}^2 \leqslant 1/4$) we have $\left\|\mathbf{b}_{i+1}^\star\right\|_2^2 \geqslant (\delta - 1/4) \left\|\mathbf{b}_i^\star\right\|_2^2$, which coupled with Lemma 4.14 also gives a bound on the SVP solution with respect to $\mathbf{b}_1$:

$$\lambda_1(L) \geqslant \sqrt{(\delta - 1/4)^{d-1}} \left\|\mathbf{b}_1\right\|_2.$$

The LLL tag comes from the names of A. K. Lenstra, H. W. Lenstra and L. Lovász, whom, in 1982, devised an algorithm [132] to compute lattice bases satisfying Definition 4.16 as a method to factor polynomials with rational coefficients in polynomial time. In its simplest form, the LLL algorithm works as follows:

Algorithm 9: LLL

Input: a basis $\mathbf{B}$ of a rank d lattice $L \subset \mathbb{R}^n, \delta \in \left(\frac{1}{4}, 1\right)$

// at the end of execution, $\mathbf{B}$ will be δ-LLL-reduced

1 $\mathbf{B} \leftarrow$ Size-Reduce($\mathbf{B}$)

2 **if** there exists a (smallest) index i violating the Lovász condition (4.12) **then**

3 swap $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$ and **goto** 1

A proof of the fact that this algorithm terminates in a finite number of operations can be consulted in [132] or [64, Ch. 2.6]. It is nonetheless constructive to sketch the idea here. Each time two vectors $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$ get swapped, only their GSO vectors change. If we denote these new GSO vectors with $\mathbf{c}_i^\star$ and $\mathbf{c}_{i+1}^\star$, then $\left\|\mathbf{c}_i^\star\right\|_2 \cdot \left\|\mathbf{c}_{i+1}^\star\right\|_2 = \left\|\mathbf{b}_i^\star\right\|_2 \cdot \left\|\mathbf{b}_{i+1}^\star\right\|_2$, a consequence of the fact that the product of the GSO vector lengths is a lattice invariant. Since the Lovász condition does not hold, we have $\left\|\mathbf{c}_i^\star\right\|_2^2 = \left\|\mathbf{b}_{i+1}^\star + \mu_{i+1,i}\mathbf{b}_i^\star\right\|_2^2 < \delta \left\|\mathbf{b}_i^\star\right\|_2^2$, which coupled with the last equality gives

$$\left\|\mathbf{c}_i^\star\right\|_2^{2(d-i+1)} \cdot \left\|\mathbf{c}_{i+1}^\star\right\|_2^{2(d-i)} < \delta \left\|\mathbf{b}_i^\star\right\|_2^{2(d-i+1)} \cdot \left\|\mathbf{b}_{i+1}^\star\right\|_2^{2(d-i)}.$$

This tells us that the quantity

$$D = \|\mathbf{b}_1^\star\|_2^{2d} \|\mathbf{b}_2^\star\|_2^{2(d-1)} \cdot \ldots \cdot \|\mathbf{b}_d^\star\|_2^2,$$

the product of the Gram determinants of the bases $(\mathbf{b}_1), (\mathbf{b}_1, \mathbf{b}_2), \ldots, (\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_d)$, decreases by a factor $\delta < 1$ each time we perform a swap. It can be shown that D is lower bounded by a positive value depending only on L, meaning the number of swaps performed must be finite.

The LLL algorithm stands as one of the most important results in lattice theory. It is effective in producing bases with relatively short vectors in many practical situations at a running time which is polynomially bounded by the rank and dimension of the lattice. If the basis elements have integer entries and $B = \max_{1 \leqslant i \leqslant d} \|\mathbf{b}_i\|_2$, then the bit-complexity required to LLL-reduce a lattice is $\mathcal{O}(d^5 n \log^3 B)$.

The original version of the LLL algorithm assumes all computations are done in exact arithmetic and is virtually never used. This is because most of the computation time is spent on the GSO part of the algorithm and using exact arithmetic means the $\mu_{i,j}$'s will have bit sizes up to $\mathcal{O}(d \log B)$, making their exact computation quite costly. That is why in practice, the GSO computations of the LLL algorithm are always done using floating-point arithmetic, which, if performed with $\mathcal{O}(d)$ precision, can produce a provably correct result. We note in particular the work of Nguyen and Stehlé [155], [156, Ch. 5], whose L^2 floating-point version of the LLL algorithm we use for our problems and has a bit complexity of $\mathcal{O}(d^4 n \log B(d + \log B))$.

HKZ-reduction

A very strong reduction notion, with better properties than what we have presented so far is

Definition 4.17 (HKZ-reduction)**.** An ordered basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ of a lattice $L \subset \mathbb{R}^n$ is called *Hermite-Korkine-Zolotarev-reduced* if it is size-reduced, $\|\mathbf{b}_1\|_2 = \lambda_1(L)$ and the basis $\{\mathbf{b}_2 - \mu_{2,1}\mathbf{b}_1, \ldots, \mathbf{b}_n - \mu_{n,1}\mathbf{b}_1\}$ (the orthogonal projection of the basis of L onto the orthogonal complement of $\mathbf{b}_1$) is also HKZ-reduced.

Their appeal is mainly due to the fact that:

Theorem 4.18. *If $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ is a* HKZ*-reduced basis of $L \subset \mathbb{R}^n$, then*

$$\frac{4}{i+3}(\lambda_i(L))^2 \leqslant \|\mathbf{b}_i\|_2^2 \leqslant \frac{i+3}{4}(\lambda_i(L))^2,$$

for all $1 \leqslant i \leqslant d$.

Proof. See for instance [126, Sec. 3]. □

Despite this, the use of HKZ-reduced bases is limited due to no known polynomial-time algorithms to compute them. For lattices of small rank though, it is practical to *enumerate* all short vectors (see for instance [85, Ch. 18.4] and the references therein for short lattice vector enumeration techniques) and hence, compute a HKZ-reduced basis.

BKZ-reduction

Schnorr [184] introduced the *block Korkine-Zolotarev* (BKZ) lattice basis reduction algorithm. It combines the LLL algorithm with the computation of HKZ-reduced bases for small-rank projections of the original lattice. Its output is generally of better quality than that of a LLL-reduced basis. The small rank is usually denoted with $\beta, 2 \leqslant \beta \leqslant d$ and is called the *block size*. We will not go into any details regarding the BKZ algorithm since it is out of the scope of the present text.

To summarize these last two subsections, dealing with lattice basis reduction and/or the SVP problem is a matter of compromises. A popular (yet oversimplified) rule of thumb in the community is that for any value of the parameter $k, 1 \leqslant k \leqslant d$, one can solve (given the current state of the art) an SVP instance in the ℓ_2 norm within an approximation factor of $2^{\mathcal{O}(d/k)}$ in time $\mathcal{O}(2^k \mathrm{poly}\,(d))$. This is showcased in Table 4.3 for the basis reduction algorithms we presented here. We remark the fact that for $\beta = 2$, the BKZ algorithm is akin to LLL, whereas for $\beta = d$, it is identical to HKZ.

Table 4.3 discusses *worst-case* complexities and approximation factors. The average behavior of LLL is also studied experimentally in [154]; roughly speaking, on average, LLL retains a c^d approximation

Table 4.3: Lattice basis reduction algorithms

Algorithm	SVP approximation factor	Time
LLL_δ [132]	$(\delta - 1/4)^{-(d-1)/2}$	poly (d)
BKZ_β [97, 184]	$\approx \beta^{d/\beta}$	poly $(d)2^{\mathcal{O}(\beta)}$
HKZ [113]	1	$2^{\mathcal{O}(d)}$

factor, but the constant c drops down to roughly 1.05. It seems highly plausible that similar facts hold for BKZ [55]. We already point out that *in our setting*, the performances of all these algorithms, both in terms of approximation factor and of complexity, seem significantly better than expected – HKZ-reducing a random lattice of rank greater than 60 is, as of today, a considerable task, whereas we were able to do it on instances of rank up to 101 coming from our problems in less than half a minute (see the discussions in Sections 4.4 and 4.5).

4.3.4 The closest vector problem

Another important question in lattice theory is that of the closest lattice vector to an arbitrary point $\mathbf{t} \in \mathbb{R}^n$, the inhomogeneous extension of the shortest vector problem:

Definition 4.19 (Closest Vector Problem (CVP))**.** Given a d-rank lattice $L \subset \mathbb{R}^n$ and a target vector $\mathbf{t} \in \mathbb{R}^n$, find a lattice vector $\mathbf{x} \in L$ such that $\|\mathbf{t} - \mathbf{x}\| = \text{dist}(\mathbf{t}, L)$.

Similar to SVP, van Emde Boas showed this problem to be NP-hard, for the ℓ_2 and ℓ_∞ norms [214]. If $L \subseteq \mathbb{Z}^n$, then CVP is also NP-hard to approximate within any constant factor or sub-polynomially in the lattice rank [14, 77]. From the same complexity point of view, CVP is regarded as being harder than SVP in the same dimension, due to the fact that a CVP oracle can be used to solve a SVP instance [91, 99].

It turns out nonetheless that lattice basis reduction can be an effective way to find approximate solutions to the closest vector problem in the ℓ_2 setting. We briefly present three such approaches here. The first two are due to Babai [15], while the idea for the third one comes from Kannan [114].

Babai's algorithms

The idea for Babai's rounding algorithm is a very simple one, but assumes that $\mathbf{t} \in \text{span}(L)$. If L has a basis B and is of rank d, then $\mathbf{t} = \sum_{i=1}^{d} \alpha_i \mathbf{b}_i, \alpha_i \in \mathbb{R}$ and we just take $\mathbf{x} = \sum_{i=1}^{d} \lfloor \alpha_i \rceil \, \mathbf{b}_i \in L$. This method is shown to solve CVP to within an approximation factor $\gamma(d) = 1 + 2d(9/2)^{d/2}$ if $\mathbf{B}$ is LLL-reduced with $\delta = 3/4$. In particular, if $\mathbf{B}$ is orthogonal, then $\mathbf{x}$ would be the exact ℓ_2 norm CVP solution. When $\mathbf{t} \notin \text{span}(L)$, then we can first orthogonally project $\mathbf{t}$ onto $\text{span}(L)$ to apply the method.

Babai's nearest plane algorithm, on the other hand, is a bit more involved, but can essentially be viewed as the size reduction of $\mathbf{t}$ with respect to $\mathbf{b}_1, \ldots, \mathbf{b}_d$. The starting point is to consider $\mathbf{t}$ as expressed with respect to the GSO vectors of $\mathbf{B}$, with the lattice vector $\mathbf{x} = \sum_{i=1}^{d} \alpha_i \mathbf{b}_i$ being constructed incrementally as follows. First $\mathbf{t}$ is projected onto $\mathbf{b}_d^\star$, giving the coefficient $\alpha_d = \lfloor \langle \mathbf{t}, \mathbf{b}_d^\star \rangle / \langle \mathbf{b}_d^\star, \mathbf{b}_d^\star \rangle \rceil$. The target vector $\mathbf{t}$ gets updated to $\mathbf{t} - \alpha_d \mathbf{b}_d$ and the process is repeated $d-1$ more times for each of the Gram-Schmidt vectors $\mathbf{b}_{d-1}^\star, \ldots, \mathbf{b}_1^\star$. Its name comes from the fact that each $\mathbf{b}_i^\star$ can be seen as defining a plane, with $\mathbf{x}$ emerging as we repeatedly take the plane closest to $\mathbf{t}$.

Just like with the rounding algorithm, the nearest plane approach depends on the quality of the initial basis $\mathbf{B}$. If it is LLL-reduced, then we get a CVP approximation factor of $\sqrt{\frac{(\delta - 1/4)^{-(d-1)}}{(5/4 - \delta)}}$, which for $\delta = 3/4$ (the default reduction parameter generally considered in the literature) stands at $2^{d/2}$, slightly better than with rounding.

Kannan's embedding

For this approach, given a basis $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ of $L \subset \mathbb{R}^n$ and the target vector $\mathbf{t} \in \mathbb{R}^n$, we form a $d+1$-rank lattice $L' \subset \mathbb{R}^{n+1}$ generated by the basis vectors

$$\underbrace{\begin{bmatrix} \mathbf{b}_1 \\ 0 \end{bmatrix}}_{\mathbf{b}'_1}, \ldots, \underbrace{\begin{bmatrix} \mathbf{b}_d \\ 0 \end{bmatrix}}_{\mathbf{b}'_d}, \underbrace{\begin{bmatrix} \mathbf{t} \\ \gamma \end{bmatrix}}_{\mathbf{t}'},$$

where γ is a real parameter to be chosen later on.

Now, if $\mathbf{x} \in \mathbb{R}^{n+1}$ is a short vector of the lattice L', then there exist integers $u_1, \ldots, u_{d+1}$ such that $\mathbf{x} = \sum_{i=1}^{d} u_i \mathbf{b}'_i + u_{d+1}\mathbf{t}'$ and hence

$$\|\mathbf{x}\|_2^2 = \left\| \sum_{i=1}^{d} u_i \mathbf{b}_i + u_{d+1}\mathbf{t} \right\|_2^2 + \gamma^2 u_{d+1}^2. \tag{4.13}$$

Since $\mathbf{x}$ is assumed to be short (for instance, a vector in a reduced basis of L'), then $\left\| \sum_{i=1}^{d} u_i \mathbf{b}_i + u_{d+1}\mathbf{t} \right\|_2^2$ is also small. If $u_{d+1} = \pm 1$, then $\mp \sum_{i=1}^{d} u_i \mathbf{b}_i \in L$ is close to the vector $\mathbf{t}$.

The key question becomes how to choose γ. For a sufficiently large γ, small vectors in L' will tend to have a 0 last coordinate, except in a reduced basis of L', where at least one vector needs to have a nonzero last coordinate. If we take $\gamma \geqslant \max_{i=1}^{d} \|\mathbf{b}_i\|_2$, then as soon as $|u_{d+1}| \neq 0$, we will have $\|\mathbf{x}\|_2 \geqslant \gamma \geqslant \|\mathbf{b}_i\|_2$ for all $i \leqslant d$. So, as reducing bases tends to produce shorter vectors, if $\mathbf{c}_1, \ldots, \mathbf{c}_d$ is a reduced basis of L, we can expect that for all $j \leqslant d$, $\|\mathbf{c}_j\|_2 < \gamma$.

Now consider $\mathbf{c}'_1, \ldots, \mathbf{c}'_{d+1}$ to be a reduced basis of L'. As any vector $\mathbf{x} \in L'$ with a nonzero last coordinate will have $\|\mathbf{x}\|_2 > \|\mathbf{c}_j\|_2$ for all j, we expect that $\mathbf{c}'_j = [\mathbf{c}_j \ 0]^T$ for $j \leqslant d$. Hence $\mathbf{c}'_{d+1}$ has to have a nonzero last coordinate. As γ is large, we assume this last coordinate to be $\pm\gamma$, otherwise, by expressing $\mathbf{c}'_{d+1}$ like in (4.13), the term $\gamma^2 u_{d+1}^2$ is large and it is likely that some shorter vector will be found by the reduction process. Thus, from $\mathbf{c}'_{d+1}$ we will often be able to deduce a vector close to $\mathbf{t}$.

In our experiments, the (heuristic) choice $\gamma = \max_{i=1}^{d} \|\mathbf{b}_i\|_2$ yielded a vector with $u_{d+1} = \pm 1$ in all cases. In particular, if $d = n$ and we perform LLL-reduction on the initial basis of L', then the embedding technique can be shown to produce the same output as Babai's nearest plane algorithm [85, Ex. 18.3.3]. We also note that Kannan's description in [114] is somewhat different, since it deals with another context and needs the target vector $\mathbf{t}$ to be rather close to L – an assumption that we cannot make about our current setting.

For the rest of the chapter, when talking about approximately solving a CVP instance, we will be doing so using the embedding technique. This choice is motivated by the fact that the result we get is generally on par or better than what we would obtain with the nearest plane method, but also because it is extremely straightforward to implement.

4.4 FIR filter design using the FPminimax algorithm

We have already seen that, given a linear phase FIR specification in terms of target number of taps and the associated degree, their fixed-point format (m_-, m_+, s), filter type, $\Omega, D(\omega)$ and $W(\omega)$, we can express the weighted L_∞ design problem in the context of Problem 4.1 with the $\mathcal{F}_i$ formats as subsets of integers.

Having done that, the overall steps of the heuristic approach we mentioned in Section 4.2.2 are:

1. use the second Remez algorithm to obtain an infinite precision solution $p^\star$ to Problem 4.1;
2. choose $m+1 \geqslant n+1$ points $z_0, \ldots, z_m \in X$ to approximately interpolate $p^\star$;
3. solve the resulting CVP problem (4.9) by using Kannan's embedding technique with the LLL, BKZ or HKZ algorithm and retrieve the filter coefficients. Note that we need to check that the obtained

coefficients a_i satisfy $m_- \leqslant a_i \leqslant m_+$, even though we have never encountered a case in practice where this is not so.

The first step is treated in considerable detail in Chapter 3, so our focus here will be mostly towards the second and third steps. We will again consider the specification of Example 4.2 as a worked example. Note that if we had chosen to use (4.10) instead of (4.9), step 1 can be bypassed.

4.4.1 Suitable interpolation points

Since we are doing something akin to weighted polynomial interpolation, it seems reasonable to pick a discretization of X which behaves well with respect to interpolation. From Section 3.6.3, good choices are points which follow the equilibrium distribution of X. In our case, those could be:

(a) the final reference set of $p^\star$, when applying the exchange algorithm $p^\star$ (*i.e.,* the local extrema of $e^\star = f - p^\star$ over X);

(b) the zeros of $e^\star$ over X together with the endpoints of the intervals composing X;

(c) the approximate Fekete points we proposed to initialize the exchange algorithm in Section 3.6.

Choices (a) and (c) give us a number of $n+2$ points each, whereas (b) will also result in at least $n+2$ points (if X is composed of b bands, we will have at least $n+2-b$ zeros, one between each pair of final reference points of $p^\star$ located inside the same subinterval of X and the $2b$ endpoints, resulting in at least $n+2+b$ points). Based on our discussion from Section 3.6, the number of AFPs we generate can be varied, so taking a size $n+1$ or larger AFP-based discretization is also possible.

Besides the fact that such points are good for interpolation/approximation purposes, the lattices that they generate for eq. (4.9) have very well-behaved starting bases. To show this, let us first consider the case where $X = [-1, 1]$ and the particular grid of $n+1$-st order Chebyshev nodes of the second kind $\nu_i = \cos\left(\frac{(n-k)\pi}{n}\right), k = 0, \ldots, n$, which follow the equilibrium distribution of X for polynomial approximation (see for instance [205, Ch. 12]). For a type I degree n filter with uniform weight over $[-1, 1]$, the basis elements will then be (looking at Table 4.1 and abstracting out the scaling factor by taking $s = 2$)

$$\underbrace{\begin{bmatrix} T_0(\nu_0)/2 \\ T_0(\nu_1)/2 \\ \vdots \\ T_0(\nu_n)/2 \end{bmatrix}}_{\mathbf{b}_0}, \underbrace{\begin{bmatrix} T_1(\nu_0) \\ T_1(\nu_1) \\ \vdots \\ T_1(\nu_n) \end{bmatrix}}_{\mathbf{b}_1}, \ldots, \underbrace{\begin{bmatrix} T_n(\nu_0) \\ T_n(\nu_1) \\ \vdots \\ T_n(\nu_n) \end{bmatrix}}_{\mathbf{b}_n}. \tag{4.14}$$

Lemma 4.20. *If* $\mathbf{G} = [g_{ij}] = \mathbf{B}^T\mathbf{B}, 0 \leqslant i, j \leqslant n$, *is the Gram matrix of the lattice basis* $\mathbf{B}$ *whose columns are the vectors of (4.14), then we have*

$$\begin{aligned} g_{00} &= \frac{1}{4}(n+1), \\ g_{ii} &= \frac{1}{2}n + 1, 0 < i < n, \\ g_{nn} &= n+1, \\ g_{i0} = g_{0i} &= \frac{1+(-1)^i}{4}, 0 < i \leqslant n, \\ g_{ij} &= \frac{1+(-1)^{i+j}}{2}, i \neq j; 0 < i, j \leqslant n. \end{aligned}$$

Proof. This result is a direct consequence of the discrete orthogonality property of Chebyshev polynomials (see Proposition 2.9), which tells us that, if

$$d_{ij} = \frac{1}{2}T_i(\nu_0)T_j(\nu_0) + \sum_{k=1}^{n-1} T_i(\nu_k)T_j(\nu_k) + \frac{1}{2}T_i(\nu_n)T_j(\nu_n),$$

then we have $d_{00} = d_{nn} = n, d_{ii} = \frac{1}{2}n$ for $0 < i < n$ and $d_{ij} = 0$ when $i \neq j; i, j \leqslant n$.

We indeed get

$$g_{00} = \frac{1}{4}\sum_{k=0}^{n} T_0^2(\nu_k) = \frac{1}{4}\left(d_{00} + \frac{1}{2}T_0^2(\nu_0) + \frac{1}{2}T_0^2(\nu_n)\right) = \frac{1}{4}(n+1),$$

$$g_{ii} = d_{ii} + \frac{1}{2}\left(T_i^2(\nu_0) + T_i^2(\nu_n)\right) = \frac{1}{2}n + 1,$$

$$g_{nn} = d_{nn} + \frac{1}{2}\left(T_n^2(\nu_0) + T_n^2(\nu_n)\right) = n + 1,$$

$$g_{i0} = g_{0i} = \frac{1}{2}\sum_{k=0}^{n} T_0(\nu_k)T_i(\nu_k) = \frac{1}{2}\left(d_{i0} + \frac{1}{2}T_0(\nu_0)T_i(\nu_0) + \frac{1}{2}T_0(\nu_n)T_i(\nu_n)\right) = \frac{1+(-1)^i}{4},$$

and

$$g_{ij} = d_{ij} + \frac{1}{2}T_i(\nu_0)T_j(\nu_0) + \frac{1}{2}T_i(\nu_n)T_j(\nu_n) = \frac{1+(-1)^{i+j}}{2}.$$

□

This result tells us that the Gram matrix is almost diagonal, meaning the basis vectors $\mathbf{b}_0, \ldots, \mathbf{b}_n$ are very close to being orthogonal. If we think about basis reduction, ideally there will also be an almost orthogonal feel to the reduced basis vectors because of the invariant nature of the lattice volume.

As a consequence, the associated Gram-Schmidt vectors will have slow varying lengths:

Lemma 4.21. *The Gram-Schmidt orthogonalization $\mathbf{b}_0^\star, \mathbf{b}_1^\star, \ldots, \mathbf{b}_n^\star$ of the basis vectors from (4.14) satisfies*

$$\mathbf{b}_k^\star = \mathbf{b}_k - \frac{2}{n+k-1}\sum_{j=1}^{\lfloor \frac{k}{2} \rfloor} \mathbf{b}_{k-2j}, k = 1, \ldots, n, \tag{4.15}$$

$$\|\mathbf{b}_k^\star\|_2^2 = \frac{n(n+k+1)}{2(n+k-1)}, k = 1, \ldots, n-1, \tag{4.16}$$

and

$$\|\mathbf{b}_n^\star\|_2^2 = \frac{2n^2}{2n-1}. \tag{4.17}$$

Proof. By the GSO definition, $\mathbf{b}_0^\star = \mathbf{b}_0$. Let us look at what happens for $\mathbf{b}_1^\star, \mathbf{b}_2^\star, \mathbf{b}_3^\star$. We have $\mathbf{b}_1^\star = \mathbf{b}_1$ because $\mathbf{b}_1 \perp \mathbf{b}_0$, and thus $\|\mathbf{b}_1^\star\|_2^2 = g_{11} = \frac{n(n+1+1)}{2(n+1-1)}$.

$$\mathbf{b}_2^\star = \mathbf{b}_2 - \frac{\langle \mathbf{b}_2, \mathbf{b}_1^\star \rangle}{\langle \mathbf{b}_1^\star, \mathbf{b}_1^\star \rangle}\mathbf{b}_1^\star - \frac{\langle \mathbf{b}_2, \mathbf{b}_0^\star \rangle}{\langle \mathbf{b}_0^\star, \mathbf{b}_0^\star \rangle}\mathbf{b}_0^\star = \mathbf{b}_2 - \frac{g_{20}}{g_{00}}\mathbf{b}_0 = \mathbf{b}_2 - \frac{2}{n+1}\mathbf{b}_0,$$

which means that

$$\begin{aligned}\|\mathbf{b}_2^\star\|_2^2 &= \left\langle \mathbf{b}_2 - \frac{2}{n+1}\mathbf{b}_0, \mathbf{b}_2 - \frac{2}{n+1}\mathbf{b}_0 \right\rangle = \|\mathbf{b}_2\|_2^2 + \frac{4}{(n+1)^2}\|\mathbf{b}_0\|_2^2 - \frac{4}{n+1}\langle \mathbf{b}_0, \mathbf{b}_2 \rangle \\ &= \frac{n}{2} + 1 + \frac{1}{n+1} - \frac{2}{n+1} = \frac{n^2+3n+2-2}{2(n+1)} = \frac{n(n+2+1)}{2(n+2-1)}.\end{aligned}$$

Similarly

$$\mathbf{b}_3^\star = \mathbf{b}_3 - \frac{\langle \mathbf{b}_3, \mathbf{b}_2^\star \rangle}{\langle \mathbf{b}_2^\star, \mathbf{b}_2^\star \rangle}\mathbf{b}_2^\star - \frac{\langle \mathbf{b}_3, \mathbf{b}_1^\star \rangle}{\langle \mathbf{b}_1^\star, \mathbf{b}_1^\star \rangle}\mathbf{b}_1^\star - \frac{\langle \mathbf{b}_3, \mathbf{b}_0^\star \rangle}{\langle \mathbf{b}_0^\star, \mathbf{b}_0^\star \rangle}\mathbf{b}_0^\star = \mathbf{b}_3 - \frac{g_{31}}{g_{11}}\mathbf{b}_1 = \mathbf{b}_3 - \frac{2}{n+2}\mathbf{b}_1$$

and

$$\|\mathbf{b}_3^\star\|_2^2 = \left\langle \mathbf{b}_3 - \frac{2}{n+2}\mathbf{b}_1, \mathbf{b}_3 - \frac{2}{n+2}\mathbf{b}_1 \right\rangle = \|\mathbf{b}_3\|_2^2 + \frac{4}{(n+2)^2}\|\mathbf{b}_1\|_2^2 - \frac{4}{n+2}\langle \mathbf{b}_1, \mathbf{b}_3\rangle$$
$$= \frac{n}{2} + 1 + \frac{2}{n+2} - \frac{4}{n+2} = \frac{n^2+4n+4-4}{2(n+2)} = \frac{n(n+3+1)}{2(n+3-1)}.$$

The rest follows from induction. Assume that the requested equalities hold true for $\mathbf{b}_1^\star, \ldots, \mathbf{b}_{k-1}^\star$. We will only consider the case when $k = 2m < n$; odd $k < n$ can be treated analogously.

From the GSO definition and the target identities being true for $i < 2m$, we have that

$$\mathbf{b}_{2m}^\star = \mathbf{b}_{2m} - \sum_{j=0}^{m-1} \mu_{2m,2j}\mathbf{b}_{2j}^\star.$$

Writing each $\mathbf{b}_{2j}^\star$ in this last expression in terms of the original basis vectors as given by the (4.15) identities already established, gives

$$\mathbf{b}_{2m}^\star = \mathbf{b}_{2m} + \sum_{j=0}^{m-1} c_j \mathbf{b}_{2j},$$

where

$$c_j = -\mu_{2m,2j} + \sum_{\ell=j+1}^{m-1} \frac{2}{n+2\ell-1}\mu_{2m,2\ell}.$$

We need to show that $c_j = -\frac{2}{n+2m-1}, 0 \leqslant j < m$. If $\ell < m$,

$$\mu_{2m,2\ell} = \frac{\langle \mathbf{b}_{2m}, \mathbf{b}_{2\ell}^\star\rangle}{\langle \mathbf{b}_{2\ell}^\star, \mathbf{b}_{2\ell}^\star\rangle} = \frac{\left\langle \mathbf{b}_{2m}, \mathbf{b}_{2\ell} - \frac{2}{n+2\ell-1}\mathbf{b}_{2\ell-2} - \frac{2}{n+2\ell-1}\mathbf{b}_{2\ell-4} - \ldots - \frac{2}{n+2\ell-1}\mathbf{b}_0 \right\rangle}{\frac{n(n+2\ell+1)}{2(n+2\ell-1)}}$$
$$= \frac{1 - \overbrace{\frac{2}{n+2\ell-1} - \ldots - \frac{2}{n+2\ell-1}}^{\ell-1} - \frac{1}{n+2\ell-1}}{\frac{n(n+2\ell+1)}{2(n+2\ell-1)}} = \frac{2}{n+2\ell+1},$$

which then results in

$$c_j = -\frac{2}{n+2j+1} + \sum_{\ell=j+1}^{m-1} \frac{4}{(n+2\ell-1)(n+2\ell+1)}$$
$$= -\frac{2}{n+2j+1} + \sum_{\ell=j+1}^{m-1} \left(\frac{2}{n+2\ell-1} - \frac{2}{n+2\ell+1} \right) = -\frac{2}{n+2m-1}.$$

For the norm of $\mathbf{b}_{2m}^\star$, we have

$$\|\mathbf{b}_{2m}^\star\|_2^2 = \left\langle \mathbf{b}_{2m} - \frac{2}{n+2m-1}\sum_{j=1}^{m}\mathbf{b}_{2m-2j}, \mathbf{b}_{2m} - \frac{2}{n+2m-1}\sum_{j=1}^{m}\mathbf{b}_{2m-2j} \right\rangle$$
$$= \|\mathbf{b}_{2m}\|_2^2 + \frac{4}{(n+2m-1)^2}\sum_{j=1}^{m}\|\mathbf{b}_{2m-2j}\|_2^2 - \frac{4}{n+2m-1}\sum_{j=1}^{m}\langle \mathbf{b}_{2m}, \mathbf{b}_{2m-2j}\rangle$$
$$+ \frac{8}{(n+2m-1)^2}\sum_{1\leqslant i<j\leqslant m}\langle \mathbf{b}_{2m-2i}, \mathbf{b}_{2m-2j}\rangle$$
$$= \frac{n+2}{2} + \frac{2(n+2)(m-1)+n+1}{(n+2m-1)^2} - \frac{4m-2}{n+2m-1} + \frac{4(m-1)^2}{(n+2m-1)^2} = \frac{n(n+2m+1)}{2(n+2m-1)}.$$

The norm of $\mathbf{b}_n^\star$ is also computed in the same manner, in the sense in the end we will have

$$\|\mathbf{b}_n^\star\|_2^2 = \frac{n}{2} + \frac{n(2n+1)}{2(2n-1)} = \frac{2n^2}{2n-1}.$$

□

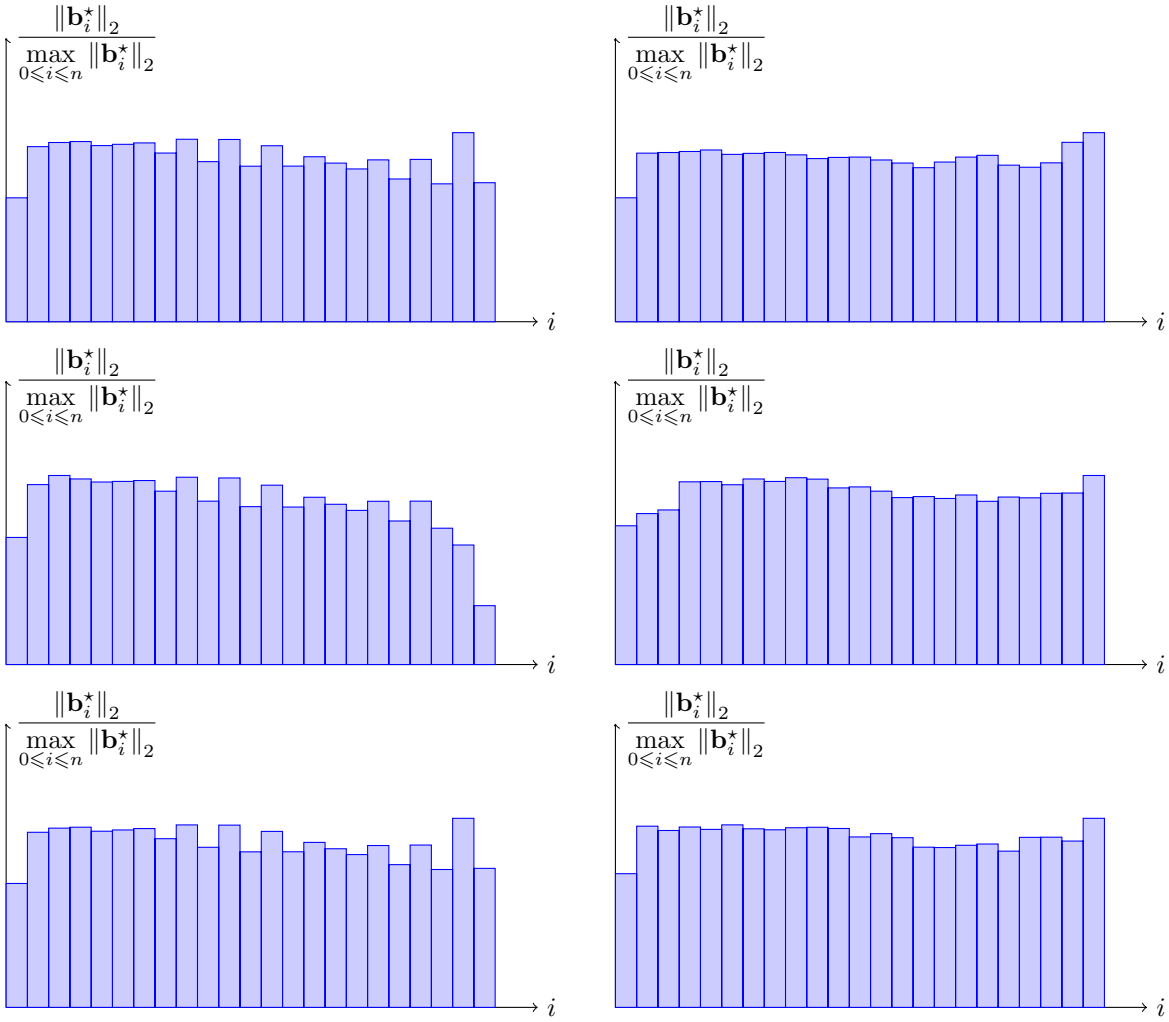

Figure 4.5: The relative lengths of successive Gram-Schmidt vectors before (left column) and after (right column) LLL-reduction, for Example 4.2. The top row uses the final reference set of $p^\star$, the middle row is for the zeros of $e^\star$ and endpoints of X, while the last row corresponds to an $n+2$-point AFP set.

Based on Lemma 4.20 and (4.15), one can easily check that $\mathbf{B}$ is already size-reduced. Inside the Lovász condition (4.12), we have that $\mu_{i+1,i} = 0, 0 \leqslant i < n$. From (4.16) and (4.17) it also follows that

$$\frac{\left\|\mathbf{b}_{i+1}^\star\right\|_2^2}{\|\mathbf{b}_i^\star\|_2^2} = \frac{(n+i+2)(n+i-1)}{(n+i)(n+i+1)} > \frac{1}{2}, 0 \leqslant i < n-1,$$

and

$$\frac{\|\mathbf{b}_n^\star\|_2^2}{\left\|\mathbf{b}_{n-1}^\star\right\|_2^2} = \frac{2n-1}{2(2n-2)} > \frac{1}{2}, 0 \leqslant i < n-1,$$

which means that $\mathbf{B}$ is a δ-LLL-reduced basis with $\delta > 0.5$. Even though the value of δ will be close to 0.5, the computational effort to reach a $\delta = 0.99$ LLL-reduced basis (the default value used in our tests) seems to be very modest in practice, as we shall see.

To a slightly lesser extent, this good behavior carries over to multi-interval domains as well, although we are not able to produce such a closed-form analysis as the one in Lemmas 4.20 and 4.21. A theoretically grounded intuition for why this is the case can nonetheless be given by taking into account how the greedy heuristic for choosing approximate Fekete points works. Recalling our discussion from Section 3.6.1, each basis vector corresponding to a new point is chosen such that the volume it generates with the previously selected vectors is as large as possible. Empirically, this suggests that the chosen basis vectors will be close to orthogonal to one another and the lengths of their Gram-Schmidt vectors will not vary by huge factors.

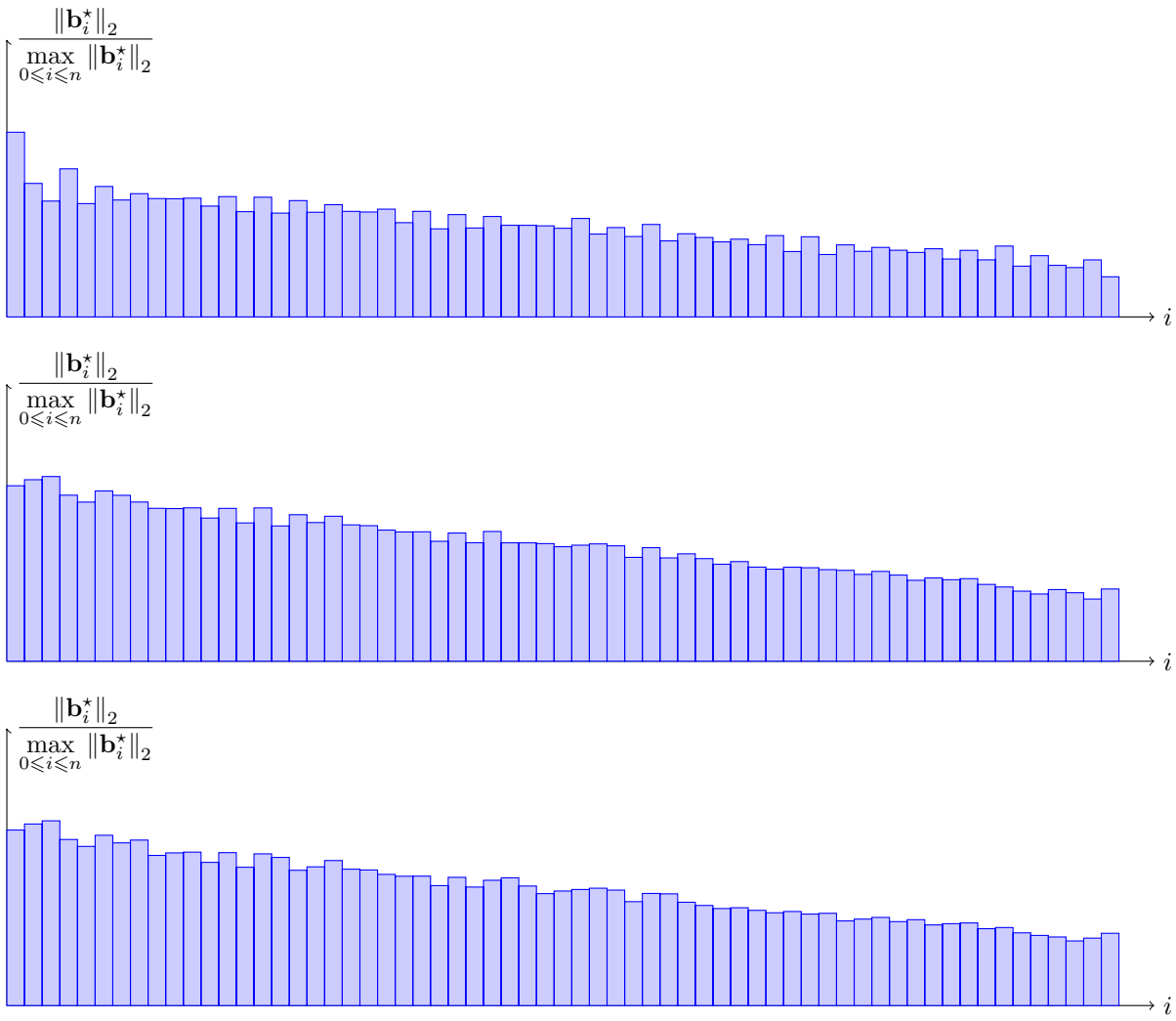

Figure 4.6: The relative lengths of successive Gram-Schmidt vectors for the design of a degree $n = 62$ type I filter adhering to specification B from Table 4.2, with 22-bit (21 bits for the fractional part) filter taps. The top row corresponds to the situation before any basis reduction algorithm was applied, whereas the middle row shows the effect of LLL reduction. BKZ (with block size $\beta = 8$) and HKZ reductions give the same result (bottom line). An AFP grid containing $n + 2$ elements was used.

Figure 4.5 shows this for Example 4.2. For all three discretization choices we advocated, the initial GSO vectors $\mathbf{b}_0^\star, \ldots, \mathbf{b}_{22}^\star$ do not vary in length very much, meaning we are starting with a well-behaved basis. For

the AFP-generated one in particular, we notice that $\mathbf{b}_0^\star$ is the shortest GSO vector, which, by Lemma 4.14, means that $\mathbf{b}_0$ is a solution for the corresponding ℓ_2 SVP problem. After performing LLL reduction in the right column, $\mathbf{b}_0^\star$ becomes the shortest GSO vector in all three cases.

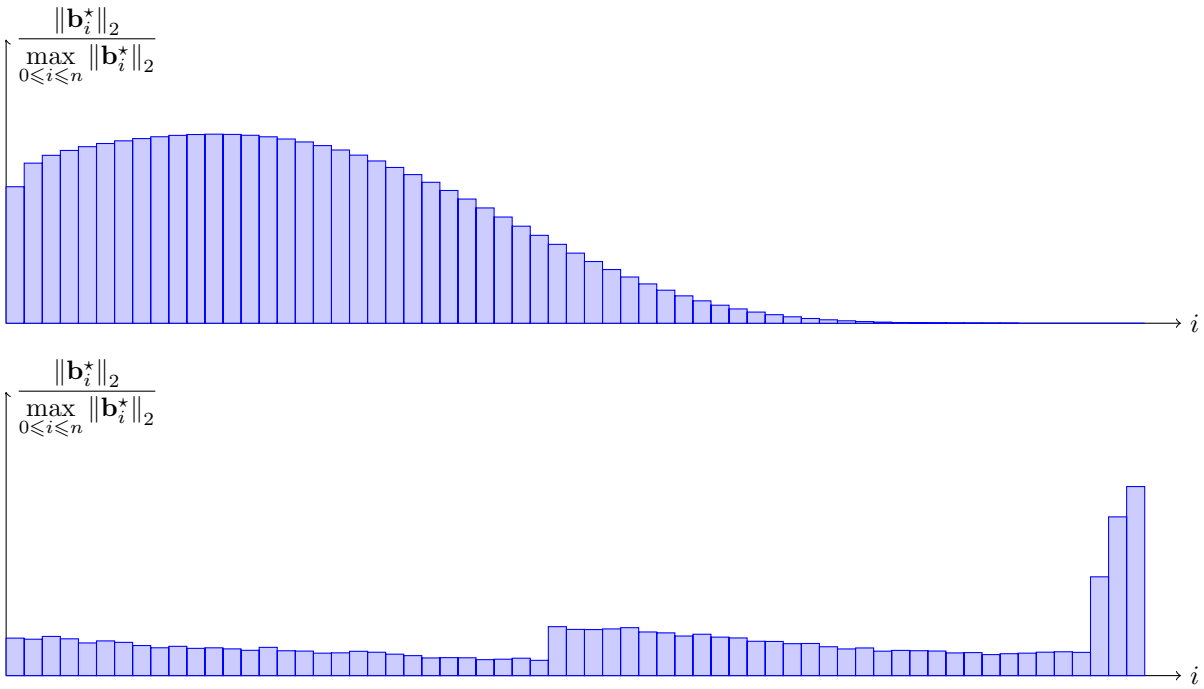

Figure 4.7: The relative lengths of successive Gram-Schmidt vectors for the design of a degree $n = 62$ type I filter adhering to specification B from Table 4.2, with 22-bit (21 bits for the fractional part) filter taps. The top row corresponds to the situation before any basis reduction algorithm was applied, whereas the bottom row shows the effect of LLL reduction. BKZ (with block size $\beta = 8$) and HKZ reductions give very similar results, which is why they are not shown. A grid of $n + 2$ equispaced points inside the convex hull of X was used.

Increasing n does not degrade by much the quality of the obtained bases. Figure 4.6 explores this for a degree $n = 62$ problem. There is a relatively small decrease in the lengths of successive $\mathbf{b}_i^\star$'s even before any reduction is applied and the rate of decrease after reduction seems only slightly slower than before. Also, and maybe more importantly, there are no essential differences in the quality of the obtained bases when different reduction algorithms are applied (middle and bottom rows). This tells us that, for our context, it is better to prefer LLL reduction due to its smaller execution times.

Remark 4.22. If we take a *bad* discretization for approximation/interpolation purposes, for instance $n + 2$ nodes which are equispaced inside the convex hull of X, then the initial lattice also seems to be of lesser quality. This is visible in Figure 4.7 for the same degree $n = 62$ specification as before. The dynamic range of the GSO vector lengths of the initial basis is much larger than in Figure 4.6 and considerable improvement is made after basis reduction. Still, we would not use such a basis in practice. Also, in this particular case, the time required to run the LLL algorithm is around 4 four times higher than if using one of the *good* discretization alternatives.

4.4.2 An approximate solution for the resulting CVP instance

Following our discussion on Kannan embedding in Section 4.3.4, our heuristic solution to (4.9) relies on the basis change

$$\underbrace{\begin{bmatrix} \mathbf{b}_0 & \mathbf{b}_1 & \cdots & \mathbf{b}_n & \mathbf{t} \\ 0 & 0 & \cdots & 0 & \gamma \end{bmatrix}}_{\mathbf{B_t}} \xrightarrow{\text{basis reduction}} \underbrace{\begin{bmatrix} \mathbf{b}'_0 & \mathbf{b}'_1 & \cdots & \mathbf{b}'_n & \mathbf{b}'_{n+1} \\ 0 & 0 & \cdots & 0 & \gamma' \end{bmatrix}}_{\mathbf{B}'_\mathbf{t}}, \tag{4.18}$$

where $\gamma = \max_{0 \leqslant i \leqslant n} \|\mathbf{b}_i\|_2$ and $\gamma' \in \{\pm\gamma\}$.

Generally, implementations of basis reduction algorithms like LLL, BKZ or HKZ can also provide the change of basis matrix $\mathbf{U} \in \mathbb{Z}^{(n+2)\times(n+2)}$ such that $\mathbf{B_t U} = \mathbf{B}'_\mathbf{t}$, without modifying the time complexity of the implementation. The a_i integer coefficients in (4.9) can then be taken as the first $n+1$ coefficients of the last column of $\mathbf{U}$ if $\gamma' = -\gamma$, or their sign swapped values if $\gamma' = \gamma$.

Refining the solution

The result of solving (4.9) gives a small $\|\sum_{i=0}^{n} a_i \mathbf{b}_i - \mathbf{t}\|_2$ which will usually be close to the optimal ℓ_2 CVP solution. Thinking optimistically, our heuristic should also give us a small ℓ_∞ error $\|\sum_{i=0}^{n} a_i \mathbf{b}_i - \mathbf{t}\|_\infty$ and, if the $(z_i)_{0 \leqslant i \leqslant m}$ discretization is well chosen, a close to optimal $\|\sum_{i=0}^{n} a_i \varphi_i - f\|_{\infty,X}$ value. Taking m to be as small as possible should also be preferred, since by the norm inequality in $\mathbb{R}^{m+1}$, we have $\|\cdot\|_\infty \leqslant \|\cdot\|_2 \leqslant \sqrt{m+1}\,\|\cdot\|_\infty$.

Because the $\mathbf{b}'_i$ vectors in (4.18) are usually short with respect to the ℓ_2 norm (and consequently, the ℓ_∞ one as well) we can use them to search in the vicinity of $\sum_{i=0}^{n} a_i \mathbf{b}_i$ to improve our L_∞ target solution. In the function approximation setting, a similar idea is described in [58, p. 128].

Let $\mathbf{x} = \sum_{i=0}^{n} a_i \mathbf{b}_i$. We then perform the following:

(1) Let $\mathbf{x}'$ successively denote the vectors $\mathbf{x} + \varepsilon_i \mathbf{b}'_i + \varepsilon_j \mathbf{b}'_j$, where $\varepsilon_i, \varepsilon_j \in \{0, \pm 1\}, t_1, t_2 \in \mathbb{N}, 0 < t_1 < t_2 \leqslant n$ and $0 \leqslant i \leqslant t_1 < j \leqslant t_2$. The values of t_1 and t_2 can be chosen by the user.

(2) Express each $\mathbf{x}'$ as $\sum_{i=0}^{n} a'_i \mathbf{b}_i$ (this is easy to do since each $\mathbf{b}'_i$ can be expressed in terms of $\mathbf{b}_0, \ldots, \mathbf{b}_n$ by taking the corresponding column of $\mathbf{U}$) and pick one for which the resulting a_i's satisfy the format constraints and

$$\left\| \sum_{i=0}^{n} a'_i \varphi_i - f \right\|_{\infty,X} \tag{4.19}$$

is minimal.

In [58], the (4.19) error is replaced with a ℓ_∞ one that uses the discretization nodes $(z_i)_{0 \leqslant i \leqslant m}$. While this ensures a faster execution (the resulting ℓ_∞ error computations are much less expensive than the L_∞ ones), the quality of the results obtained by using (4.19) should overall be better. The value of t_1 should be very small (say $t_1 \leqslant 2$) if the computational cost of the vicinity search is to be kept reasonable. In our tests, increasing t_1 beyond 4 did not improve the final output and only incurred a significant execution overhead.

4.4.3 A theoretical estimate of the quality of our solution

Let p be an approximation returned by our algorithm and p_{opt} an optimal solution. We set $m = n$. Let $\{\ell_k(x)\}_{0 \leqslant k \leqslant n}$ be the Lagrange basis functions associated to $S_n = \mathrm{span}_\mathbb{R}(\varphi_i)_{0 \leqslant i \leqslant n}$ and $\mathbf{z} := \{z_0, \ldots, z_n\}$. We denote by $\Lambda_{S_n,X}(\mathbf{z})$ the Lebesgue constant associated to $\mathbf{z}$ and X (see Section 3.6.1 for the corresponding definitions).

Let us consider $\delta = \max_{0 \leqslant k \leqslant n} |p(z_k) - f(z_k)|$, the ℓ_∞ error yielded by our approximate solution to the CVP stated in (4.10). For all $x \in X$,

$$p(x) - p_{opt}(x) = \sum_{i=0}^{n} (p(z_i) - p_{opt}(z_i)) \ell_i(x),$$

hence, for all $x \in X$, we have, from (3.8),

$$\begin{aligned} |p(x) - p_{opt}(x)| &\leqslant \Lambda_{S_n,X}(\mathbf{z}) \max_{0 \leqslant k \leqslant n} |p(z_k) - p_{opt}(z_k)| \\ &\leqslant \Lambda_{S_n,X}(\mathbf{z}) \max_{0 \leqslant k \leqslant n} (|p(z_k) - f(z_k)| + |f(z_k) - p_{opt}(z_k)|) \\ &\leqslant \Lambda_{S_n,X}(\mathbf{z})(\delta + ||f - p_{opt}||_{\infty,X}). \end{aligned}$$

Now, if we remember that $||f - p||_{\infty,X} - ||f - p_{opt}||_{\infty,X} \leqslant ||p - p_{opt}||_{\infty,X}$, it follows that

$$||f - p||_{\infty,X} \leqslant (1 + \Lambda_{\mathbb{R}_n[x],X}(\mathbf{z}))||f - p_{opt}||_{\infty,X} + \Lambda_{\mathbb{R}_n[x],X}(\mathbf{z})\delta.$$

This inequality tells us that the smaller δ and the Lebesgue constant, the better the approximation quality of the returned polynomial p.

4.5 Experimental results

The numerical computations in this section were performed using the multiple precision version of the exchange algorithm discussed in the previous chapter and the lattice basis reduction algorithm implementations from the FPLLL library [204] (also their multiple precision versions). The same machine that performed the computations from Section 4.2 is used here also.

To show the capabilities of our method, we will compare it to the telescoping rounding approach introduced in [124] and the MATLAB DSP Toolbox implementation of the error shaping algorithm from [157], since both these methods address the same coefficient format context as us. We use all three discretization approaches mentioned at the start of Section 4.4.1 and take the best result. On some of the examples, we will use the passband ripple $20 \log_{10} \left(\frac{1+\delta_p}{1-\delta_p} \right)$ and stopband attenuation $-20 \log_{10} \delta_s$ to measure the quality of our outputs, where δ_p and δ_s represent the unweighted approximation errors on the passband and stopband, respectively.

We will start with the examples in [124, Table 2]. In terms of design instance codes of the form A35/8, the letter designates the specification from Table 4.2, the second value is the filter length N from (2.10) (meaning a degree $n = 17$ approximation), whereas the third value is the number of bits used to store the resulting filter taps, which in this case corresponds to $b = 8$ bits. Out of all these bits, one is used to store the sign of the taps, whereas the remaining $b - 1$ are used to store their fractional part. In the context of Problem 4.1, this translates to $s = 2^{b-1}$ and $[m_-, m_+] = [-2^{b-1}, 2^{b-1}]$.

Let us first compare the quality of the obtained results between the three different discretization choices (the error from (4.19) after we perform the vicinity search). They are explored in Table 4.4. The first column designates the problem instance, while the next three columns represent the approximation errors at the end of executing our routine for each choice. The fourth column is the average real time (over the three discretization choices) required by the LLL algorithm to work on each corresponding basis $\mathbf{B_t}$ inside (4.18). Similarly, the fifth column gives the average runtime of the vicinity search after $\mathbf{B'_t}$ and $\mathbf{U}$ are computed. The last column gives the total runtime and consists of the execution of all three steps of our heuristic for all three discretization choices. In particular, the 7.1 seconds required for A45/8 (the specification for Example 4.2) is higher than the 5.83 seconds needed by the double-precision MILP solver. This is mainly due to the use of multiple precision in our code (a default value of 200 bits of precision in our `MPFR`-based routines).

We used multiple precision because it provides a safer context for performing basis reduction, but for speed considerations, both the minimax computation using the Parks-McClellan algorithm and the vicinity search should behave in double-precision arithmetic without any numerical issues. Switching to double-precision in those contexts should decrease the total runtime by at least an order of magnitude. We are currently investigating precisely which parts of the implementation can be done in double precision arithmetic, without any significant accuracy impact towards the final result.

Nevertheless, since we are dealing with offline computations, a runtime of less than a minute for the larger degree problems should not be a limitation for our method (MILP approaches seem to require significantly

Table 4.4: Approximation error comparison for the three different discretization choices. LLL basis reduction is used. We have taken $t_1 = 2, t_2 = n$ for $n < 40$ and $t_1 = 1, t_2 = \lfloor n/2 \rfloor$ for $n \geqslant 40$ to limit the vicinity search runtime.

Filter	Extrema error	Zeros + band edges error	AFP error	LLL runtime	Vicinity search runtime	Total runtime
A35/8	0.032645	**0.030013**	0.040432	0.00065	1.03	3.4
A45/8	0.031887	**0.030556**	0.030864	0.00094	2.22	7.1
A125/21	$1.25956 \cdot 10^{-5}$	$\mathbf{1.25384 \cdot 10^{-5}}$	$1.28772 \cdot 10^{-5}$	0.02229	17.01	53
B35/9	0.095081	**0.091226**	0.095081	0.00061	1.03	3.5
B45/9	0.064344	0.073617	**0.061486**	0.00121	2.25	7.4
B125/22	$3.33786 \cdot 10^{-5}$	$3.45306 \cdot 10^{-5}$	$\mathbf{3.24349 \cdot 10^{-5}}$	0.02235	17.1	53.8
C35/8	0.021751	0.019169	**0.017871**	0.00064	0.79	3
C45/8	0.023437	0.020912	**0.016096**	0.00176	1.7	5.8
C125/21	$\mathbf{1.62125 \cdot 10^{-6}}$	$1.93359 \cdot 10^{-6}$	$2.21053 \cdot 10^{-6}$	0.07971	12.9	41.4
D35/9	0.040518	0.038283	**0.032914**	0.00092	0.78	2.8
D45/9	**0.027056**	0.040024	0.027148	0.00167	1.7	5.7
D125/22	$2.10324 \cdot 10^{-6}$	$\mathbf{2.00989 \cdot 10^{-6}}$	$2.25242 \cdot 10^{-6}$	0.07715	12.8	41.5
E35/8	0.039796	**0.034046**	0.040618	0.00044	0.98	3.3
E45/8	0.032895	**0.029713**	0.037960	0.00083	2.1	6.9
E125/21	$1.2857 \cdot 10^{-5}$	$1.28243 \cdot 10^{-5}$	$\mathbf{1.24587 \cdot 10^{-5}}$	0.01719	16.5	51.8

more time for those instances). In terms of error values, it becomes clear from these 15 examples that none of the three discretization choices is universally better than the others, although all of them seem to provide pretty similar results. This is why we work with all of them and at the end we pick the best result.

Table 4.5: Approximation error and execution times for LLL, BKZ and HKZ reduction.

Filter	LLL error	BKZ error	HKZ error	LLL runtime	BKZ runtime	HKZ runtime
A35/8	**0.030013**	**0.030013**	**0.030013**	0.00065	0.00081	0.00089
A45/8	**0.030556**	**0.030556**	**0.030556**	0.00094	0.00156	0.00166
A125/21	$1.25384 \cdot 10^{-5}$	$1.25168 \cdot 10^{-5}$	$\mathbf{1.22264 \cdot 10^{-5}}$	0.02229	0.03418	0.07814
B35/9	**0.091226**	0.095081	0.095081	0.00061	0.00108	0.00105
B45/9	**0.061486**	0.0639243	0.065839	0.00121	0.00171	0.00211
B125/22	$\mathbf{3.24349 \cdot 10^{-5}}$	$3.3449 \cdot 10^{-5}$	$3.3449 \cdot 10^{-5}$	0.02235	0.03242	0.21234
C35/8	**0.017871**	0.019169	0.019169	0.00064	0.00123	0.00116
C45/8	**0.016096**	**0.016096**	0.021371	0.00176	0.00331	0.00318
C125/21	$1.62125 \cdot 10^{-6}$	$1.57285 \cdot 10^{-6}$	$\mathbf{1.56468 \cdot 10^{-6}}$	0.07971	0.15751	118
D35/9	**0.032914**	0.0358513	0.0358513	0.00092	0.00186	0.00124
D45/9	**0.27056**	0.0280512	**0.027056**	0.00167	0.00321	0.00331
D125/22	$2.00989 \cdot 10^{-6}$	$\mathbf{1.77774 \cdot 10^{-6}}$	$1.94256 \cdot 10^{-6}$	0.07715	0.18441	115
E35/8	**0.034046**	**0.034046**	**0.034046**	0.00044	0.00068	0.00078
E45/8	**0.029713**	0.0301061	0.032895	0.00083	0.00128	0.00187
E125/21	$1.24587 \cdot 10^{-5}$	$1.24587 \cdot 10^{-5}$	$\mathbf{1.21758 \cdot 10^{-5}}$	0.01719	0.04123	0.867

Table 4.5 offers information about the use of LLL, BKZ and HKZ reduction. The results in columns 2 to 4 represent the best values after all three discretization choices were tried, while the last three columns show the average run times of the basis reduction algorithms on $B_{\mathbf{t}}$. We remark that the more expensive algorithms like BKZ and HKZ do not seem to offer many improvements over LLL in our context. This means that LLL already performs a very good job, matching our analysis from Section 4.4.1. HKZ-reducing a random lattice of dimension greater than 60 is, as of today, a considerable task. The fact that HKZ managed to finish in less than two minutes for the degree $n = 62$ instances considered here is further experimental proof of the good basis reduction properties of the lattices appearing in our problems.

Table 4.6: Comparison with optimal results and the heuristic approach from [124].

Filter	Minimax δ_X	Optimal δ_{opt}	Naive rounding δ_{naive}	Telescoping δ_{tel}	Lattice reduction δ_{lattice}	
A35/8	0.01595	0.02983	0.03266	*0.03266*	**0.030013**	LLL
A45/8	$7.132 \cdot 10^{-3}$	0.02962	0.03701	*0.03186*	**0.030556**	LLL
A125/21†	$8.054 \cdot 10^{-6}$	$\mathit{1.077 \cdot 10^{-5}}$	$1.620 \cdot 10^{-5}$	$\boldsymbol{1.179 \cdot 10^{-5}}$	$1.222 \cdot 10^{-5}$	HKZ
B35/9	0.05275	0.07709	0.15879	***0.07854***	0.09122	LLL
B45/9	0.02111	0.05679	0.11719	*0.06641*	**0.06148**	LLL
B125/22†	$2.498 \cdot 10^{-5}$	$\mathit{2.959 \cdot 10^{-5}}$	$6.198 \cdot 10^{-5}$	$\mathit{3.293 \cdot 10^{-5}}$	$\mathbf{3.243 \cdot 10^{-5}}$	LLL
C35/8	$2.631 \cdot 10^{-3}$	0.01787	0.04687	***0.01787***	**0.01787**	LLL
C45/8	$6.709 \cdot 10^{-4}$	0.01609	0.03046	*0.02103*	**0.01609**	LLL
C125/21‡	$1.278 \cdot 10^{-8}$	$1.564 \cdot 10^{-6}$	$8.203 \cdot 10^{-6}$	$\mathit{2.1 \cdot 10^{-6}}$	$\mathbf{1.564 \cdot 10^{-6}}$	HKZ
D35/9	0.01043	0.03252	0.12189	*0.03368*	**0.0329**	LLL
D45/9	$2.239 \cdot 10^{-3}$	0.02612	0.10898	*0.02859*	**0.02705**	LLL
D125/22‡	$4.142 \cdot 10^{-8}$	$1.777 \cdot 10^{-6}$	$3.425 \cdot 10^{-5}$	$\mathit{2.16 \cdot 10^{-6}}$	$\mathbf{1.777 \cdot 10^{-6}}$	BKZ
E35/8	0.01761	0.03299	0.04692	***0.03404***	**0.03404**	LLL
E45/8	$6.543 \cdot 10^{-3}$	0.02887	0.03571	*0.03403*	**0.02971**	LLL
E125/21†	$7.884 \cdot 10^{-6}$	$\mathit{1.034 \cdot 10^{-5}}$	$1.479 \cdot 10^{-5}$	$\boldsymbol{1.127 \cdot 10^{-5}}$	$1.217 \cdot 10^{-5}$	HKZ

Inside Table 4.6, the second column gives the real-valued coefficient minimax error computed using the Parks-McClellan algorithm. All remaining columns represent approximation errors of the transfer function for various finite precision coefficient strategies. The third column errors are *optimal* in the sense that most of them (*i.e.,* those corresponding to the problems without a † or ‡ tag) are obtained using an MILP solver. Problems marked with † (of degree $n = 62$) are those where the exact solver was stopped after a certain time limit and the result given is the best one found so far. The two specifications marked with ‡ also have time-limited MILP values in [124], but during our lattice-based computations we were able to find quantizations with smaller errors (in less time), which are given here.

Column four lists the errors obtained by simply rounding the real-valued minimax coefficients to their closest values in the imposed formats, whereas the fifth column gives the best errors from using the coefficient telescoping rounding approach of [124, Sec. 4]. That article actually describes two versions of the method: a one coefficient version and a two coefficient one. Without going into the details, the one coefficient telescoping method works by iteratively rounding the coefficients, one at a time, starting from a_n and finishing at a_0. At each step, the coefficients which have not yet been discretized are updated in such a way that the approximation error due to this progressive quantization does not grow too large. The two coefficient version is a more complicated variation of this idea. Since we were not able to reproduce many of the results obtained with these methods using our tools, these values are reported here in italic (and represent the best values between the one and two coefficient versions).

The last column gives the best results we obtained with our lattice-based approach, together with the lattice reduction algorithm that generated it. In particular, we remark that the LLL-based results in 12 out of the 15 specifications are on par or better than the telescoping rounding approach. They are also very close to the optimal ones.

Adaptive weighting

Consider now a lowpass filter specification similar to that of [157, Sec. 5]. We want to design a degree $n = 34$ type I filter with passband $[0, 0.4\pi]$ and stopband $[0.6\pi, \pi]$ in such a way that all coefficients are stored using 16 bits of precision and the stopband attenuation of the final design is maximized. Inside MATLAB, we can first construct a real-valued coefficient filter and then use the error shaping algorithm from [157]:

```
d = fdesign.lowpass('N,Fp,Fst,Ast',68,0.4,0.6,120);
Hd = design(d,'equiripple');
Hq = maximizestopband(Hd,16,'Ntrials',100);
```

The first two lines design an equiripple filter with a 120 dB target stopband attenuation using the Parks-McClellan algorithm, whereas the last one performs the actual coefficient quantization by trying to maximize the quality of the approximation on the stopband. Since the `maximizestopband` routine is stochastic, we execute it 100 times and take the best result, which has a 101.54 dB attenuation on the stopband.

We can use our heuristic in a similar way to optimize the approximation error more on certain bands (without changing the scaling factor s). This is done by changing the CVP problem (4.9) into

$$\sum_{i=0}^{n} a_i \underbrace{\begin{bmatrix} \alpha_0\varphi_i(z_0) \\ \alpha_1\varphi_i(z_1) \\ \vdots \\ \alpha_m\varphi_i(z_m) \end{bmatrix}}_{\mathbf{b}_i} \simeq \underbrace{\begin{bmatrix} \alpha_0 p^\star(z_0) \\ \alpha_1 p^\star(z_1) \\ \vdots \\ \alpha_m p^\star(z_m) \end{bmatrix}}_{\mathbf{t}},$$

where $\alpha_0, \ldots, \alpha_m \in \mathbb{R}_+$. By default, we have $\alpha_0 = \ldots = \alpha_m = 1$, but by slightly increasing the α_i values corresponding only to the z_i nodes located in a certain band, that should decrease the approximation error of our result on that particular band. Typical values we considered are $\alpha_i \in \{1.1, 1.2, 1.3, 1.4, 1.5\}$.

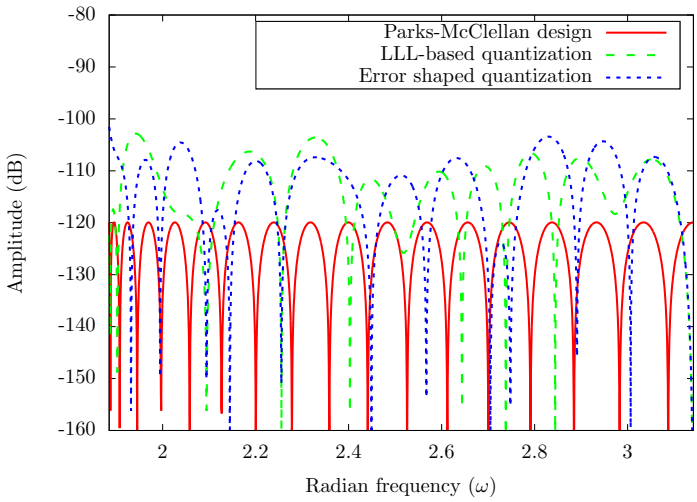

Figure 4.8: Stopband attenuation for the minimax design of a given lowpass specification, alongside the 16-bit fixed-point designs of the error shaped approach and our LLL-based algorithm.

Used in this way, the LLL basis reduction approach on the same starting minimax design gives us a larger 102.83 dB stopband attenuation. These responses are also shown in Figure 4.8. The cost of having a better stopband behavior for our LLL-designed filter is a larger passband ripple (0.046 dB as opposed to 0.013 dB for the error shaping design).

Table 4.7: Adaptive weighting improvements on the examples in Table 4.6.

Filter	Minimax δ_X	Optimal δ_{opt}	Naive rounding δ_{naive}	Telescoping δ_{tel}	LLL reduction δ_{LLL}
A125/21†	$8.054 \cdot 10^{-6}$	*$1.077 \cdot 10^{-5}$*	$1.620 \cdot 10^{-5}$	*$1.179 \cdot 10^{-5}$*	**$\mathbf{1.155 \cdot 10^{-5}}$**
B35/9	0.05275	0.07709	0.15879	***0.07854***	**0.07854**
E125/21†	$7.884 \cdot 10^{-6}$	*$1.034 \cdot 10^{-5}$*	$1.479 \cdot 10^{-5}$	***$\mathbf{1.127 \cdot 10^{-5}}$***	$1.133 \cdot 10^{-5}$

This *adaptive weighting* strategy can also be used on the four examples from Table 4.6 that did not produce the best results. One such case is the lowpass design A125/21. When we used uniform unit weighting

for all the interpolation nodes, the initial LLL quantization error was $1.397 \cdot 10^{-5}$. This value represented the passband error, while the stopband error was a slightly smaller $1.369 \cdot 10^{-5}$. Running the LLL reduction algorithm with a new passband weight of 1.5 reduces the overall starting error to $1.216 \cdot 10^{-5}$. This new value corresponds to the stopband, while for the passband we got $1.124 \cdot 10^{-5}$. Further improving this initial guess with the local search using short lattice vectors decreased the error to what is given in Table 4.7. For the time being, we are not able to fully automate this refinement process. Nevertheless, a filter designer can very quickly experiment with different weighting possibilities and pick the best one. More examples are given in the test file accompanying the code.

Higher degree examples

Table 4.8: High degree type I FIR specifications.

Filter	Bands	$D(\omega)$	$W(\omega)$
F	$[0, 0.2\pi]$	0	10
	$[0.205\pi, \pi]$	1	1
G	$[0, 0.4\pi]$	1	1
	$[0.405\pi, 0.7\pi]$	0	10
	$[0.705\pi, \pi]$	1	1
H	$[0, 0.45\pi]$	0	10
	$[0.47\pi, \pi]$	1	1
I	$[0, 0.2\pi]$	0	10
	$[0.21\pi, \pi]$	1	1
J	$[0, 0.25\pi]$	0	10
	$[0.27\pi, 0.7\pi]$	1	1
	$[0.72\pi, \pi]$	0	10
K	$[0, 0.25\pi]$	0	10
	$[0.3\pi, 0.7\pi]$	1	1
	$[0.75\pi, \pi]$	0	10

Although filters with degree $n > 100$ having small bit size fixed-point coefficients (the largest degree we have considered so far in this section is $n = 62$) do not seem to be common in practice, we remark that during our tests we were able to handle degrees of up to $n = 800$ in less than 10 minutes. Some results are shown in Table 4.9, which adhere to the specifications from Table 4.8.

Table 4.9: High degree quantization results.

Filter	Degree n	Coefficient bit size b	Minimax $E^\star$	Naive rounding E_{naive}	LLL reduction E_{LLL}
F	800	18	$8.64 \cdot 10^{-4}$	$3.31 \cdot 10^{-3}$	$\mathbf{1.24 \cdot 10^{-3}}$
G	600	16	$4.78 \cdot 10^{-3}$	$1.24 \cdot 10^{-2}$	$\mathbf{6.73 \cdot 10^{-3}}$
H	250	16	$1.67 \cdot 10^{-4}$	$5.79 \cdot 10^{-3}$	$\mathbf{1.71 \cdot 10^{-3}}$
I	300	18	$4.67 \cdot 10^{-3}$	$6.56 \cdot 10^{-3}$	$\mathbf{4.91 \cdot 10^{-3}}$
J	180	16	$1.77 \cdot 10^{-3}$	$6.19 \cdot 10^{-3}$	$\mathbf{3.11 \cdot 10^{-3}}$
K	150	21	$2.99 \cdot 10^{-6}$	$7.68 \cdot 10^{-5}$	$\mathbf{2.15 \cdot 10^{-5}}$

4.6 Conclusion and future work

In this chapter we have presented a novel approach for uniform coefficient quantization of FIR filters based on an idea previously introduced in [38] enhanced by the introduction of an efficient discretization of the approximation domain, which transforms the design problem using the language of Euclidean lattices.

The values obtained show that our method is very robust and competitive in practice. It frequently produces results which are close to optimal and/or beats other heuristic approaches. A current endeavor is automatizing the weight changes when trying to improve the quantization quality.

It might be interesting to try and address IIR coefficient quantization problems using Euclidean lattices. There are several difficulties which have to be overcome in the rational IIR setting:

- nonlinearity of the transfer function;
- the approximation domain switches from X to a subset of the unit circle;
- ensure stability of the transfer function (control position of the poles).

5 A complete design chain for FIR filter synthesis on FPGAs

With four parameters I can fit an elephant, and with five I can make him wiggle his trunk.
J. von Neumann

In what follows, we propose an open-source automatic FIR filter synthesis tool[1] targeting FPGAs. It is the result of a collaboration with N. Brisebarre, F. de Dinechin and M. Istoan [39]. User intervention is limited to a very small number of relevant input parameters (a high-level frequency-domain specification and input/output formats) without any sacrifices in terms of design flexibility. All the other design parameters are computed automatically, using the approaches discussed in Chapters 3–4 and the direct-form architecture implementation methodology presented in [71]. Our computations assure that the resulting architecture respects the specification, while attempting to keep the hardware use cost low. A comparison with other mainstream design tools is also performed.

5.1 Introduction

FPGAs (Field Programmable Gate Arrays) are maybe the most prominent examples of what is called *reconfigurable hardware* (*i.e.,* during its lifetime, the logic behavior of the hardware device can be modified by the user and is not fixed during manufacturing, like for the microprocessors we find in our computers). With FPGAs, users can program the behavior of their circuit at the logic gate level using hardware description languages like `VHDL` or `Verilog`.

5.1.1 Why FPGAs?

Since their appearance in the 1980s, FPGAs have continued to evolve at a steady pace. Today they are being used in a myriad of domains such as automotive, medical and aerospace, just to name a few. Modern offerings from major vendors like Altera[2] and Xilinx[3] are in fact a mix of several building blocks: configurable memory elements, high-speed input/output pins, configurable logic blocks (CLBs) composed of multiple smaller look-up tables (LUTs) and programmable routing channels (linked through switch matrices).

These programmable logic blocks are usually structured as a large bidimensional array (see Figure 5.1). Such a structure and the huge duplicity in available resources hint at the massive parallelism FPGAs can achieve. Hardware intellectual property (IP) cores like digital signal processors are also included on FPGAs because of their usefulness in many practical scenarios (such as those requiring FIR filters).

[1] see http://perso.ens-lyon.fr/silviuioan.filip/icassp.html for the accompanying C++ source code
[2] https://www.altera.com/
[3] http://www.xilinx.com/

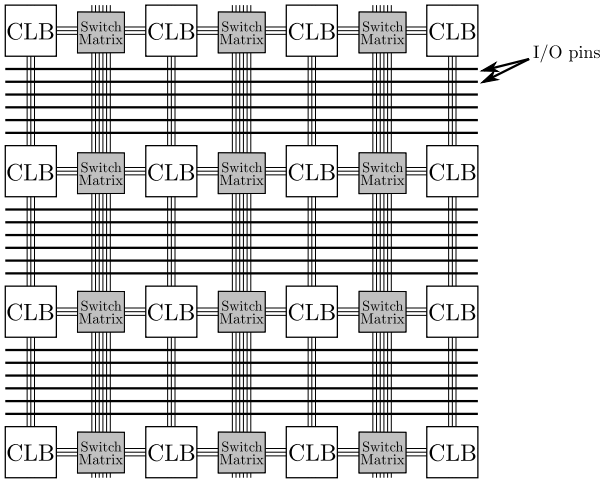

Figure 5.1: Simplified depiction of a generic FPGA layout.

To quantify the cost and performance of an FPGA-bound circuit we generally have to look at the resource consumption in terms of memory elements (measured in registers) and programmable logic (measured in terms of LUTs) coupled with its speed/throughput characteristics. In the majority of cases, there is a trade-off to be made between the two (*i.e.,* a faster circuit uses more resources). If we couple this aspect with the reasonable run times required to synthesize a logic design written in a language like `VHDL`, there is a lot of *flexibility* to be had, in particular for digital signal processing tasks like FIR filtering [150].

5.1.2 Digital filters: from specification to implementation

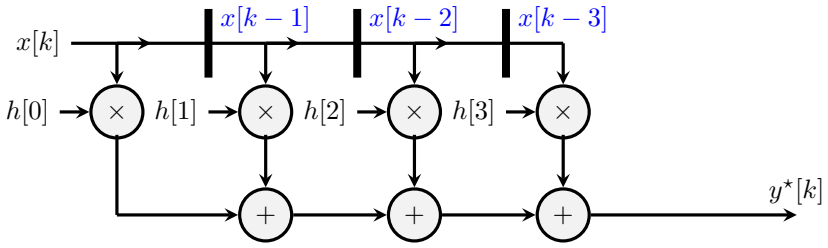

Figure 5.2: An ideal FIR filter architecture.

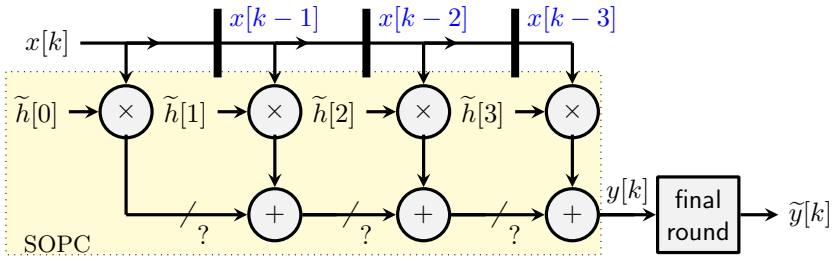

Figure 5.3: An actual FIR filter architecture.

We reiterate the fact that a digital filter is usually specified in the frequency domain (FD) by its frequency response, an example of which is provided in Figure 3.2. On that particular example, the specification

consists of the passband frequency ω_p, stopband frequency ω_s, and the bounds δ_p and δ_s on the passband ripple and stopband attenuation. An actual FIR filter implementation, however, computes sums of products in the time domain (TD) (see eqs. (5.2) and (5.3)). Ideally, such an implementation would look something like Figure 5.2, but due to finite-precision errors, something more akin to Figure 5.3 will in fact be implemented.

To obtain such an implementation of a FIR filter out of its FD specification, a three-step process is usually applied:

Step 1: the focus of Chapter 3. Determine the filter length N and the real coefficients $\{h[k]\}_{k=0}^{N-1}$ of an appropriate frequency response

$$H(\omega) = \sum_{k=0}^{N-1} h[k]e^{-ik\omega}, \quad \omega \in [0, 2\pi). \tag{5.1}$$

Normally, we want to implement

$$y^\star[k] = \sum_{i=0}^{N-1} h[i]x[k-i]. \tag{5.2}$$

Step 2: the focus of Chapter 4. As stated before, we are usually limited in what we can do with (5.2), so we quantize each $h[k]$ coefficient to a machine-representable (*e.g.,* fixed-point) number $\widetilde{h}[k]$.

Step 3: the focus of this chapter. Implement the time domain computation:

$$y[k] = \sum_{i=0}^{N-1} \widetilde{h}[i]x[k-i], \tag{5.3}$$

where the $x[i]$ are the fixed-point input signal values. Here (5.3) is again an ideal description of the computation, but due to limitations on the output format, the value computed by the implementation will be some $\widetilde{y}[k]$ (Figure 5.3).

5.1.3 Implementation parameter space

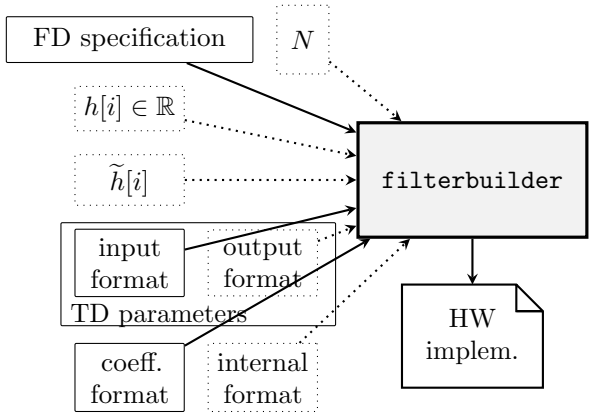

Figure 5.4: MATLAB design flow with all the parameters. The dashed ones can be computed by the tool or input by the user. The external `minimizecoeffwl` function can help compute the coefficient formats, while the quantized coefficients themselves can be computed using one of `minimizecoeffwl`, `constraincoeffwl` or `maximizestopband`.

Each of these three steps involves several parameters. The user should only need to provide the values describing the desired FD behavior (*e.g.,* frequency response bands and error masks) and the I/O formats of the implementation, letting the tool take care of the rest in an efficient manner.

Functional, FD parameters and constraints control the first step. The second and third steps depend on architectural, time domain parameters and constraints: the fixed-point format of $x[k]$ (or input format); the

format of $y[k]$ (output format); the format of the intermediate products and sums, represented by a ? on Figure 5.3. The filter order N is a parameter shared by all the steps.

The main challenge for a filter designer is that architectural choices made in the time domain obviously also impact the quality of a filter, as specified in the frequency domain. For instance, however clever the filter design may be, the quantization noise added by the I/O formats will affect the filter frequency response. Conversely, the choice of N will also limit the quality of the filter, however large the I/O formats.

To address this difficulty, current mainstream design flows tend to expose all these parameters (including the internal formats that normally should be optimized by the tool). Figure 5.4 shows the parameter interface to MATLAB's `filterbuilder` tool included in its Signal Processing Toolbox, which goes from a FD specification of the problem to a synthesizable (`VHDL` or `Verilog`) hardware description.

5.1.4 Contributions and outline of the present work

Our main goal is to propose a design tool where near-optimal values of most implementation parameters can be deduced, in an automatic way, from 1) the FD filter specification and 2) the desired input/output formats of the signals. A nice side effect is a simplified user experience, since the user is required to provide only these parameters.

The method is automated and based on error analysis performed both in the frequency and time domains. It builds upon recent open-source efforts that address each of the three design steps with guarantees of numerical quality: equiripple designed filters using the Parks-McClellan algorithm (see Ch. 3 or [82]) for step 1, Euclidean lattice-based coefficient quantization (see Ch. 4 or [41]) for step 2, and sum of products with constants (SOPC) operators [71] for step 3, available in the FloPoCo[4] arithmetic operator generator.

FloPoCo (the name comes from **Flo**ating-**Po**int **Co**res) is an open-source tool written in C++ which generates synthesizable `VHDL` hardware descriptions of arithmetic operators (*e.g.,* multipliers, polynomial evaluators, dot products, etc.). It is largely targeted towards FPGAs and is thus a good choice for the present work.

The robustness results from Ch. 3 make it possible to address a broad class of FIR filters. Our quantization process from Ch. 4 usually gives near-optimal quantized coefficients, whereas MATLAB, for instance, uses by default a randomized process whose output quality is variable. Finally, the architecture from [71] offers a guarantee on the accuracy of the evaluation of the signal in the time domain.

This integration should be done with care because of the mutual impact of FD and TD parameters on the filter quality. We will discuss this interaction in Section 5.3. Section 5.4 shows that our tool leads to efficient architectures when compared to MATLAB-generated ones. Before that, Section 5.2 will survey relevant previous works on the subject.

5.2 Background and state of the art

If we focus on solutions that are able to generate hardware FIR filters starting from their FD specification, at the moment of writing this text, the most common choice is very probably MATLAB's Signal Processing toolbox. When working with FPGAs, the mainstream solution is to use vendor tools (from Xilinx, Altera or Actel). They provide FIR IP cores that mainly address step 3, relying on MATLAB/Simulink for coefficient generation and simulation [105]. This justifies why we only compare to MATLAB in Section 5.4.

Open-source alternatives like GNURadio[5] or the Scipy `signal` package[6] essentially address only step 1. It is hoped that our present efforts could establish a missing link to FPGA back-end work [140, 142].

There have been numerous academic works targeting FPGAs or application-specific integrated circuits (ASICs) before the MATLAB hegemony. Unfortunately, none of these efforts (to our knowledge) describes an open-source tool. Among these, [233], [108], [110] address all three steps, but without explicit error analysis and no guarantees on the output quality: the two first steps are validated iteratively by simulation. The same holds for some recent work [194] whose main originality is to perform all computations using

[4] http://flopoco.gforge.inria.fr
[5] http://gnuradio.org
[6] http://docs.scipy.org/doc/scipy/reference/signal.html

the Residue Number System (see for instance [202]). The work in [52] is based on an error analysis that optimizes the formats subject to a prescribed output noise power.

Our method is novel in two respects. Firstly, all previous approaches rely on approximate double-precision computations and/or simulation to evaluate the quality of steps 1 and 2. By using the code from Ch. 3 and 4, we can, afterwards, if needed, provide guaranteed upper bounds on the obtained frequency domain errors (see discussion from Section 4.2.1 regarding the use of Sollya's `supnorm` command).

Secondly, all these previous methods (including the MATLAB-based approach) are based on shift-and-add constant multipliers (that is, multiplier circuits where one of the inputs is a constant and the other is a variable), thus avoiding the use of more expensive general purpose multipliers. Our work, on the other hand, uses the fused SOPC architecture of [71]. There, constant multipliers use an optimization of the KCM algorithm [54], and are merged into a single bit array [43]. A possible advantage of using the SOPC generator of [71] is that it determines automatically an optimal internal format to use for all the additions and multiplications from (5.3) such that the target output accuracy of the $\tilde{y}[k]$'s is guaranteed, thus helping streamline the rest of the design process.

5.3 Integrated hardware FIR filter design

Let us denote the overall TD error of an FIR architecture as $\varepsilon_k = |\widetilde{y}[k] - y[k]|$. The accuracy of the architecture will therefore be defined as a bound $\overline{\varepsilon}$ on ε_k. However, there is a deep relationship between $\overline{\varepsilon}$ and the output format. On the one hand, it is not possible to output more accuracy than the format can hold. On the other hand, it makes little sense to output less accuracy than the output format allows, especially in a hardware context where we pay for every bit. Therefore, in this work, the accuracy of the architecture is specified by the output format: if the least significant bit (LSB) of the output has weight 2^{-p}, then the architecture must be such that, for all allowable input signals and for all k, $|\widetilde{y}[k] - y[k]| < \overline{\varepsilon} = 2^{-p}$. Conversely, if an architecture is accurate to $\overline{\varepsilon} = 2^{-p}$, then its output format will have a LSB of weight 2^{-p}.

The design process can thus be carried out in two ways (the input format is always a design parameter the user specifies):

1. A value of N and corresponding quantized coefficients for which the FD specification holds are chosen. We compute the smallest output format such that the implemented filter is not affected by any rounding errors (this can always be done since all inputs and filter coefficients are finite precision values).

2. The user provides the output format (*i.e.,* gives the target LSB). The tool then computes a small N and a set of quantized coefficients such that the FD specification holds and its output is accurate to the specified limit.

The first one easier to implement, but the second is probably more intuitive in a hardware-centered scenario, when I/O formats are part of the specification. We will thus focus only on this second use case.

5.3.1 Computing the optimal filter out of the I/O format specification

Figure 5.5 gives an idea of our automated design flow. The input and output formats and FD quality bounds are given by the user. It might happen that there is no sound solution with these constraints, in which case the user is alerted. The first step determines an acceptable real-coefficient minimax filter with small N. This is to limit the complexity of the implemented filter. The standard approach to determine N seems to be based on a formula of Kaiser (see for instance [158, Ch. 7.7.4]; refinements of this guess are also presented by Shen and Strang [188]). It usually gives a good starting estimate for N, which can then be incremented or decremented as necessary (see also [11, Ch. 15.7–15.9]).

In the next box (emulate HW), FloPoCo determines the internal format it will use to satisfy the TD constraints inherited from the I/O formats. We chose to also use this format to quantize the coefficients, since in many cases, the format of the filter coefficients does not impact in a significant way the resource consumption of the (KCM-based) constant multipliers, but does affect the FD quality of the filter. As suggested in the next paragraphs, this choice can be reevaluated later on if needed.

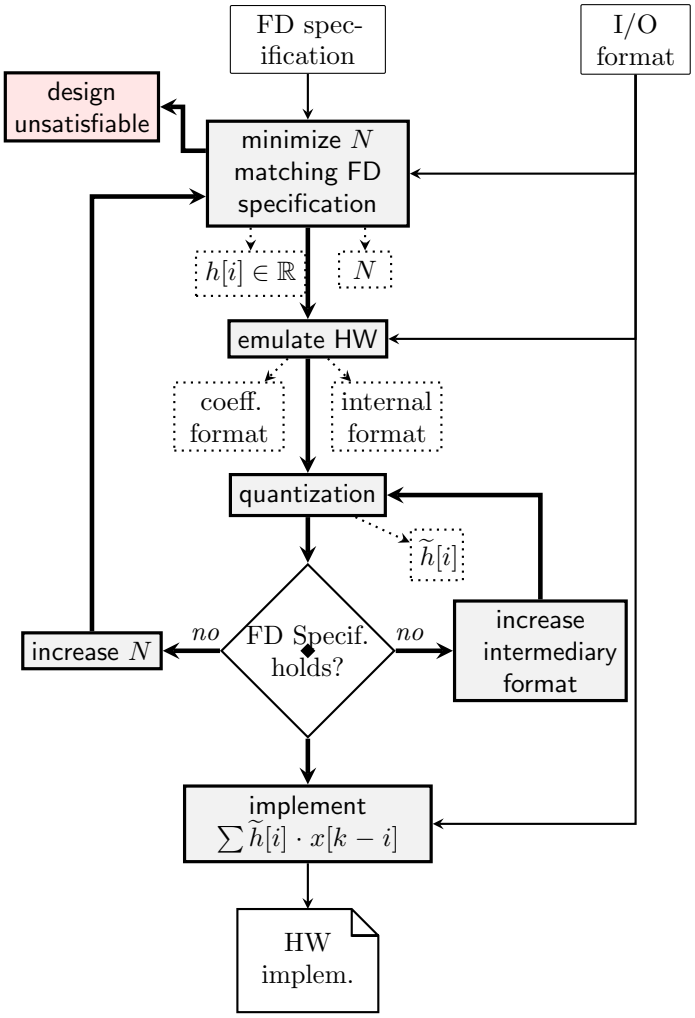

Figure 5.5: Proposed design flow

Based on this coefficient format, the coefficients are quantized, still subject to the FD constraints. In general, this internal format seems to be sufficient in ensuring that the Euclidean lattice-quantized filter we obtain satisfies the FD specification. We have never encountered an example where this is not the case, but if needed, we can do one of the following (or a combination of the two):

1. increase the coefficient and internal formats until the FD constraints are satisfied;

2. increment N and restart the design process.

The fact that we are not able to produce realistic design problems where the initial internal format is not sufficient seems to suggest this is not a concerning issue in practice.

5.4 Examples and results

As a working example, the default highpass filter specification of the `filterbuilder` command in MATLAB R2014B illustrates well the scope of our tool (other examples should provide very similar statistics in terms

of efficiency and resource consumption). It is a highpass FD specification with $\Omega = [0, 0.45\pi] \cup [0.55\pi, \pi]$ and

$$D(\omega) = \begin{cases} 0, & \omega \in [0, 0.45\pi] \quad \text{(stopband)}, \\ 1, & \omega \in [0.55\pi, \pi] \quad \text{(passband)}. \end{cases}$$

The error bounds are specified as a passband ripple below 1dB ($\delta_p \leqslant 0.0575$) and a stopband attenuation of at least 60dB ($\delta_s \leqslant 10^{-3}$). We take the input format to be that of signed values $(w_i, f_i) = (16, 15)$ with total width of 16 bits, out of which 15 of them are fractional bits (meaning the values will be inside the interval $[-1, 1)$, with the one bit not in the fractional part acting as sign bit). The output has $f_o = p = 15$ fractional bits.

In order to compare the performance of our tool with respect to MATLAB's `filterbuilder` from the hardware point of view, we generated and synthesized several filters that are supposed to adhere to the given specification. Syntheses were performed with Xilinx ISE 14.7, using the Virtex6 xc6vcx75t-2ff484 FPGA as a target device, with design goals set to balanced. Based on our discussion from Section 5.1.1, the performance of the designs is measured in FPGA resource consumption (number of look-up tables (LUT) and registers (Reg.)) and maximum operating frequency at a given latency (number of cycles required to compute a result from the moment the corresponding input was given).

MATLAB's overwhelming choice

The first two parts of Table 5.1, obtained using MATLAB's `filterbuilder`, show how runtime performance and resource utilization depend on the coefficient and internal fixed-point formats that the user gives. Using a too large internal (and coefficient) format leads to wasted resources with respect to the output format (first part of the table and second line in the second part). A too small internal format with respect to the coefficient and input formats (first and third line in the second part of the table) will also lead to wasted resources and an inaccurate output, due to all the roundings done internally to fit to the desired format.

The third part of the table is obtained using MATLAB's `minimizecoeffwl` routine, which tries to minimize the coefficient wordlengths, while still respecting the initial FD constraints. We had to leave the graphical interface to do this, hence the *expert mode* nomenclature in Table 5.1. The list of MATLAB commands used to generate those two quantizations are:

```
% FD specification for the filter

f = fdesign.highpass('Fst,Fp,Ast,Ap',0.45,0.55,60,1);
Hd = design(f,'equiripple');

% Apply the Matlab quantization tools to reduce the coefficient
% wordlengths as much as possible, while still satisfying the FD
% specification (can change the filter length)
% this line outputs a length N=49 filter with 10-bit coefficients

Hq = minimizecoeffwl(Hd);

% another quantization which uses the same number of coefficients
% as the double precision filter
% we obtain N=45 and a 12-bit coefficient quantization

Hq2 = minimizecoeffwl(Hd,'MatchrefFilter',true);
```

On most executions of the previous code, for `Hq` we obtained a stopband attenuation of about 60.15dB and a passband ripple of 0.79dB. The analogous values for `Hq2` were approximately 60.2dB and 0.88dB. At another level of comparison, when we tried to mimic the behavior of these two calls to `minimizecoeffwl` using our coefficient quantization approach, we got $N = 43$ with stopband attenuation 60.2dB and passband ripple

Table 5.1: Virtex6 synthesis results and accuracy measurements for generated FIR filters.

N	coeff. (w, f)	internal (w, f)	cost LUT + Reg.	performance cycles @ freq.
Using MATLAB's graphical interface defaults				
45	(16, 16)	(32, 31)	5463L + 754R	2 @ 91 MHz
Using MATLAB with naive trial and error ($N = 45$)				
45	(16,15)	(16, 15)†	5504L + 766R	2 @ 88 MHz
	(16,15)	(33, 31)	5180L + 779R	2 @ 95 MHz
	(31, 31)	(16, 15)†	12251L + 759R	2 @ 65 MHz
Using MATLAB in expert mode (with `minimizecoeffwl`)				
49	(10, 10)	(26, 25)	3087L + 811R	2 @ 104 MHz
45	(12, 12)	(28, 27)	4169L + 757R	2 @ 98 MHz
Using the proposed tool				
43	(22, 21)	(23,21)	3449L + 717R	2 @ 207 MHz

†: Ouput error analysis for these architectures shows that $\overline{\varepsilon} \simeq 2^{-10}$. For all the others it seems to be the case that $\overline{\varepsilon} \leqslant 2^{-15}$, as specified.

0.97dB for the 12-bit coefficient format. For 10-bit coefficient quantization, the values were $N = 47$, 60.2dB for the stopband attenuation and 0.79dB for the passband ripple.

In terms of resource use and frequency characteristics, we remark that the two expert mode filters are best among the MATLAB suite of trials (*i.e.,* smallest resource consumptions at the highest frequencies).

All these trials are illustration of the complexity incurred by the vast design parameter space. They show that both coefficient and internal formats should be optimized jointly. Such an exploration of the parameter space takes time. Besides, MATLAB does not seem to readily provide implementation accuracy data ($\overline{\varepsilon}$ or an equivalent signal/noise measure) at the end of the code generation process. We have already mentioned that with our approach, $\overline{\varepsilon} = 2^{-p}$ is guaranteed by the error analysis from [71].

Running the proposed algorithm

Our tool starts by determining a minimal N for which the FD specification remains satisfied when the filter has real coefficients. The `filterbuilder` result is a type I filter with $N = 45$ coefficients. We on the other hand, find a type I filter with $N = 43$.

Then, FloPoCo's SOPC kernel determines that it needs an internal format with 21 fractional bits. A filter with coefficients quantized to this format is then found: its passband ripple is smaller than 0.9 dB and its stopband attenuation is larger than 61 dB, matching the FD specification.

Automating the choice leads to better performance

The last line of Table 5.1 presents data about the filter generated using our approach (the total runtime was less than one minute). It is comparable in resource use to filters generated with MATLAB in expert mode, while being obtained automatically and is about twice as fast. The results are much better than those obtained using only MATLAB's graphical user interface.

The better frequency of our architecture (207MHz as opposed to the maximum of 104MHz for the other filters) essentially comes from a better implementation of the summation (using a bit heap data structure in our approach as opposed to rows of additions in the MATLAB-generated code).

However, this comparison of resource use and runtime performance shows where there is room for improvement in our approach. An expert user of MATLAB is able to take advantage of smaller quantization formats for the coefficients, which our current architectural choice inside FloPoCo is unable to exploit efficiently. This will be improved in future versions of our tool.

5.5 Conclusion and future work

The main purpose of this chapter was to present a first attempt at a complete and automatic hardware design work flow for fixed-point FIR filters using the ideas and tools developed in the previous two chapters and the FloPoCo generator. A real-size filter implementation of guaranteed quality is obtained within several seconds: apart from requiring minimal user intervention, this tool produces results which are more accurate, almost twice as fast, and comparable in area with what can be generated with similar mainstream tools.

We have obtained this result thanks to a constant concern to minimize the size of computations in the resulting architecture, coupled with strict error evaluation techniques used by FloPoCo. The main issue that this work has raised is the relationship between quality specification in the frequency and time domains. We have, for both, well understood and well implemented error analysis procedures. However, we still do not really transfer the error analysis done in one domain to the other. Improving on this could allow a better balance of error contributions (approximation error in the FD and evaluation/rounding error in the TD) to the overall filter quality, hence even better efficiency. Short-term work also focuses on extending this approach to the hardware synthesis of infinite impulse response filters.

6

Rational minimax approximation problems

The greater danger for most of us lies not in setting our aim too high and falling short; but in setting our aim too low, and achieving our mark.
Michelangelo

For a long time, the theories of minimax approximation by rational functions and by polynomials evolved along a common axis, due in no small part to similar properties of best approximations in the two settings. In particular, the Remez exchange algorithm for linear minimax approximations (the focal point of Chapter 3 of this text), was extended by Werner [227] and Maehly [141] to the rational case at the beginning of the 1960s, the latter referring to it as the *first direct method*.

Since its introduction, the rational Remez algorithm has played an important part in the design of efficient routines for the evaluation of special functions (see for instance the SPECFUN software package [62] and [87, Table 1]). Some computations from lattice field theory [125], [230, Ch. 15.7] also use it and there have been attempts to adapt the method in signal processing for the design of IIR filters [181, 182, 186, 187]. More recently, it is also used in nuclear physics, for burnup calculations [171].

It is perhaps appropriate to say that the exchange algorithm for computing best rational approximations is not as robust as its polynomial counterpart, for reasons we will discuss later on in the text. This is true, for instance, in the case of Chebfun's rational `remez` implementation, which is described by Pachón in his PhD thesis [161, Ch. 6]. Recently, with M. Javed (Chebfun group, Oxford University), we have started investigating if we can introduce any algorithmic and/or implementation improvements to this code, so that it has a more scalable numerical behavior (a multiple precision prototype implementation is also in development). We are still in the early exploratory stages of this work and this chapter is meant to give a brief account of some of the things we have done so far and what challenges and problems are left to conquer. Because of this, it is much shorter and less polished than Ch. 3 or 4.

6.1 The setting

Similarly to before, we want to approximate functions $f \in \mathcal{C}([a,b])$. The goal is to find $p \in \mathbb{R}_m[x]$ and $q \in \mathbb{R}_n[x]$ such that

$$\left\| f - \frac{p}{q} \right\|_{\infty,[a,b]} = \max_{x\in[a,b]} \left| f(x) - \frac{p(x)}{q(x)} \right|$$

is minimized, with the assumption that $q(x)$ has no zeros in $[a,b]$. If q has any zeros, they should also be zeros of p, in which case they can be factored out; we call the resulting rational function *irreducible* (up to

constant factors). We then denote by

$$\mathcal{R}_{m,n} = \left\{ \frac{p(x)}{q(x)} : \deg p \leqslant m, \deg q \leqslant n, \frac{p}{q} \text{ is irreducible and } q(x) > 0 \text{ over } [a,b] \right\}$$

the feasible set from which a solution to the rational minimax problem is to be searched for; zero has the irreducible form $0/1$ and $\deg 0 = -\infty$. We also define the *defect* of $p/q \in \mathcal{R}_{m,n}$ to be $d = \min\{m - \deg p, n - \deg q\}$, giving us a way to measure how both numerator and denominator degree differ from their maximal allowed values. If $d > 0$ we call the rational approximation *degenerate*, otherwise it is *nondegenerate* or *normal*. As an example, consider $1/(1+x^2)$ over the interval $[-1,1]$. It has defect 0 in $\mathcal{R}_{0,2}$ or $\mathcal{R}_{1,2}$ and defect 1 in $\mathcal{R}_{1,3}$.

The existence and characterization of best rational approximations can be summarized by the following result, which is very similar to its polynomial equivalent:

Theorem 6.1 (Uniqueness and alternation of minimax approximations)**.** *If $f \in \mathcal{C}([a,b])$, then $r^\star = p^\star/q^\star$ is the unique best uniform approximation of f (up to a constant scaling factor) out of $\mathcal{R}_{m,n}$, if and only if $f - r^\star$ equioscillates between at least $m+n+2-d$ extreme points, with d being the defect of $r^\star$ in $\mathcal{R}_{m,n}$.*

Proof. See for instance [205, Ch. 24]. The existence of best rational approximations can be traced to Walsh [220], while the uniqueness and alternation properties are due to Achieser [2,3]. □

Theorem 6.1 is supplemented by a result due to de la Vallée Poussin [72] which provides a lower bound on the minimax error [3, p. 52–53], analogous to Theorem 3.10:

Theorem 6.2 (de la Vallée Poussin)**.** *Let $r = p/q \in \mathcal{R}_{m,n}$ with defect d and $N = m+n+2-d$. If at the points $x_0 < x_1 < \ldots < x_{N-1}$ in $[a,b]$ we have*

$$f(x_i) - r(x_i) = (-1)^i \lambda_i, \lambda_i \neq 0, i = 0, \ldots, N-1$$

and all λ_i have the same sign, then, for all $s \in \mathcal{R}_{m,n}$

$$\min_i |f(x_i) - r(x_i)| \leqslant \max_i |f(x_i) - s(x_i)|, \tag{6.1}$$

and consequently

$$\min_i |f(x_i) - r(x_i)| \leqslant \|f - r^\star\|_{\infty,[a,b]} \leqslant \|f - r\|_{\infty,[a,b]}. \tag{6.2}$$

6.2 The rational exchange algorithm

The second Remez algorithm as it applies to the rational approximation problem is very similar to Algorithm 2:

(1) Choose $N = m+n+2$ points in $[a,b]$ (call them $x_i^{(k)}, i = 0, \ldots, N-1$) with k set to the value 1.

(2) Determine h_k and $r_k = \frac{p_k}{q_k} \in \mathcal{R}_{m,n}$ to satisfy the system of equations

$$f(x_i^{(k)}) - r_k(x_i^{(k)}) = (-1)^i h_k, \qquad i = 0, \ldots, N-1. \tag{6.3}$$

(3) Choose N local maxima $\left\{x_i^{(k+1)}\right\}_{i=0}^{N-1}$ of $|f - r_k|$ such that

$$\left| f(x_i^{(k+1)}) - r_k(x_i^{(k+1)}) \right| \geqslant |h_k|, \qquad i = 0, \ldots, N-1,$$

at least one point where $\|f - r_k\|_{\infty,[a,b]}$ is attained is included, and the sign of $f - r_k$ alternates at these points. If the current approximation has converged to within a given threshold $\varepsilon_t > 0$ (*i.e.*, $\frac{\|f-r_k\|_{\infty,[a,b]} - h_k}{\|f-r_k\|_{\infty,[a,b]}} \leqslant \varepsilon_t$) return r_k, else go to step (2) with $k = k+1$.

If step (2) always succeeds during execution, then convergence to the best approximation is assured (see for instance [222, Thm. 9.14]). If, in addition, regularity assumptions similar to those of [217] hold, then the convergence speed is quadratic [67], just like with minimax polynomial approximation.

Contrary to the polynomial case however, the rational Remez algorithm can fail (both in theory and in practice). As mentioned in [187], there are two theoretical reasons for this, but which have the same effect: at some iteration k, the reference $\mathbf{x}^{(k)}$ does not allow for a $r_k \in \mathcal{R}_{m,n}$ solution of (6.3). They are:

- The best approximation $r^\star$ is in fact degenerate. Then either the best rational approximation on a reference set during execution is degenerate or the algorithm yields a sequence of approximations that approach degeneracy;
- Sensitivity to the initial reference set, the algorithm failing even though $r^\star$ is nondegenerate. On the other hand, if we start sufficiently close to the target minimax approximation, then the exchange algorithm will converge [36, Ch. V.6.B]. To our knowledge, it seems extremely difficult to determine such a starting approximant in general.

Example 6.3. *Inspired by [222, Ch. 9.3], let us consider*

$$f(x) = x^2, \qquad [a, b] = [a, 1], \qquad m = n = 1.$$

When $-1 < a < 1$, then the corresponding best approximation $r^\star$ is nondegenerate, giving rise to an alternating set of $1+1+2 = 4$ points. On the other hand, when $a = -1$, the best approximation is degenerate, since $r^\star = 1/2$ verifies Theorem 6.1. Figure 6.3 shows what happens when $a \to -1$ and we have near degenerate approximations. The coalescing of the two rightmost references hints at computational issues. Indeed, when using our rational Remez code for $a = -0.99$, it failed to produce a result except for when the initial reference (especially the last two nodes) was very close to the equialternating set. For $a = -1$, as expected, the algorithm did not even converge.

A possible way of accounting for (near) degeneracy is if we have knowledge of so-called square block structures in the *Walsh table* of f. Introduced in [221] by Walsh, it is a bi-dimensional array housing the best degree (m, n) approximations to f, where m indexes the rows and n the columns. If there are identical entries inside this table, it means that f is degenerate for certain degrees (m, n). For instance, in the case of even functions, the Walsh table contains blocks of 2×2 identical entries, that is, the minimax approximations of types $(m, n), (m + 1, n), (m, n + 1), (m + 1, n + 1)$ are the same, when m and n are even. A similar property holds for odd functions (see also discussion in [205, Ch. 24] and the references therein). Chebfun `remez` has an automated test for detecting degeneracy in even/odd functions. Other nontrivial examples and theoretical properties regarding degeneracy are explored by Ralston [174].

The remainder of this section discusses how we perform each of the three steps and the issues we encountered in practice.

6.2.1 Initialization

A conclusion to draw from the previous paragraphs is the sensitivity of the rational exchange algorithm to the starting reference. There are several alternatives in the literature on how to perform step (1). For instance, Curtis and Osborne [67] suggest the use of Chebyshev nodes of the second kind scaled to $[a, b]$, while Werner, Stoer and Bommas [228] propose taking r_1 as the rational interpolant of f at the scaled roots of the Chebyshev polynomial T_N. Another approach [98] is to first construct the best polynomial approximant of degree $m + n$ (which should pose no problems); next, use the resulting equialternating reference as the initial guess for the degree $(m + n - 1, 1)$ problem. The degree of the numerator is successively reduced and the degree of the denominator is successively raised until the target (m, n) approximation is reached.

More recently, Pachón and Trefethen (see [161, Ch. 6] and [163]) use Chebyshev-Padé and Carathéodory-Fejér rational approximations, which, when numerically computable, tend to be near minimax. A CF-based approach with a fall back to Chebyshev nodes is the current default in the rational Chebfun `remez` (CF approximation inside Chebfun is done through the `cf` command [213]), a strategy which, at least for the moment, we also use in our code.

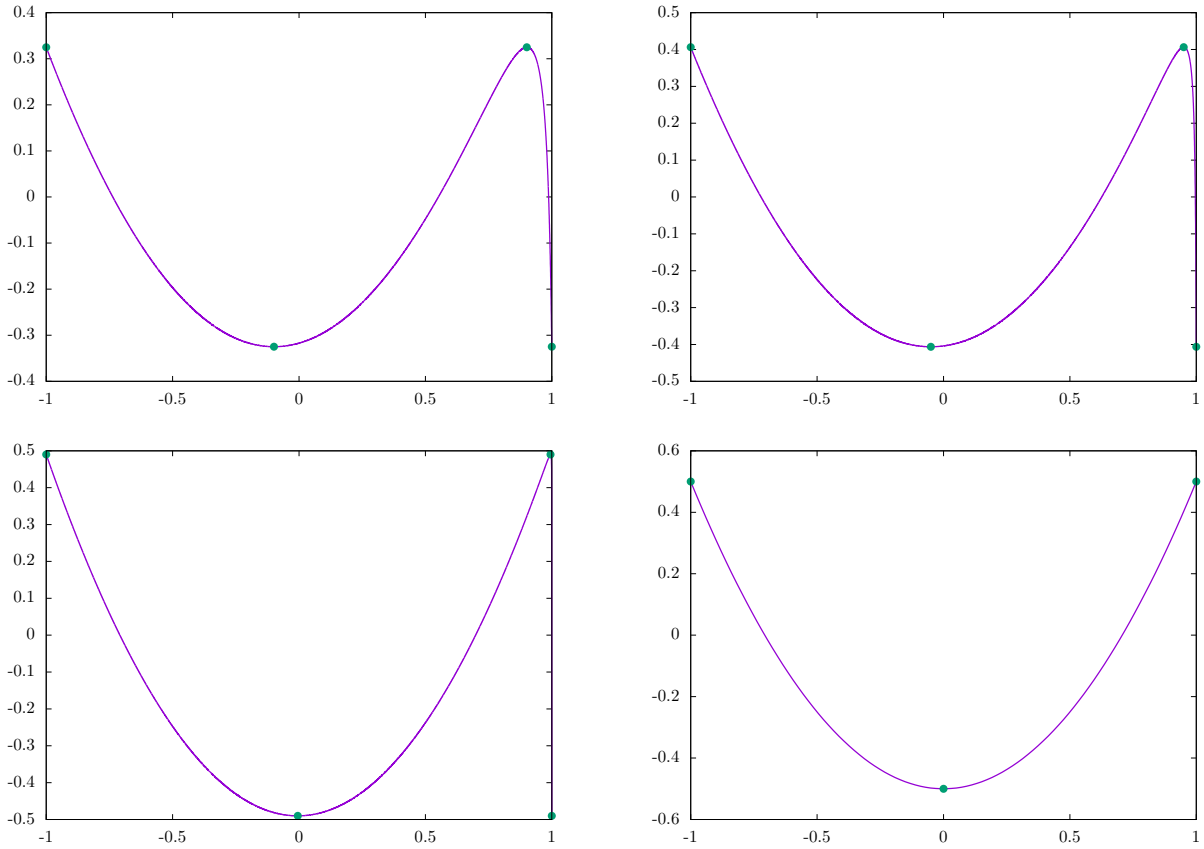

Figure 6.1: Minimax approximation **errors** for Example 6.3 when $a = -0.8$ (top left), $a = -0.9$ (top right), $a = -0.99$ (bottom left) and $a = -1$ (bottom right). In all four cases the abscissa is linearly scaled to $[-1, 1]$.

Based on our experiments, sensitivity to the initial reference is not only common to near degenerate cases, but also for instance when trying to approximate functions which have complex-valued singularities inside or close to the interior of $[a, b]$ and the reference values tend to cluster near those points at exponential rates (we especially looked at functions that were not everywhere differentiable on $[a, b]$).

Example 6.4. *Consider approximating the function* $f(x) = |x - 0.4| \sin(|x - 0.2|)$ *over* $[-1, 1]$ *(not differentiable at* $x = 0.2$ *and* $x = 0.4$*) with rational interpolants of type* (n, n) *of increasing degree. Starting from degree* $(4, 4)$ *we had convergence problems when starting with Chebyshev nodes; the same thing happened when we used the Remez algorithm implementation from the Computer Algebra system Maple (the* `minimax` *routine). Even using* 1000 *bits of precision (roughly* 19 *times more than what* `double` *values can store) did not ensure convergence with our multiple precision prototype. Using a CF-based initialization however, coupled with the changes we suggest in Section 6.2.3, we were able to go up to degree* $(8, 8)$ *approximations using only double precision arithmetic inside Chebfun. For* $n \geqslant 9$*, the numerically computed CF approximation was not usable. When we gave the same CF-based points to the multiple precision code, it also converged without problems.*

It seems, however, that numerically, none of these approaches is sufficiently robust (see also experiments in [161, Ch. 6.6]), with the caveat that when the CF and/or Chebyshev-Padé methods work, convergence is usually achieved very quickly, Example 6.4 being evidence of this. When the function starts possessing some regularity and any singularities are close to a or b, we noticed much less sensitivity to the starting reference.

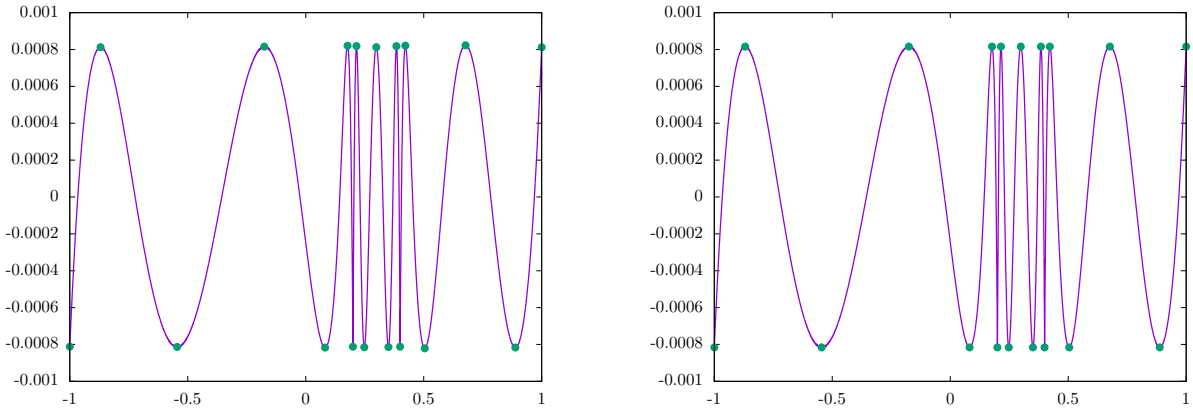

Figure 6.2: Approximation errors after the first iteration of the exchange algorithm has been executed (left) and at the end of execution (right) for the degree $(8,8)$ best approximation of $f(x) = |x-0.4|\sin(|x-0.2|), x \in [-1,1]$. The extrema of the degree $(8,8)$ CF approximation to f were used as starting reference vector.

6.2.2 Determining the current approximant

For a simplified notation, let us drop the iteration index k in eq. (6.3) and look at how we can determine the rational interpolant $r = p/q$ and levelled error h corresponding to an iteration of the exchange algorithm; to begin with, we consider both p and q expressed in the same basis $\{\varphi_0, \varphi_1, \ldots, \varphi_n, \ldots\}$ of $\mathbb{R}[x]$ as $p(x) = \sum_{k=0}^{m} a_k \varphi_k(x)$ and $q(x) = 1 + \sum_{k=1}^{n} b_k \varphi_k(x)$, where $\deg \varphi_i = i$. Since $r \in \mathcal{R}_{m,n}$, we can rewrite the system of equations as

$$p(x_i) + (-1)^i h q(x_i) = f(x_i) q(x_i), \qquad i = 0, \ldots, N-1, \tag{6.4}$$

which in matrix notation gives

$$\begin{bmatrix} (\mathbf{D}_2 - \mathbf{D}_1 h)\mathbf{V}_2 & -\mathbf{V}_1 \end{bmatrix} \begin{bmatrix} \mathbf{b} \\ \mathbf{a} \end{bmatrix} = \mathbf{0} \tag{6.5}$$

or

$$\mathbf{V}_1 \mathbf{a} + h\mathbf{D}_1 \mathbf{V}_2 \mathbf{b} = \mathbf{D}_2 \mathbf{V}_2 \mathbf{b} \tag{6.6}$$

where

$$\begin{aligned} \mathbf{a} &= [a_0 \cdots a_m]^t, & \mathbf{b} &= [1\ b_1 \cdots b_n]^t, \\ (\mathbf{V}_1 \mathbf{a})_i &= p(x_i), & (\mathbf{V}_2 \mathbf{b})_i &= q(x_i), \\ (\mathbf{D}_1)_{i,i} &= (-1)^i, & (\mathbf{D}_2)_{i,i} &= f(x_i). \end{aligned}$$

$\mathbf{D}_1$ and $\mathbf{D}_2$ are both diagonal matrices and $\mathbf{V}_1$ is a basis matrix of the form

$$\mathbf{V}_1 = \begin{bmatrix} \varphi_0(x_0) & \varphi_1(x_0) & \cdots & \varphi_m(x_0) \\ \vdots & & & \vdots \\ \varphi_0(x_{N-1}) & \varphi_1(x_{N-1}) & \cdots & \varphi_m(x_{N-1}) \end{bmatrix}$$

with a similar structure for $\mathbf{V}_2$. Since $\{\varphi_0, \ldots, \varphi_m\}$ satisfies the Haar condition (see Definition 3.3), $\mathbf{V}_1$ has full rank $m+1$ and there exists a matrix $\mathbf{Q} \in \mathbb{R}^{(n+1)\times(m+n+2)}$ also of full rank such that $\mathbf{Q}\mathbf{V}_1 = \mathbf{0}$. Applied to (6.6), $\mathbf{a}$ is eliminated and we have a generalized eigenvalue problem in h and $\mathbf{b}$

$$\mathbf{Q}\mathbf{D}_2\mathbf{V}_2\mathbf{b} = h\mathbf{Q}\mathbf{D}_1\mathbf{V}_2\mathbf{b}. \tag{6.7}$$

It can be shown that all of the $n+1$ eigenvalues h which verify (6.7) are real valued (see for instance [168, Ch. 10.2]) and at most one of the corresponding eigenvectors $\mathbf{q}$ gives rise to a pole free interpolant over $[a,b]$ (this was first shown by Maehly in [141, Sec. 7]).

Rational interpolation

It seems that in many references on the computational aspects of the rational exchange algorithm, the nonlinear set of equations (6.3) are not solved via the generalized eigenvalue problem explicitly, but by using Newton's method on (6.5) (see for instance [67, Sec. 4] and [36, Ch. V.6.B]). The approach taken in Chebfun `remez` on the other hand, which we also continue to use, solves a form of eq. (6.7) by means of MATLAB's `eig` command. In order to explain how it works, we need some results introduced in [161, Ch. 4] and [162] regarding rational interpolation problems, which we now summarize for convenience.

Consider the following instance of rational interpolation (known as the *Cauchy interpolation problem*): given two numerator and denominator degrees $m, n \in \mathbb{N}$ and $n+m+1$ pairwise distinct points $x_0, \dots, x_{m+n} \in \mathbb{C}$ and values $f_0, \dots, f_{m+n} \in \mathbb{C}$, determine a rational function $r = p/q$, where $p \in \mathbb{C}_m[x]$ and $q \in \mathbb{C}_n[x]$ have no common factors other than constants, such that

$$r(x_i) = \frac{p(x_i)}{q(x_i)} = f_i, \qquad i = 0, \dots, m+n. \tag{6.8}$$

When $q(x_i) \neq 0, i = 0, \dots, m+n$, solving (6.8) is equivalent to the linear problem

$$p(x_i) = f_i q(x_i), \qquad i = 0, \dots, m+n, \tag{6.9}$$

Working with (6.9) instead of (6.8) is a common strategy for the computation of rational interpolants. In case of the exchange algorithm it is also appropriate to consider (6.9) since the denominator of the trial approximant used at each iteration should not change sign over $[a, b]$.

The approach for solving (6.9) on an arbitrary grid vector $\mathbf{x} = [x_0 \ \cdots \ x_{m+n}]^t \in \mathbb{C}^{m+n}$ presented by Pachón, Gonnet and van Deun in [162] consists of representing p and q via a family of polynomials $\{\varphi_i\}_{i=0}^{m+n}$, where $\deg \varphi_i = i$ and the φ_i are orthogonal with respect to the inner product

$$\langle f, g \rangle_{\mathbf{x}} := \sum_{k=0}^{m+n} f(x_k)\overline{g(x_k)}, \tag{6.10}$$

meaning

$$\langle \varphi_i, \varphi_j \rangle_{\mathbf{x}} = \begin{cases} c_i > 0 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases} \tag{6.11}$$

If we take the basis matrix $\mathbf{C} = [\varphi_0(\mathbf{x})| \ \cdots \ |\varphi_{m+n}(\mathbf{x})] \in \mathbb{C}^{(m+n+1)\times(m+n+1)}$ and $\hat{\mathbf{c}} = [c_0 \ \cdots \ c_{m+n}]^t$, then (6.11) is equivalent to $\mathbf{C}^\star\mathbf{C} = \operatorname{diag}(\hat{\mathbf{c}})$. In the following, for any $k \times k$ matrix $\mathbf{A}$, $\mathbf{A}_{i:j}$ with $0 \leqslant i < j \leqslant k$, will denote the submatrix of size $k \times (j - i + 1)$ formed with the columns i to j of $\mathbf{A}$. The result used to solve the rational interpolation problem is

Theorem 6.5. *Given* $\mathbf{f} = [f_0 \ \cdots \ f_{m+n}]^t \in \mathbb{C}^{(m+n+1)}$*, the polynomials*

$$p(x) = \sum_{k=0}^{m} a_k \varphi_k(x) \text{ and } q(x) = \sum_{k=0}^{n} b_k \varphi_k(x)$$

satisfy (6.9) on the grid $\mathbf{x}$ *if and only if*

$$\mathbf{b} \in \ker(\mathbf{C}^\star_{m+1:(m+n)}\mathbf{F}\mathbf{C}_{0:n}) \text{ and } \mathbf{C}^\star_{0:m}\mathbf{F}\mathbf{C}_{0:n}\mathbf{b} = \mathbf{c}, \tag{6.12}$$

where $\mathbf{F} = \operatorname{diag}(\mathbf{f}), \mathbf{b} = [b_0 \ \cdots \ b_n]^t$ *and* $\mathbf{c} = [c_0 a_0 \ \cdots \ c_m a_m]^t$.

Proof. See [162, Thm. 3.1]. □

Let us denote $\mathbf{C}^\star_{m+1:(m+n)}\mathbf{F}\mathbf{C}_{0:n}$ with $\mathbf{Z} \in \mathbb{C}^{n\times(n+1)}$. A common way of determining $\mathbf{b}$ is to extract it from the singular value decomposition (SVD) of $\mathbf{Z}$. In fact, we have

Theorem 6.6. *Let $p \in \mathbb{C}_m[x]$ and $q \in \mathbb{C}_n[x]$ satisfy (6.9). Then p and q are of the form $p(x) = (\overline{p}sv)(x)$ and $q(x) = (\overline{q}sv)(x)$, where*

- *$\overline{p}$ and $\overline{q}$ are relatively prime polynomials, unique up to normalization, and of exact degrees $\overline{m}$ and $\overline{n}$, $0 \leqslant \overline{m} \leqslant m$, and $0 \leqslant \overline{n} \leqslant n$,*
- *s is a monic polynomial of degree δ_s, where $0 \leqslant \delta_s \leqslant \delta = \min(m - \overline{m}, n - \overline{n})$, whose zeros are the interpolation points x_j such that $q(x_j) = 0$, and*
- *v is an arbitrary polynomial of degree $\delta_v = \delta - \delta_s$.*

Moreover, we have that

$$\operatorname{rank}(\mathbf{Z}) = n - \delta_v,$$

which is equivalent to saying that the number of zero singular values of $\mathbf{Z}$ is equal to the number of common factors of p/q (other than those of s).

Proof. See [162, Thm. 4.1] and the references therein. □

We recognize δ to be the defect of p/q. Numerical problems with rational interpolation seem, in many cases, to be traceable to v, whose degree is equal to the nullity of $\mathbf{Z}$. They are due to the numerical rank-deficiency of $\mathbf{Z}$ which manifests itself through the fast decay of $\mathbf{Z}$'s singular values (see discussions from [162, Sec. 4] and [205, Ch. 26]) when m and n increase.

Going back to the rational exchange algorithm, we can interpret (6.4) as a rational interpolation problem with *weighted deviations* [141] (if we consider the levelled error h a constant):

$$r(x_i) = f(x_i) - (-1)^i h, \qquad i = 0, \ldots, N-1.$$

Using Theorem 6.5, we construct the Vandermonde-type matrix $\mathbf{C} = [\varphi_0(\mathbf{x})| \ \cdots \ |\varphi_{N-1}(\mathbf{x})] \in \mathbb{R}^{N \times N}$, where the φ_i polynomials are orthogonal with respect to the inner product $\langle \cdot, \cdot \rangle_{\mathbf{x}}$. The coefficients $\mathbf{b} = [b_0 \ \cdots \ b_n]^t$ of q are such that

$$\mathbf{b} \in \ker(\mathbf{C}^{\star}_{m+1:N-1}\mathbf{D}\mathbf{C}_{0:n}), \tag{6.13}$$

where $\mathbf{D} \in \mathbb{R}^{N \times N}$ is a diagonal matrix with $(\mathbf{D})_{(i,i)} = f(x_i) - (-1)^i h$. Eq. (6.13) gives rise to the generalized eigenvalue problem (equivalent to (6.7))

$$\left[\mathbf{C}^{\star}_{m+1:N-1}\mathbf{F}\mathbf{C}_{0:n}\right]\mathbf{b} = h\left[\mathbf{C}^{\star}_{m+1:N-1}\mathbf{\Delta}\mathbf{C}_{0:n}\right]\mathbf{b}, \tag{6.14}$$

where both $\mathbf{F}$ and $\mathbf{\Delta}$ are diagonal matrices with $(\mathbf{F})_{(i,i)} = f(x_i)$ and $(\mathbf{\Delta})_{(i,i)} = (-1)^i$.

A discussion on some computational aspects of determining $\mathbf{C}$ is given in [162, Sec. 3.3]. For the moment, we compute it as the orthogonal part of the QR decomposition of an $N \times N$ Vandermonde-type matrix whose columns are polynomials of an arbitrary basis evaluated over $\mathbf{x}$.

As mentioned before, at most one eigenpair of (6.14) will give rise to a bounded r over $[a, b]$. Inside Chebfun `remez`, for each eigenpair, the values $q(x_i)$ are computed, and one of them where the entries of $q(\mathbf{x})$ have the same sign is chosen. In many of the cases we applied the algorithm on, when such a pair exists, it is unique, although in the future we should probably replace this tactic with a more robust test.

6.2.3 Updating the reference set

Once the current rational interpolant is determined, we need to find the local maxima of $|f - r|$ to construct the next reference set, analogous to what we did in Section 3.8 for the polynomial case.

The way this is done in the implementation of the `remez` code from [161, Ch. 6] is that, once the appropriate h and $\mathbf{b}$ are computed, it is not hard to determine $p(\mathbf{x})$ and $q(\mathbf{x})$. These values then serve to construct barycentric polynomial representations of p and q on the current reference $\mathbf{x}$, which are then given to Chebfun to construct `chebfuns` of p and q. The local maxima of $|f - r|$ are now retrieved as the roots of the `chebfun` representation of $q^2 f' - qp' + pq'$ using the `roots` command [205, Ch. 18].

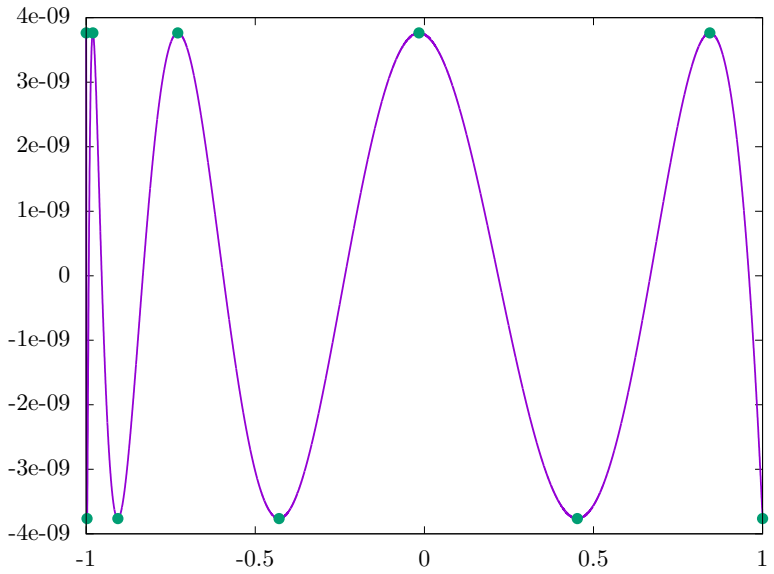

Figure 6.3: The error function for the best $\mathcal{R}_{4,4}$ minimax approximation of $f(x) = \frac{1}{\log(x+2.005)}$ over $[-1, 1]$.

When trying to approximate functions which have singularities close to or on $[a, b]$, the reference sets $\mathbf{x}$ will tend to cluster near those points, many times adversely affecting the numerical quality of the barycentric formula for computing p and q and their ratio. Experimentally, we noticed better numerical behavior when using a *rational* barycentric representation of r (as mentioned in Section 3.7.1). We are currently working on trying to better understand the stability properties of this approach in the context of the exchange algorithm (a starting point is the error analysis on rational barycentric interpolation from Salazar Celis's PhD thesis [50]). To compute the barycentric weights, we retrieve them from the kernel of an appropriate matrix, which can be consulted in [23]. We also use the subdivision strategy introduced in Section 3.8.2 to split $[a, b]$ into subintervals where it is easier to retrieve the extrema of the current error function $f - r$.

As hinted in Example 6.4, these two changes are currently included in the prototype `ratRemez` function from the `feature-rational-remez` branch[1] of the main Chebfun repository. They are also available in the multiple precision prototype[2].

6.3 More examples

If there are no problems with solving the generalized eigenproblem (6.14) in double precision arithmetic (*i.e.,* each iteration of the exchange algorithm generates a pole-free interpolant and the numerical solutions for h and $\mathbf{b}$ are accurate), the `ratRemez` function seems to scale a little bit better with m and n than the main Chebfun `remez` code. An example is the specification whose minimax error is given in Figure 6.3 which only converged when calling `ratRemez`, while computing the best $\mathcal{R}_{4,3}$ minimax approximation for the same function was successful with both codes. In Maple, the `minimax` routine did not converge for the $(4, 3)$ and $(4, 4)$ problems, even when using 100 digits of accuracy.

Similar things happened when we considered $f(x) = |x|$ over $[-1, 1]$; we were able to converge up to degree $(10, 10)$ with a minimax error of about $2.69 \cdot 10^{-4}$, while the unmodified code already failed for degree $(6, 6)$. When we tried the $\mathcal{R}_{12,12}$ problem, the eigensolver was not accurate enough in double precision arithmetic; it failed to produce a correct result even when we initialized the function with the minimax reference, which we were able to retrieve with our multiple precision exchange routine. For Maple, we had

[1] https://github.com/chebfun/chebfun/tree/feature-rational-remez
[2] https://github.com/sfilip/ratremez

issues even for the degree $(2,2)$ instance.

Theoretically, it was Varga and Carpenter [215] that showed that the minimax error for polynomial approximations of $|x|$ satisfies

$$E_n(|x|\,;[-1,1]) \sim \frac{\beta}{n}, \qquad \beta = 0.2801\ldots,$$

leading to a very modest $\mathcal{O}(n^{-1})$ rate of convergence, if we are to follow what we said in Section 3.6.1.

In the rational case however, we know that the errors of best approximations of type (n,n) to $|x|$ satisfy

$$E_{n,n}(|x|\,;[-1,1]) \sim 8e^{-\pi\sqrt{n}},$$

a result due to Stahl [197]. This result was in fact conjectured by Varga, Ruttan and Carpenter [216] not long before, using a multiple precision (200 decimal digits) implementation of the rational Remez algorithm. The computations they performed took advantage of the fact that

$$E_{2n,2n}(|x|\,;[-1,1]) = E_{n,n}(\sqrt{x};[0,1]),$$

and they actually used the exchange algorithm on the square root function, for $n = 1,\ldots,40$. We were also successful in reproducing their minimax error computations with our multiple precision exchange prototype (Figure 6.4 shows one of the error curves we obtained).

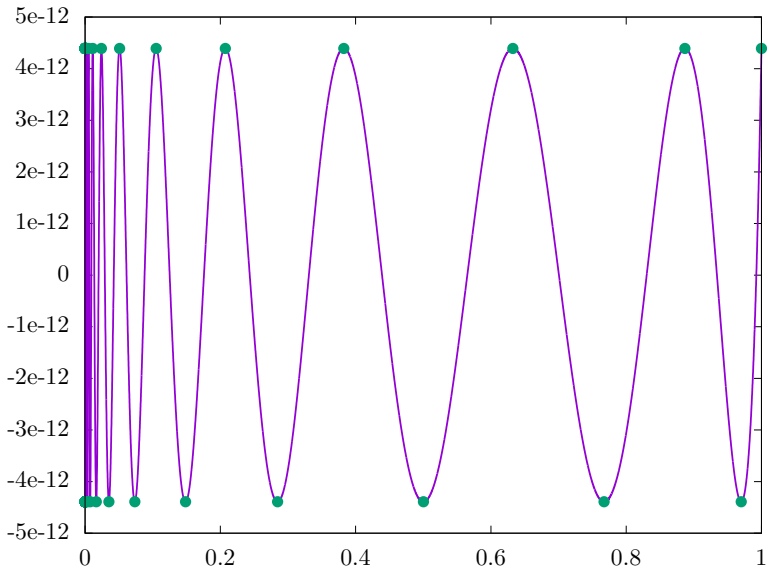

Figure 6.4: The error function for the best $\mathcal{R}_{40,40}$ minimax approximation of $f(x) = \sqrt{x}$ over $[0,1]$.

Another well-studied example, which we briefly mentioned in the introduction of the present text, is the exponential function e^x over $(-\infty,0]$. If for the previous example polynomials still had some approximation power, in this case we can do no better than an approximation error of $1/2$, due to the constant function $p^\star = 1/2$.

By considering rational functions however, things are much better. For type $(0,n)$ approximations, Schönhage [185] showed that we have

$$\lim_{n\to\infty} E_{0,n}^{1/n}(e^x;(-\infty,0]) = \frac{1}{3},$$

whereas for type (n,n) minimax approximations we know from Aptekarev [12] that,

$$E_{n,n}(e^x;(-\infty,0]) \sim 2H^{n+1/2}, \qquad H = 1/9.2890354\ldots$$

Table 6.1: Rational minimax approximation errors for e^x on $(-\infty, 0]$ of type $(n, n), n = 1, \ldots, 60$.

n	$E_{n,n}(e^x; (-\infty, 0])$	n	$E_{n,n}(e^x; (-\infty, 0])$
1	$6.683104216185046347061 \cdot 10^{-1}$	31	$6.439041065521390118176 \cdot 10^{-31}$
2	$7.358670169580529280013 \cdot 10^{-3}$	32	$6.932444346272943657112 \cdot 10^{-32}$
3	$7.993806363356878288081 \cdot 10^{-4}$	33	$7.463619283376389623748 \cdot 10^{-33}$
4	$8.652240695288852348224 \cdot 10^{-5}$	34	$8.035458036521132725158 \cdot 10^{-34}$
5	$9.345713153026646476754 \cdot 10^{-6}$	35	$8.651074112710932240980 \cdot 10^{-35}$
6	$1.008454374899670707935 \cdot 10^{-6}$	36	$9.313819316267808643033 \cdot 10^{-36}$
7	$1.087497491375247960867 \cdot 10^{-7}$	37	$1.002730199532148674598 \cdot 10^{-36}$
8	$1.172265211633490717795 \cdot 10^{-8}$	38	$1.079540668419148048255 \cdot 10^{-37}$
9	$1.263292483322314146095 \cdot 10^{-9}$	39	$1.162231524864483662593 \cdot 10^{-38}$
10	$1.361120523345447749871 \cdot 10^{-10}$	40	$1.251252964917550487127 \cdot 10^{-39}$
11	$1.466311194937487140668 \cdot 10^{-11}$	41	$1.347089644624453421663 \cdot 10^{-40}$
12	$1.579456837051238771487 \cdot 10^{-12}$	42	$1.450263318089079447854 \cdot 10^{-41}$
13	$1.701187076340352966416 \cdot 10^{-13}$	43	$1.561335677432057988000 \cdot 10^{-42}$
14	$1.832174378254041275156 \cdot 10^{-14}$	44	$1.680911410106628688161 \cdot 10^{-43}$
15	$1.973138996612803428626 \cdot 10^{-15}$	45	$1.809641490212876992183 \cdot 10^{-44}$
16	$2.124853710495223748800 \cdot 10^{-16}$	46	$1.948226721724903899404 \cdot 10^{-45}$
17	$2.288148563247891960405 \cdot 10^{-17}$	47	$2.097421552916174407673 \cdot 10^{-46}$
18	$2.463915737765169274831 \cdot 10^{-18}$	48	$2.258038182743982444147 \cdot 10^{-47}$
19	$2.653114658063312766926 \cdot 10^{-19}$	49	$2.430950981542696891201 \cdot 10^{-48}$
20	$2.856777383549093706691 \cdot 10^{-20}$	50	$2.617101250085849882192 \cdot 10^{-49}$
21	$3.076014349505790506914 \cdot 10^{-21}$	51	$2.817502342918495410770 \cdot 10^{-50}$
22	$3.312020500551318690751 \cdot 10^{-22}$	52	$3.033245183843618403323 \cdot 10^{-51}$
23	$3.566081860636424584770 \cdot 10^{-23}$	53	$3.265504203580495204747 \cdot 10^{-52}$
24	$3.839582582168132126936 \cdot 10^{-24}$	54	$3.515543731910406302247 \cdot 10^{-53}$
25	$4.134012517285363006271 \cdot 10^{-25}$	55	$3.784724879098481327846 \cdot 10^{-54}$
26	$4.450975355730424689793 \cdot 10^{-26}$	56	$4.074512944043172953905 \cdot 10^{-55}$
27	$4.792197375888904189931 \cdot 10^{-27}$	57	$4.386485389471399102518 \cdot 10^{-56}$
28	$5.159536858257132654665 \cdot 10^{-28}$	58	$4.722340427583360413152 \cdot 10^{-57}$
29	$5.554994213751622674642 \cdot 10^{-29}$	59	$5.083906262873225056042 \cdot 10^{-58}$
30	$5.980722882849695437271 \cdot 10^{-30}$	60	$5.473151042428354389685 \cdot 10^{-59}$

with H being known as *Halphen's constant*. Aptekarev's result follows earlier work of Gonchar and Rakhmanov [92] on the asymptotic behavior of best approximations of the nth root function.

During the 1970s, the rate of convergence for (n, n) best approximations of the exponential function was thought to be $\mathcal{O}(9^{-n})$ and was known as the 1/9 *conjecture*. It was during the first half of the 1980s that H began to surface, through numerical experiments such as those performed by Trefethen and Gutknecht [206] using the Carathéodory-Fejér method and by Carpenter, Ruttan and Varga [48], by again using a multiple precision implementation (with 230 decimal digits of precision) of the exchange algorithm for $n = 0, \ldots, 30$.

We also repeated these last experiments with our own prototype code, but we doubled n, going up to $n = 60$ (total computation time for all degrees was about 8 hours on my Intel i7-3687U CPU Linux machine; the 1984 runtime [48] for the degree $n = 30$ instance took 15 CPU hours). The resulting minimax errors, outputted with 21 digits of accuracy are listed in Table 6.1. For a better conditioning of the computations (see [205, Ch. 25]), we actually computed the (n, n) minimax approximation of

$$e^{9(1-s)/(1+s)}, \qquad s \in (-1, 1], \tag{6.15}$$

which maps to $e^x, x \in (-\infty, 0]$ if we perform the change of variable $x = 9\dfrac{s-1}{s+1}$.

For the exponential function, CF approximation for $e^{9(1-s)/(1+s)}$ seems to be almost indistinguishable from the minimax solution [207], all the while being much faster. The Chebfun double precision `cf` code can

go up to degree $(13, 13)$ on (6.15) without any issues. As a consequence, initializing the Chebfub `remez` and prototype `ratRemez` codes with this approximation leads to convergence in one or two iterations.

6.4 Conclusion & perspectives for future work

While the results we are able to achieve on the absolute value and exponential functions are impressive, we have nonetheless seen in this chapter that the rational exchange algorithm is much tougher to handle than its (weighted) polynomial cousin, usually breaking down for quite small numerator and denominator degrees. Our goal is to try and change this fact, even if it is only in incremental steps, with small ideas like those we advocated in Section 6.2.3 (*i.e.,* better conditioned representations of r_k and an a priori subdivision of the approximation domain to make the search for the next reference more robust).

The difficulties we encountered echo those set forth in [161, Ch. 6.5]:

- *A trial reference leads to no pole free solution in (6.3).* This problem can theoretically be solved if the initial reference is good enough. When CF does not work, how should we proceed? There are two directions that are possibly worth investigating:

  (1) use near-best fixed pole rational interpolants and the roots of Chebyshev rational functions for the starting reference [210–212]. In order to use this approach, some knowledge on the position of the poles of f is needed;

  (2) is there an algorithmic way to take advantage of Potential Theory results on the equilibrium distribution of the equialternating sets of best rational approximations and can this lead to a robust initialization routine? The reference values will tend to cluster near singularities.

- *The ill-conditioned distribution of optimal references.* Because of this reference clustering near singularities, evaluation of r will become complicated, since even the weights in a rational barycentric interpolation will begin to vary exponentially at moderately low degrees, usually well before the best approximation has an error anywhere close to machine precision (this was the case with the degree $(12, 12)$ approximation of $|x|$ in the last example). Can other representations of rational functions be more robust in this case?

- *Numerical problems when solving (6.3).* A possible explanation for this is, as argued in [161, 162, 205], the fast decay of the singular values of the right-hand side matrix of (6.13) which introduces artificial factors in r. What should we do in this case? This difficulty also manifests itself through the fact that determining the $\mathbf{a}$ and $\mathbf{b}$ coefficients of p and q is frequently "ill-posed", in the sense that the computed coefficients may vary quite a lot from the actual ones. In some cases though, the results obtained when evaluating p/q using these perturbed coefficients can be quite accurate due to a so-called phenomenon of error "autocorrelation", which is analyzed for instance by Litvinov [139].

It may well be the case that a fully robust implementation of the rational exchange algorithm is out of reach and other approaches are more fruitful. The other family of methods for computing best rational approximations that have proven to be useful in practice are based on linear programming and sprung from ideas set forth by Cheney and Loeb [57]. They are known as *differential correction* algorithms (details and further references on them can be found for instance in [222, Ch. 9.5], [168, Ch. 10.4] and [36, Ch. 6.6.A]) and have the advantage of guaranteed convergence properties (not conditional as in the case of the rational Remez algorithm). On the other hand, differential correction approaches are slower and seem to be more sensitive to rounding errors than exchange methods [36, Ch. 6.6.B]. Hybrid methods that attempt to combine the benefits of both have also been proposed [20, 117].

Still, the simplicity of the exchange algorithm has made it extremely interesting to study, both from a theoretical and practical point of view. We believe that, at least on the practical side of things, there may still be improvements to be made, [161, Ch. 6] and the current text being evidence for this.

Bibliography

[1] Abu-Al-Saud, W. A., and Stüber, G. L. Efficient wideband channelizer for software radio systems using modulated PR filterbanks. *IEEE Transactions on Signal Processing, 52*, 10 (Oct 2004), 2807–2820.

[2] Achieser, N. I. On extremal properties of certain rational functions. In *Doklady Akad. Nauk* (1930), pp. 495–499.

[3] Achieser, N. I., and Hyman, C. J. *Theory of Approximation*. Dover books on mathematics. Ungar, 1956.

[4] Achterberg, T. SCIP: Solving constraint integer programs. *Mathematical Programming Computation 1*, 1 (Jul 2009), 1–41. http://mpc.zib.de/index.php/MPC/article/view/4.

[5] Adams, J. W., and Wilson, A. N., J. On the fast design of high-order FIR digital filters. *IEEE Transactions on Circuits and Systems 32*, 9 (Sep 1985), 958–960.

[6] Ahsan, M., and Saramäki, T. Two Novel Implementations of the Remez Multiple Exchange Algorithm for Optimum FIR Filter Design. http://www.intechopen.com/download/pdf/39374, 2012.

[7] Ajtai, M. The shortest vector problem in l_2 is NP-hard for randomized reductions (extended abstract). In *Proceedings of the 30th Symposium on the Theory of Computing (STOC 1998)* (1998), ACM, pp. 284–293.

[8] Aksoy, L., Flores, P., and Monteiro, J. Exact and approximate algorithms for the filter design optimization problem. *IEEE Transactions on Signal Processing 63*, 1 (2015), 142–154.

[9] Antoniou, A. Accelerated procedure for the design of equiripple nonrecursive digital filters. *IEE Proceedings G, Electronic Circuits and Systems 129*, 1 (Feb 1982), 1–10.

[10] Antoniou, A. New Improved Method for the Design of Weighted-Chebyshev, Nonrecursive, Digital Filters. *IEEE Transactions on Circuits and Systems 30*, 10 (Oct 1983), 740–750.

[11] Antoniou, A. *Digital Signal Processing: Signals, Systems, and Filters*. McGraw-Hill Education, 2005.

[12] Aptekarev, A. I. Sharp constants for rational approximations of analytic functions. *Sbornik: Mathematics 193*, 1 (2002), 1–72.

[13] Armstrong, R. D., and Kung, D. S. A dual method for discrete Chebychev curve fitting. *Mathematical Programming 19*, 1 (1980), 186–199.

[14] Arora, S., Babai, L., Stern, J., and Sweedyk, Z. The hardness of approximate optima in lattices, codes, and systems of linear equations. In *Foundations of Computer Science, 1993. Proceedings., 34th Annual Symposium on* (1993), IEEE, pp. 724–733.

[15] Babai, L. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica 6*, 1 (1986), 1–13.

[16] Barrodale, I., and Phillips, C. Algorithm 495: Solution of an Overdetermined System of Linear Equations in the Chebychev Norm. *ACM Transactions on Mathematical Software (TOMS) 1*, 3 (1975), 264–270.

[17] Barrodale, I., and Young, A. Algorithms for Best L_1 and L_∞ Linear Approximations on a Discrete Set. *Numerische Mathematik 8*, 3 (1966), 295–306.

[18] Bartels, R. H., and Golub, G. H. Numerical Analysis: Stable numerical methods for obtaining the Chebyshev solution to an overdetermined system of equations. *Communications of the ACM 11*, 6 (1968), 401–406.

[19] Barvinok, A. *A Course in Convexity*, vol. 54 of *Graduate Studies in Mathematics*. American Mathematical Society, 2002.

[20] Belogus, D., and Liron, N. DCR2: An improved algorithm for ℓ_∞ rational approximation on intervals. *Numerische Mathematik 31*, 1 (1978), 17–29.

[21] Bernstein, S. N. Sur l'ordre de la meilleure approximation des fonctions continues par des polynômes de degré donné. *Mémoires de l'Académie Royale de Belgique 4* (1912), 1–104.

[22] Berrut, J.-P., and Mittelmann, H. D. Lebesgue constant minimizing linear rational interpolation of continuous functions over the interval. *Computers & Mathematics with Applications 33*, 6 (1997), 77–86.

[23] Berrut, J.-P., and Mittelmann, H. D. Matrices for the direct determination of the barycentric weights of rational interpolation. *Journal of Computational and Applied Mathematics 78*, 2 (1997), 355–370.

[24] Berrut, J.-P., and Trefethen, L. N. Barycentric Lagrange Interpolation. *SIAM Rev. 46* (2004), 501–517.

[25] Blatt, H.-P. Exchange algorithms, error estimations and strong unicity in convex programming and Chebyshev approximation. In *Approximation Theory and Spline Functions*. Springer, 1984, pp. 23–63.

[26] Blichfeldt, H. F. Note on the Functions of the Form $f(x) \equiv \varphi(x)+a_1x^{n-1}+a_2x^{n-2}+\cdots+a_n$ Which in a Given Interval Differ the Least Possible From Zero. *Transactions of the American Mathematical Society 2*, 1 (1901), 100–102.

[27] Bonzanigo, F. Some improvements to the design programs for equiripple FIR filters. In *IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP'82.* (1982), vol. 7, pp. 274–277.

[28] Borel, É. *Leçons sur les fonctions de variables réelles et les développements en séries de polynomes*. Gauthier-Villars, 1905.

[29] Borwein, P., and Erdélyi, T. *Polynomials and polynomial inequalities*, vol. 161. Springer Science & Business Media, 2012.

[30] Bos, L., De Marchi, S., Sommariva, A., and Vianello, M. Computing multivariate Fekete and Leja points by numerical linear algebra. *SIAM Journal on Numerical Analysis 48*, 5 (2010), 1984–1999.

[31] Bos, L. P., Calvi, J.-P., Levenberg, N., Sommariva, A., and Vianello, M. Geometric weakly admissible meshes, discrete least squares approximations and approximate Fekete points. *Mathematics of Computation 80*, 275 (2011), 1623–1638.

[32] Bos, L. P., and Levenberg, N. On the calculation of approximate Fekete points: the univariate case. *Electronic Transactions on Numerical Analysis 30* (2008), 377–397.

[33] Boyd, J. P. Computing the zeros, maxima and inflection points of Chebyshev, Legendre and Fourier series: solving transcendental equations by spectral interpolation and polynomial rootfinding. *Journal of Engineering Mathematics 56*, 3 (2006), 203–219.

[34] Boyd, J. P. Finding the Zeros of a Univariate Equation: Proxy Rootfinders, Chebyshev Interpolation, and the Companion Matrix. *SIAM Rev. 55*, 2 (2013), 375–396.

[35] Boyd, J. P., and Gally, D. H. Numerical experiments on the accuracy of the Chebyshev-Frobenius companion matrix method for finding the zeros of a truncated series of Chebyshev polynomials. *J. Comput. Appl. Math. 205*, 1 (2007), 281 – 295.

[36] Braess, D. *Nonlinear approximation theory*, vol. 7. Springer Science & Business Media, 2012.

[37] Brent, R. P. *Algorithms for Minimization Without Derivatives*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1973.

[38] Brisebarre, N., and Chevillard, S. Efficient polynomial L^∞-approximations. In *18th IEEE Symposium on Computer Arithmetic, 2007. ARITH '07* (June 2007), pp. 169–176.

[39] Brisebarre, N., de Dinechin, F., Filip, S.-I., and Istoan, M. Automatic generation of hardware FIR filters from a frequency domain specification. https://hal.inria.fr/hal-01308377, 2016. submitted.

[40] Brisebarre, N., Filip, S.-I., and Hanrot, G. De nouveaux résultats sur la synthèse de filtres RIF. In *Proc. 25ème colloque du Groupement de Recherche en Traitement du Signal et des Images (GRETSI), Lyon* (2015).

[41] Brisebarre, N., Filip, S.-I., and Hanrot, G. A Lattice Basis Reduction Approach for the Design of Quantized FIR Filters. *submitted* (2016).

[42] Brisebarre, N., Muller, J.-M., and Tisserand, A. Computing machine-efficient polynomial approximations. *ACM Transactions on Mathematical Software (TOMS) 32*, 2 (2006), 236–256.

[43] Brunie, N., de Dinechin, F., Istoan, M., Sergent, G., Illyes, K., and Popa, B. Arithmetic core generation using bit heaps. In *Field-Programmable Logic and Applications* (Sept. 2013).

[44] Burrus, C. S. Multiband least squares FIR filter design. *IEEE Transactions on Signal Processing 43*, 2 (Feb 1995), 412–421.

[45] Burrus, C. S., Soewito, A. W., and Gopinath, R. A. Least squared error FIR filter design with transition bands. *IEEE Transactions on Signal Processing 40*, 6 (Jun 1992), 1327–1340.

[46] Businger, P., and Golub, G. H. Linear least squares solutions by Householder transformations. *Numerische Mathematik 7*, 3 (1965), 269–276.

[47] Calvi, J.-P., and Levenberg, N. Uniform approximation by discrete least squares polynomials. *Journal of Approximation Theory 152*, 1 (May 2008), 82–100.

[48] Carpenter, A. J., Ruttan, A., and Varga, R. S. Extended numerical computations on the "1/9" conjecture in rational approximation theory. In *Rational approximation and interpolation*. Springer, 1984, pp. 383–411.

[49] Çivril, A., and Magdon-Ismail, M. On selecting a maximum volume sub-matrix of a matrix and related problems. *Theoretical Computer Science 410*, 47-49 (Nov. 2009), 4801–4811.

[50] Celis, O. S. *Practical rational interpolation of exact and inexact data: theory and algorithms*. PhD thesis, Universiteit Antwerpen, 2008.

[51] Chan, D. S. K., and Rabiner, L. R. Analysis of quantization errors in the direct form for finite impulse response digital filters. *IEEE Transactions on Audio and Electroacoustics 21*, 4 (1973), 354–366.

[52] Chan, S. C., Tsui, K. M., and Zhao, S. H. A methodology for automatic hardware synthesis of multiplier-less digital filters with prescribed output accuracy. In *APCCAS 2006 - 2006 IEEE Asia Pacific Conference on Circuits and Systems* (Dec 2006), pp. 61–64.

[53] Chandrasekaran, S., Gu, M., Sun, X., Xia, J., and Zhu, J. A Superfast Algorithm for Toeplitz Systems of Linear Equations. *SIAM Journal on Matrix Analysis and Applications 29*, 4 (2008), 1247–1266.

[54] Chapman, K. Fast integer multipliers fit in FPGAs (EDN 1993 design idea winner). *EDN magazine*, 10 (May 1993), 80.

[55] Chen, Y., and Nguyen, P. Q. BKZ 2.0: Better Lattice Security Estimates. In *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings* (2011), pp. 1–20.

[56] Cheney, E. W. *Introduction to Approximation Theory*. AMS Chelsea Publishing Series. AMS Chelsea Pub., 1982.

[57] Cheney, E. W., and Loeb, H. L. Two new algorithms for rational approximation. *Numerische Mathematik 3*, 1 (1961), 72–75.

[58] Chevillard, S. *Évaluation efficace de functions numériques. Outils et exemples*. PhD thesis, École Normale Supérieure de Lyon, 2009.

[59] Chevillard, S., Harrison, J., Joldeş, M., and Lauter, C. Efficient and accurate computation of upper bounds of approximation errors. *Theoretical Computer Science 412*, 16 (2011), 1523–1543.

[60] Chevillard, S., Joldeş, M., and Lauter, C. Sollya: An Environment for the Development of Numerical Codes. In *Mathematical Software - ICMS 2010* (September 2010), K. Fukuda, J. van der Hoeven, M. Joswig, and N. Takayama, Eds., vol. 6327 of *Lecture Notes in Computer Science*, Springer, pp. 28–31.

[61] Clenshaw, C. W. A note on the summation of Chebyshev series. *Mathematics of Computation 9*, 51 (1955), 118–120.

[62] Cody, W. J. Algorithm 715: SPECFUN–a portable FORTRAN package of special function routines and test drivers. *ACM Transactions on Mathematical Software (TOMS) 19*, 1 (1993), 22–30.

[63] Cody, W. J., Meinardus, G., and Varga, R. Chebyshev rational approximations to e^{-x} in $[0, +\infty)$ and applications to heat-conduction problems. *Journal of Approximation Theory 2*, 1 (1969), 50–65.

[64] Cohen, H. *A course in computational algebraic number theory*, vol. 138. Springer Science & Business Media, 2013.

[65] Cook, W., Koch, T., Steffy, D. E., and Wolter, K. A hybrid branch-and-bound approach for exact rational mixed-integer programming. *Mathematical Programming Computation 5*, 3 (2013), 305–344.

[66] Cortelazzo, G., Lightner, M. R., and Jenkins, W. K. An alternate technique for min-max design of multiband finite impulse response digital filters. *Circuits, Systems and Signal Processing 2*, 3 (1983), 285–309.

[67] Curtis, A., and Osborne, M. R. The construction of minimax rational approximations to functions. *The Computer Journal 9*, 3 (1966), 286–293.

[68] da Silva, E. A., Lovisolo, L., Dutra, A. J., and Diniz, P. S. Fir filter design based on successive approximation of vectors. *IEEE Transactions on Signal Processing 62*, 15 (2014), 3833–3848.

[69] Dagum, L., and Menon, R. OpenMP: an industry standard API for shared-memory programming. *IEEE Computational Science Engineering 5*, 1 (Jan. 1998), 46–55.

[70] Day, D., and Romero, L. Roots of Polynomials Expressed in Terms of Orthogonal Polynomials. *SIAM J. Numer. Anal. 43*, 5 (2005), 1969–1987.

[71] De Dinechin, F., Istoan, M., and Massouri, A. Sum-of-product architectures computing just right. In *2014 IEEE 25th International Conference on Application-specific Systems, Architectures and Processors (ASAP)* (June 2014), pp. 41–47.

[72] de La Vallée-Poussin, C. *Leçons sur l'approximation des fonctions d'une variable réelle*. Gauthier-Villars, 1919.

[73] De Marchi, S., Piazzon, F., Sommariva, A., and Vianello, M. Polynomial meshes: Computation and approximation. In *Proceedings of the 15th International Conference on Computational and Mathematical Methods in Science and Engineering* (2015), pp. 414–425.

[74] Descloux, J. Dégénérescence dans les approximations de Tschebyscheff linéaires et discrètes. *Numerische Mathematik 3*, 1 (1961), 180–187.

[75] DeVore, R. A., and Lorentz, G. G. *Constructive Approximation*, vol. 303. Springer Science & Business Media, 1993.

[76] Diniz, P. S. R., Da Silva, E. A. B., and Netto, S. L. *Digital signal processing: system analysis and design*. Cambridge University Press, 2010.

[77] Dinur, I., Kindler, G., Raz, R., and Safra, S. Approximating CVP to within almost-polynomial factors is NP-hard. *Combinatorica 23*, 2 (2003), 205–243.

[78] Driscoll, T. A., Hale, N., and Trefethen, L. N. *Chebfun Guide*. Pafnuty Publications, Oxford, 2014.

[79] Dutt, A., Gu, M., and Rokhlin, V. Fast algorithms for polynomial interpolation, integration, and differentiation. *SIAM Journal on Numerical Analysis 33*, 5 (1996), 1689–1711.

[80] Ebert, S., and Heute, U. Accelerated design of linear or minimum phase FIR filters with a Chebyshev magnitude response. *IEE Proceedings G, Electronic Circuits and Systems 130*, 6 (Dec 1983), 267–270.

[81] Embree, M., and Trefethen, L. N. Green's Functions for Multiply Connected Domains via Conformal Mapping. *SIAM Rev. 41*, 4 (Dec. 1999), 745–761.

[82] Filip, S.-I. A robust and scalable implementation of the Parks-McClellan algorithm for designing FIR filters. *ACM Transactions on Mathematical Software 43*, 1 (Aug. 2016), 7:1–7:24.

[83] Floater, M. S., and Hormann, K. Barycentric rational interpolation with no poles and high rates of approximation. *Numer. Math. 107*, 2 (2007), 315–331.

[84] Fousse, L., Hanrot, G., Lefèvre, V., Pélissier, P., and Zimmermann, P. MPFR: A Multiple-precision Binary Floating-point Library with Correct Rounding. *ACM Trans. Math. Software 33*, 2 (2007), 13.

[85] Galbraith, S. D. *Mathematics of public key cryptography.* Cambridge University Press, 2012.

[86] Gauss, C. F. Disquisitiones arithmeticae. *Lipsiae in commissis apud Gerh. Fleischer Jux.* (1801).

[87] Gautschi, W. Computational Methods in Special Functions–A Survey. In *Theory and Application of Special functions*, R. A. Askey, Ed. Academic Press, New York, 1975, pp. 1–98.

[88] Gautschi, W. *Orthogonal polynomials: computation and approximation.* Oxford University Press, 2004.

[89] Gleixner, A. M., Steffy, D. E., and Wolter, K. Improving the accuracy of linear programming solvers with iterative refinement. In *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation* (2012), ACM, pp. 187–194.

[90] Gleixner, A. M., Steffy, D. E., and Wolter, K. Iterative Refinement for Linear Programming. Tech. Rep. 15-15, ZIB, Takustr.7, 14195 Berlin, 2015.

[91] Goldreich, O., Micciancio, D., Safra, S., and Seifert, J.-P. Approximating shortest lattice vectors is not harder than approximating closest lattice vectors. *Information Processing Letters 71*, 2 (1999), 55–61.

[92] Gonchar, A. A., and Rakhmanov, E. A. Equilibrium distributions and degree of rational approximation of analytic functions. *Mathematics of the USSR-Sbornik 62*, 2 (1989), 305–348.

[93] Grenez, F. Design of linear or minimum-phase FIR filters by constrained Chebyshev approximation. *Signal Processing 5*, 4 (1983), 325–332.

[94] Guennebaud, G., Jacob, B., et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[95] Gurobi Optimization, I. Gurobi Optimizer Reference Manual. http://www.gurobi.com, 2015.

[96] Gustafsson, O., Johansson, H., and Wanhammar, L. An MILP approach for the design of linear-phase FIR filters with minimum number of signed-power-of-two terms. In *Proc. Eur. Conf. Circuit Theory Design* (2001), vol. 2, pp. 217–220.

[97] Hanrot, G., Pujol, X., and Stehlé, D. Analyzing Blockwise Lattice Algorithms Using Dynamical Systems. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings* (2011), pp. 447–464.

[98] Hart, J. F., Cheney, E. W., Lawson, C. L., Maehly, H. J., Mesztenyi, C. K., Rice, J. R., Thacher, Jr., H. G., and Witzgall, C. *Computer Approximations.* Wiley, New York, 1968.

[99] Henk, M. Note on shortest and nearest lattice vectors. *Information Processing Letters 61*, 4 (1997), 183–188.

[100] Hennessy, J., Patterson, D., and Asanović, K. *Computer Architecture: A Quantitative Approach*, 5 ed. Morgan Kaufmann/Elsevier, 2012.

[101] Hermite, C. Extraits de lettres de M. Ch. Hermite à M. Jacobi sur différents objects de la théorie des nombres.(Continuation). *Journal für die reine und angewandte Mathematik 40* (1850), 279–315.

[102] Higham, N. J. *Accuracy and stability of numerical algorithms.* SIAM, 2002.

[103] Higham, N. J. The numerical stability of barycentric Lagrange interpolation. *IMA J. Numer. Anal. 24* (2004), 547–556.

[104] Hopper, M. J., and Powell, M. J. D. A technique that gains speed and accuracy in the minimax solution of overdetermined linear equations. *Mathematical Software 3* (1977), 15–32.

[105] Hwang, J., and Ballagh, J. Building custom FIR filters using system generator. In *Proceedings of the Reconfigurable Computing Is Going Mainstream, 12th International Conference on Field-Programmable Logic and Applications* (2002), FPL '02, pp. 1101–1104.

[106] IEEE Task P754. *ANSI/IEEE 754-1985, Standard for Binary Floating-Point Arithmetic.* IEEE, New York, NY, USA, Aug. 1985.

[107] IEEE Task P754. *IEEE 754-2008, Standard for Floating-Point Arithmetic.* IEEE, New York, NY, USA, Aug. 2008.

[108] Ishikawa, M., Edahiro, M., Yoshimura, T., Miyazaki, T., Aikoh, S. I., Nishitani, T., Mitsuhashi, K., and Furuichi, M. Automatic layout synthesis for FIR filters using a silicon compiler. In *Circuits and Systems, 1990., IEEE International Symposium on* (May 1990), pp. 2588–2591.

[109] Jackson, D. *Über die Genauigkeit der Annäherung stetiger Funktionen durch ganze rationale Funktionen gegebenen Grades und trigonometrische Summen gegebener Ordnung.* Dieterich, 1911.

[110] Jain, R., Yang, P. T., and Yoshino, T. FIRGEN: a computer-aided design system for high performance FIR filter integrated circuits. *IEEE Transactions on Signal Processing 39*, 7 (Jul 1991), 1655–1668.

[111] Javed, M., and Trefethen, L. N. The Remez algorithm for trigonometric approximation of periodic functions. *submitted* (2015).

[112] Kaiser, J. F. Some practical considerations in the realization of linear digital filters. In *Proc. 3rd Allerton Annual Conference on Circuit and System Theory* (1965), pp. 621–633.

[113] Kannan, R. Improved algorithms for integer programming and related lattice problems. In *Proceedings of the 15th Symposium on the Theory of Computing (STOC 1983)* (1983), ACM, pp. 99–108.

[114] Kannan, R. Minkowski's Convex Body Theorem and Integer Programming. *Mathematics of Operations Research 12*, 3 (Aug 1987), 415–440.

[115] Karam, L. J., and McClellan, J. H. Complex Chebyshev approximation for FIR filter design. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing 42*, 3 (Mar 1995), 207–216.

[116] Karam, L. J., and McClellan, J. H. Design of optimal digital FIR filters with arbitrary magnitude and phase responses. In *ISCAS '96., IEEE International Symposium on Circuits and Systems* (May 1996), vol. 2, pp. 385–388.

[117] Kaufman, E. H., Leeming, D. J., and Taylor, G. D. A combined Remes-differential correction algorithm for rational approximation. *Mathematics of Computation 32*, 141 (1978), 233–242.

[118] Kirchberger, P. *Ueber Tchebychefsche Annäherungsmethoden.* PhD thesis, Götingen, 1902.

[119] Kodek, D., and Steiglitz, K. Comparison of optimal and local search methods for designing finite wordlength FIR digital filters. *IEEE Transactions on Circuits and Systems 28*, 1 (1981), 28–32.

[120] Kodek, D. M. Design of optimal finite wordlength FIR digital filters using integer programming techniques. *IEEE Transactions on Acoustics, Speech and Signal Processing 28*, 3 (June 1980), 304–308.

[121] Kodek, D. M. Design of optimal finite wordlength FIR digital filters. In *Proc. of the European Conference on Circuit Theory and Design, ECCTD '99.* (Aug 1999), vol. 1, pp. 401–404.

[122] Kodek, D. M. Performance limit of finite wordlength FIR digital filters. *IEEE Transactions on Signal Processing 53*, 7 (Jul 2005), 2462–2469.

[123] Kodek, D. M. LLL Algorithm and the Optimal Finite Wordlength FIR Design. *IEEE Transactions on Signal Processing 60*, 3 (Mar. 2012), 1493–1498.

[124] Kodek, D. M., and Krisper, M. Telescoping rounding for suboptimal finite wordlength FIR digital filter design. *Digital Signal Processing 15*, 6 (2005), 522–535.

[125] Kogut, J. B., and Sinclair, D. K. Rational hybrid Monte Carlo algorithm for theories with unknown spectral bounds. *Physical Review D 74*, 11 (2006), 114505.

[126] Lagarias, J. C., Lenstra Jr, H. W., and Schnorr, C.-P. Korkin-Zolotarev bases and successive minima of a lattice and its reciprocal lattice. *Combinatorica 10*, 4 (1990), 333–348.

[127] Lagrange, J. L. Recherches d'arithmetique. *Nouv. Mem. Acad. Roy. Sc. Belles Lettres Berlin* (1773), 365–312.

[128] Langhammer, M., and Pasca, B. Floating-Point DSP Block Architecture for FPGAs. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (New York, NY, USA, 2015), FPGA '15, ACM, pp. 117–125.

[129] Laurent, P. J., and Carasso, C. An algorithm of successive minimization in convex programming. *RAIRO-Analyse numérique 12*, 4 (1978), 377–400.

[130] Lauter, C., and de Dinechin, F. Optimising polynomials for floating-point implementation. In *Real Numbers and Computers* (2008), pp. 7–16.

[131] Leja, F. Sur certaines suites liées aux ensembles plans et leur application à la représentation conforme. *Annales Polonici Mathematici 4*, 1 (1957), 8–13.

[132] Lenstra, A. K., Lenstra, Jr., H. W., and Lovász, L. Factoring polynomials with rational coefficients. *Math. Ann. 261* (1982), 515–534.

[133] Levin, E., and Saff, E. B. Potential Theoretic Tools in Polynomial and Rational Approximation. In *Harmonic Analysis and Rational Approximation*, J.-D. Fournier, J. Grimm, J. Leblond, and J. R. Partington, Eds., vol. 327 of *Lecture Notes in Control and Information Science*. Springer Berlin Heidelberg, 2006, pp. 71–94.

[134] Lim, Y. C. Design of discrete-coefficient-value linear phase FIR filters with optimum normalized peak ripple magnitude. *IEEE Transactions on Circuits and Systems, 37*, 12 (Dec 1990), 1480–1486.

[135] Lim, Y. C., and Parker, S. R. Discrete coefficient FIR digital filter design based upon an LMS criteria. *IEEE Transactions on Circuits and systems 30*, 10 (1983), 723–739.

[136] Lim, Y. C., and Parker, S. R. FIR filter design over a discrete powers-of-two coefficient space. *IEEE Transactions on Acoustics, Speech and Signal Processing 31*, 3 (June 1983), 583–591.

[137] Lim, Y. C., Parker, S. R., and Constantinides, A. G. Finite word length FIR filter design using integer programming over a discrete coefficient space. *IEEE Transactions on Acoustics, Speech and Signal Processing 30*, 4 (Aug 1982), 661–664.

[138] Lim, Y. C., Sun, Y., and Yu, Y. J. Design of discrete-coefficient FIR filters on loosely connected parallel machines. *IEEE Transactions on Signal Processing 50*, 6 (2002), 1409–1416.

[139] Litvinov, G. Approximate construction of rational approximations and the effect of error autocorrection. Applications. *Russian Journal of Mathematical Physics 1*, 3 (1994), 313–352. see also http://arxiv.org/abs/math/0101042.

[140] Lotze, J., Fahmy, S. A., Noguera, J., Doyle, L., and Esser, R. An FPGA-based cognitive radio framework. In *Irish Signals and Systems Conference (ISSC 2008)* (June 2008), IET, pp. 138–143.

[141] Maehly, H. J. Methods for fitting rational approximations, Parts II and III. *Journal of the ACM (JACM) 10*, 3 (1963), 257–277.

[142] Marlow, R., Dobson, C., and Athanas, P. An enhanced and embedded GNU radio flow. In *Field Programmable Logic and Applications (FPL)* (Sept. 2014), IEEE.

[143] Mascarenhas, W. F. The stability of barycentric interpolation at the Chebyshev points of the second kind. *Numer. Math. 128*, 2 (2014), 265–300.

[144] Mascarenhas, W. F., and Camargo, A. On the backward stability of the second barycentric formula for interpolation. *Dolomites Research Notes on Approximation 7* (2014), 1–12.

[145] Mascarenhas, W. F., and de Camargo, A. P. The effects of rounding errors in the nodes on barycentric interpolation. *Numerische Mathematik* (2013), 1–29.

[146] Mason, J. C., and Handscomb, D. C. *Chebyshev polynomials*. CRC Press, 2002.

[147] McCallig, M. T., and Leon, B. J. Constrained ripple design of FIR digital filters. *IEEE Transactions on Circuits and Systems 25*, 11 (1978), 893–902.

[148] McClellan, J. H., Parks, T. W., and Rabiner, L. A computer program for designing optimum FIR linear phase digital filters. *IEEE Transactions on Audio and Electroacoustics 21*, 6 (Dec 1973), 506–526.

[149] Merchant, G., and Parks, T. W. Efficient solution of a Toeplitz-plus-Hankel coefficient matrix system of equations. *IEEE Transactions on Acoustics, Speech and Signal Processing 30*, 1 (Feb 1982), 40–44.

[150] Meyer-Baese, U. *Digital signal processing with field programmable gate arrays*, vol. 65 of *Signals and Communication Technology*. Springer, 2007.

[151] Micciancio, D., and Goldwasser, S. *Complexity of lattice problems: a cryptographic perspective*, vol. 671. Springer Science & Business Media, 2012.

[152] Minkowski, H. *Geometrie der Zahlen*. BG Teubner (Leipzig), 1910.

[153] Muller, J. M., Brisebarre, N., de Dinechin, F., Jeannerod, C. P., Lefèvre, V., Melquiond, G., Revol, N., Stehlé, D., and Torres, S. *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston, 2009.

[154] Nguyen, P., and Stehlé, D. LLL on the average. In *Proceedings of the 7th Algorithmic Number Theory Symposium (ANTS VII)* (2006), vol. 4076 of *LNCS*, Springer, pp. 238–256.

[155] Nguyen, P. Q., and Stehlé, D. Floating-point LLL revisited. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2005), Springer, pp. 215–233.

[156] Nguyen, P. Q., and Vallée, B., Eds. *The LLL Algorithm - Survey and Applications*. Information Security and Cryptography. Springer, 2010.

[157] Nielsen, J. J. Design of linear-phase direct-form FIR digital filters with quantized coefficients using error spectrum shaping. *IEEE Transactions on Acoustics, Speech and Signal Processing 37*, 7 (Jul 1989), 1020–1026.

[158] Oppenheim, A. V., and Schafer, R. W. *Discrete-Time Signal Processing*. Prentice-Hall Signal Processing Series. Prentice Hall, 2010.

[159] Osborne, M. R., and Watson, G. A. On the best linear Chebyshev approximation. *The Computer Journal 10*, 2 (1967), 172–177.

[160] Osborne, M. R., and Watson, G. A. A note on singular minimax approximation problems. *Journal of Mathematical Analysis and Applications 25*, 3 (1969), 692–700.

[161] Pachón, R. *Algorithms for Polynomial and Rational Approximation*. PhD thesis, University of Oxford, 2010.

[162] Pachón, R., Gonnet, P., and Van Deun, J. Fast and stable rational interpolation in roots of unity and Chebyshev points. *SIAM Journal on Numerical Analysis 50*, 3 (2012), 1713–1734.

[163] Pachón, R., and Trefethen, L. N. Barycentric-Remez algorithms for best polynomial approximation in the Chebfun system. *BIT Numerical Mathematics 49*, 4 (2009), 721–741.

[164] Parks, T. W., and McClellan, J. H. Chebyshev Approximation for Nonrecursive Digital Filters with Linear Phase. *IEEE Transactions on Circuit Theory 19*, 2 (March 1972), 189–194.

[165] Parlett, B. N., and Reinsch, C. Balancing a matrix for calculation of eigenvalues and eigenvectors. *Numerische Mathematik 13*, 4 (1969), 293–304.

[166] Pelloni, D., and Bonzanigo, F. On the design of high-order linear phase FIR filters. *Digital Signal Processing 1* (1980), 3–10.

[167] Piazzon, F., and Vianello, M. On the stability of Lebesgue constants. www.math.unipd.it/~marcov/pdf/leb.pdf, 2016.

[168] Powell, M. J. D. *Approximation Theory and Methods*. Cambridge University Press, 1981.

[169] Prandoni, P., and Vetterli, M. *Signal processing for communications*. CRC Press, 2008.

[170] Psarakis, E. Z., and Moustakides, G. V. A robust initialization scheme for the Remez exchange algorithm. *IEEE Signal Processing Letters 10*, 1 (Jan 2003), 1–3.

[171] Pusa, M. *Numerical methods for nuclear fuel burnup calculations*. PhD thesis, Aalto University, Greater Helsinki, Finland, 2013.

[172] Rabiner, L., Kaiser, J., and Schafer, R. W. Some Considerations in the Design of Multiband Finite-Impulse-Response Digital Filters. *IEEE Transactions on Acoustics, Speech and Signal Processing 22*, 6 (Dec 1974), 462–472.

[173] Rack, H.-J., and Reimer, M. The numerical stability of evaluation schemes for polynomials based on the Lagrange interpolation form. *BIT Numerical Mathematics 22*, 1 (1982), 101–107.

[174] Ralston, A. Some aspects of degeneracy in rational approximations. *IMA Journal of Applied Mathematics 11*, 2 (1973), 157–170.

[175] Reichel, L. Newton interpolation at Leja points. *BIT 30*, 2 (1990), 332–346.

[176] Remes, E. Sur le calcul effectif des polynômes d'approximation de Tchebichef. *Comptes rendus hebdomadaires des séances de l'Académie des Sciences 199* (1934), 337–340.

[177] Remes, E. Sur un procédé convergent d'approximations successives pour déterminer les polynômes d'approximation. *Comptes rendus hebdomadaires des séances de l'Académie des Sciences 198* (1934), 2063–2065.

[178] Runge, C. Über empirische Funktionen und die Interpolation zwischen äquidistanten Ordinaten. *Zeitschrift für Mathematik und Physik 46* (1901), 224–243.

[179] Rutishauser, H. *Vorlesungen über Numerische Mathematik*. Birkhäuser Basel, Stuttgard, 1976. English translation, 1990. *Lectures on Numerical Mathematics*. Walter Gautschi, ed., Birkhäuser Boston.

[180] Salzer, H. E. Lagrangian Interpolation at the Chebyshev points $x_{n,\nu} \equiv \cos(\nu\pi/n), \nu = 0(1)n$; some unnoted advantages. *The Computer Journal 15*, 2 (1972), 156–159.

[181] Saramäki, T. An efficient Remez-type algorithm for the design of optimum IIR filters with arbitrary partially constrained specifications. In *ISCAS '92., IEEE International Symposium on Circuits and Systems* (May 1992), vol. 5, pp. 2577–2580.

[182] Saramäki, T. Generalizations of classical recursive digital filters and their design with the aid of a Remez-type algorithm. In *ISCAS '94., IEEE International Symposium on Circuits and Systems* (May 1994), vol. 2, pp. 549–552.

[183] Schneider, C., and Werner, W. Some new aspects of rational interpolation. *Mathematics of Computation 47*, 175 (1986), 285–299.

[184] Schnorr, C. P. A Hierarchy of Polynomial Lattice Basis Reduction Algorithms. *Theoretical Computer Science 53* (1987), 201–224.

[185] Schönhage, A. Zur rationalen Approximierbarkeit von e^{-x} über $[0, \infty$. *Journal of Approximation Theory 7*, 4 (1973), 395–398.

[186] Selesnick, I. W. New exchange rules for IIR filter design. In *ICASSP-97., IEEE International Conference on Acoustics, Speech, and Signal Processing* (Apr 1997), vol. 3, pp. 2209–2212.

[187] Selesnick, I. W., Lang, M., and Burrus, C. S. Magnitude squared design of recursive filters with the Chebyshev norm using a constrained rational Remez algorithm. In *Sixth IEEE Digital Signal Processing Workshop,* (Oct 1994), pp. 23–26.

[188] Shen, J., and Strang, G. The asymptotics of optimal (equiripple) filters. *IEEE Transactions on Signal Processing 47*, 4 (Apr. 1999), 1087–1098.

[189] Shen, J., Strang, G., and Wathen, A. J. The Potential Theory of Several Intervals and Its Applications. *Applied Mathematics and Optimization 44*, 1 (Jan. 2001), 67–85.

[190] Shi, D., and Yu, Y. J. Design of Linear Phase FIR Filters With High Probability of Achieving Minimum Number of Adders. *IEEE Transactions on Circuits and Systems I: Regular Papers 58*, 1 (Jan 2011), 126–136.

[191] Shpak, D. J., and Antoniou, A. A generalized Remez method for the design of FIR digital filters. *IEEE Transactions on Circuits and Systems 37*, 2 (Feb 1990), 161–174.

[192] Skaf, J., and Boyd, S. P. Filter Design With Low Complexity Coefficients. *IEEE Transactions on Signal Processing 56*, 7 (Jul 2008), 3162–3169.

[193] Smoktunowicz, A. Backward stability of Clenshaw's algorithm. *BIT Numerical Mathematics 42*, 3 (2002), 600–610.

[194] Smyk, R. FIReWORK: FIR filters hardware structures auto-generator. *Journal of Applied Computer Science 21*, 1 (2013), 135–149.

[195] Sommariva, A., and Vianello, M. Computing approximate Fekete points by QR factorizations of Vandermonde matrices. *Computers & Mathematics with Applications 57*, 8 (Apr. 2009), 1324–1336.

[196] Sommariva, A., and Vianello, M. Approximate Fekete points for weighted polynomial interpolation. *Electronic Transactions on Numerical Analysis 37* (2010), 1–22.

[197] Stahl, G. Best uniform approximation of $|x|$ on $[-1, 1]$. *Russian Academy of Sciences. Sbornik Mathematics 76*, 2 (1993), 461–487.

[198] Stiefel, E. Über diskrete und lineare Tschebyscheff-Approximationen. *Numerische Mathematik 1*, 1 (1959), 1–28.

[199] Stiefel, E. Note on Jordan elimination, linear programming and Tchebycheff approximation. *Numerische Mathematik 2*, 1 (1960), 1–17.

[200] Stieltjes, T. J. Sur les polynômes de Jacobi. *Comptes rendus hebdomadaires des séances de l'Académie des Sciences 100* (1885), 620–622.

[201] Strang, G. The discrete cosine transform. *SIAM Rev. 41*, 1 (1999), 135–147.

[202] Taylor, F. J. Residue arithmetic a tutorial with examples. *Computer 17*, 5 (1984), 50–62.

[203] Tchebycheff, P. L. *Sur les questions de minima qui se rattachent à la représentation aproximative des fonctions.* Mémoires de l'Académie impériale des Sciences de Saint-Pétersbourg, 1859.

[204] The FPLLL development team. fplll, a lattice reduction library. Available at https://github.com/fplll/fplll, 2016.

[205] Trefethen, L. N. *Approximation Theory and Approximation Practice.* SIAM, 2013.

[206] Trefethen, L. N., and Gutknecht, M. H. The Carathéodory-Fejér method for real rational approximation. *SIAM Journal on Numerical Analysis 20*, 2 (1983), 420–436.

[207] Trefethen, L. N., Weideman, J. A. C., and Schmelzer, T. Talbot quadratures and rational approximations. *BIT Numerical Mathematics 46*, 3 (2006), 653–670.

[208] Turán, P. On some open problems of approximation theory. *Journal of Approximation Theory 29*, 1 (1980), 23–85.

[209] van de Pol, J. *Lattice-based cryptanalysis.* PhD thesis, University of Bristol, 2015.

[210] Van Deun, J. Eigenvalue problems to compute almost optimal points for rational interpolation with prescribed poles. *Numerical Algorithms 45*, 1-4 (2007), 89–99.

[211] Van Deun, J. Computing near-best fixed pole rational interpolants. *Journal of Computational and Applied Mathematics 235*, 4 (2010), 1077–1084.

[212] Van Deun, J., Deckers, K., Bultheel, A., and Weideman, J. Algorithm 882: Near-best fixed pole rational interpolation with applications in spectral methods. *ACM Transactions on Mathematical Software (TOMS) 35*, 2 (2008), 14.

[213] Van Deun, J., and Trefethen, L. N. A robust implementation of the Carathéodory-Fejér method for rational approximation. *BIT Numerical Mathematics 51*, 4 (2011), 1039–1050.

[214] van Emde Boas, P. Another NP-complete problem and the complexity of computing short vectors in a lattice. Tech. Rep. 8104, University of Amsterdam, Department of Mathematics, Netherlands, 1981.

[215] Varga, R. S., and Carpenter, A. J. On the Bernstein conjecture in approximation theory. *Constructive Approximation 1*, 1 (1985), 333–348.

[216] Varga, R. S., Ruttan, A., and Carpenter, A. J. Numerical results on best uniform approximation of $|x|$ on $[-1, +1]$. *Mathematics of the USSR-Sbornik 74*, 2 (1993), 271–290.

[217] Veidinger, L. On the numerical determination of the best approximation in the Chebyshev sense. *Numer. Math. 2*, 1 (1960), 99–105.

[218] Verdoolaege, S. isl: An integer set library for the polyhedral model. In *International Congress on Mathematical Software* (2010), Springer, pp. 299–302.

[219] Vetterli, M., Kovačević, J., and Goyal, V. K. *Foundations of signal processing.* Cambridge University Press, 2014.

[220] Walsh, J. L. The existence of rational functions of best approximation. *Transactions of the American Mathematical Society 33*, 3 (1931), 668–689.

[221] Walsh, J. L. On approximation to an analytic function by rational functions of best approximation. *Mathematische Zeitschrift 38*, 1 (1934), 163–176.

[222] Watson, G. A. *Approximation theory and numerical methods.* Wiley, 1980.

[223] Watson, G. A. Approximation in normed linear spaces. *Journal of Computational and Applied Mathematics 121*, 1 (2000), 1–36.

[224] Webb, M., Trefethen, L. N., and Gonnet, P. Stability of Barycentric Interpolation Formulas for Extrapolation. *SIAM J. Scientific Computing 34*, 6 (2012), A3009–A3015.

[225] Wei, D. *Design of discrete-time filters for efficient implementation.* PhD thesis, Massachusetts Institute of Technology, 2011.

[226] Weinstein, S. E. Approximations of functions of several variables: product Chebychev approximations, I. *Journal of Approximation Theory 2*, 4 (1969), 433–447.

[227] Werner, H. Die konstruktive Ermittlung der Tschebyscheff-Approximierenden im Bereich der rationalen funktionen. *Archive for Rational Mechanics and Analysis 11*, 1 (1962), 368–384.

[228] Werner, H., Stoer, J., and Bommas, W. Rational Chebyshev approximation. *Numerische Mathematik 10*, 4 (1967), 289–306.

[229] Widrow, B., and Kollár, I. *Quantization noise.* Cambridge University Press, 2008.

[230] Wipf, A. *Statistical Approach to Quantum Field Theory: An Introduction*, 2013 ed., vol. 864 of *Lecture Notes in Physics.* Springer, 2012.

[231] Wright, G. B., Javed, M., Montanelli, H., and Trefethen, L. N. Extension of Chebfun to periodic functions. *SIAM Journal on Scientific Computing 37*, 5 (2015), C554–C573.

[232] Yang, L., and Giannakis, G. B. Ultra-wideband communications: an idea whose time has come. *IEEE Signal Processing Magazine, 21*, 6 (2004), 26–54.

[233] Yang, P. T., Jain, R., Yoshino, T., Gass, W., and Shah, A. A functional silicon compiler for high speed FIR digital filters. In *Acoustics, Speech, and Signal Processing, 1990. ICASSP-90., 1990 International Conference on* (Apr 1990), pp. 1329–1332.

[234] Yli-Kaakinen, J., and Saramaki, T. A systematic algorithm for the design of lattice wave digital filters with short-coefficient wordlength. *IEEE Transactions on Circuits and Systems I: Regular Papers 54*, 8 (2007), 1838–1851.

[235] Zahradnik, P., and Vlcek, M. Robust analytical design of equiripple comb FIR filters. In *IEEE International Symposium on Circuits and Systems, 2008. ISCAS 2008* (May 2008), pp. 1128–1131.